

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

ПРИЛОЖЕНИЕ ПО УЛУЧШЕНИЮ

КАЧЕСТВА ЖИЗНИ

Студент

Матюшин Кирилл Михайлович
(подпись, дата)

Матюшин Кирилл Михайлович
(ФБЮ полностью)

Руководитель

Кузнецов А.В.
(подпись, дата) 02.06.2026

Кузнецов А.В.
(фамилия, инициалы)

Нормоконтролёр

Бычков А.С.
(подпись, дата) 02.06.2026

Бычков А.С.
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.2026 г. № 14)

Заведующий кафедрой

М.А. Митрохин
(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой удовлетворительно (протокол заседания ГЭК от

15 июля 2026 г. № 29)

Секретарь ГЭК

Е

(подпись)

Е.И. Гурин

(фамилия, инициалы)

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«04» 04 2026 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Матюшин Кирилл Михайлович гр. 22ВВ12 факультета вычислительной техники
2. Тема работы: Приложение по улучшению качества жизни

Тема утверждена приказом ПГУ № 347/0 от "04" апреля 2026 г.

3. Руководитель бакалаврской работы Кузнецов А.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать кроссплатформенное мобильное приложение для отслеживания качества жизни. Приложение должно выполнять функции комплексного учёта воды, питания, сна, веса и привычек, систему достижений, визуализацию данных, экспорт в CSV/PDF и резервное копирование.

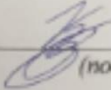
5. Перечень подлежащих разработке вопросов

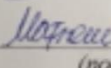
Требуется произвести анализ предметной области, изучить существующие аналоги, определить архитектуру приложения, выбрать инструменты для разработки, реализовать и провести тестирование приложения.

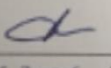
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	30	17.04.2026 – 22.04.2026	
2	Анализ требований	24	23.04.2026 – 25.04.2026	
3	Выбор технологий, применяемых при проектировании и разработке	40	26.04.2026 – 30.04.2026	
4	Выбор архитектуры приложения	30	01.05.2026 – 03.05.2026	
5	Разработка приложения	126	04.05.2026 – 22.05.2026	
6	Тестирование и отладка	50	23.05.2026 – 28.05.2026	
7	Оформление пояснительной записки	24	29.05.2026 – 02.06.2026	

Дата выдачи задания " 07 " апреля 2026 г.

Руководитель бакалаврской работы  27.05.2026
(подпись, дата)

Задание к исполнению принял студент  27.05.2026
(подпись, дата)

Работу к защите допустить
Декан факультета  Л.Р.Финцова
10.06.2026 (подпись, дата)

Реферат 14

Бакалаврская работа содержит 158 листов, 41 рисунок, 1 таблицу, 14 листингов, 7 источников, 3 приложения.

КРОССПЛАТФОРМЕННОЕ ПРИЛОЖЕНИЕ, КАЧЕСТВО ЖИЗНИ, ТРЕКЕР ВОДЫ, ПИТАНИЯ, СНА, ВЕСА, ПРИВЫЧКИ, ДОСТИЖЕНИЯ, Flutter, SQLite

Цель работы – разработка кроссплатформенного мобильного приложения «StatLife» для комплексного контроля качества жизни.

В процессе работы проведён анализ предметной области и аналогов, выбрана архитектура на Flutter и SQLite, реализованы трекеры воды, питания, сна, веса и привычек, система достижений, графики, календари, экспорт в CSV/PDF, резервное копирование и уведомления.

Разработанное приложение работает на Android и iOS, позволяет централизованно управлять показателями здоровья, отслеживать прогресс и получать мотивационную поддержку

					ПГУ1.090301.43.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Матвишин К.М.		02.05.24	Лит.	Лист	Листов
Провер.		Кузнецов А.В.		02.05.24		2	158
Н. Контр		Басков А.С.		02.05.24	ФВТ, каф. ВТ гр. 22ВВП2		
Утверд.		Михралиев М.А.		02.05.24			
					Приложение по улучшению качества жизни Пояснительная записка		

Содержание

Перечень сокращений и обозначений.....	5
Введение.....	6
1 Приложение по улучшению качества жизни	9
1.1 Анализ предметной области	9
1.2 Обзор аналогов	14
1.3 Анализ требований к разработке.....	19
1.4 Выбор архитектуры системы	25
1.5 Выбор языка программирования.....	33
1.6 Выбор среды разработки	34
1.7 Проектирование и разработка	41
1.8 Разработка управляющей программы.....	45
2 Демонстрация работы приложения.....	49
2.1 Регистрация и вход.....	49
2.2 Добавление записи о воде	50
2.3 Добавление приёма пищи.....	51
2.4 Отслеживание сна	52
2.5 Работа с привычками	53
2.6 Получение достижения.....	54
2.7 Экспорт данных.....	55
2.8 Заключение по демонстрации.....	55
3 Общие требования безопасности.....	57
3.1 Требования безопасности при работе за ПК.....	57
3.2 Требования безопасности при работе с мобильным телефоном.....	58
3.3 Меры пожарной безопасности.....	59
Заключение	61
Список использованных источников	62
Приложение А	63

__Приложение Б.....	71
__Приложение В.....	154

					ПГУ1.090301.43.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		4

Перечень сокращений и обозначений

БД – база данных

БЖУ – белки, жиры, углеводы

ОС – операционная система

ПГУ – Пензенский государственный университет

ПК – персональный компьютер

ПО – программное обеспечение

РПЗ – расчётно-пояснительная записка

СУБД – система управления базами данных

API (Application Programming Interface) – программный интерфейс приложения

CSV (Comma-Separated Values) – формат данных с разделителями-запятыми

Dart – язык программирования, разработанный Google

Flutter – фреймворк для кроссплатформенной разработки от Google

IDE (Integrated Development Environment) – интегрированная среда разработки

JIT (Just-In-Time) – компиляция «точно в время»

JSON (JavaScript Object Notation) – текстовый формат обмена данными

mHealth (mobile Health) – мобильное здравоохранение

MVVM (Model-View-ViewModel) – архитектурный паттерн «модель-представление-модель представления»

PDF (Portable Document Format) – портативный формат документов

SHA-256 – алгоритм криптографического хеширования

SQL (Structured Query Language) – язык структурированных запросов

SQLite – встраиваемая реляционная система управления базами данных

UML (Unified Modeling Language) – унифицированный язык моделирования

URL (Uniform Resource Locator) – единообразный указатель ресурса

Введение

В современном мире автоматизация процессов становится не просто преимуществом, а необходимостью для любой сферы деятельности, особенно когда речь идет о заботе о здоровье и качестве жизни. Одной из таких сфер является управление личным здоровьем, которое в последние годы переживает активный рост благодаря цифровизации. Количество людей, следящих за своим здоровьем, увеличивается, а вместе с ним растет и потребность в удобных инструментах для контроля физиологических показателей. В таких условиях пользователи сталкиваются с рядом проблем, связанных с ведением учёта воды, питания, сна, веса и привычек. Традиционные способы контроля, основанные на бумажных дневниках, заметках в телефоне или электронных таблицах, перестают быть эффективными при большом количестве параметров и необходимости долгосрочного отслеживания динамики.

Проблема ручного сбора и оформления статистики особенно остро проявляется в процессе отслеживания прогресса по привычкам. Когда пользователь хочет контролировать, например, отказ от курения или соблюдение водного баланса, он вынужден вручную записывать каждый приём воды, каждую съеденную калорию, каждую минуту сна. Такой подход требует значительных временных затрат. Кроме того, при ручной обработке данных неизбежно возникают ошибки: забытая запись, неверно указанный объём или время, потеря информации. Всё это создаёт дополнительные сложности и снижает мотивацию пользователя продолжать вести учёт.

Не менее важной является и задача анализа прогресса. Пользователю необходимо понимать, какие привычки уже сформировались, какие показатели улучшились, а какие требуют дополнительного внимания. Без наглядной визуализации данных невозможно эффективно корректировать свой образ жизни. Традиционное решение — сбор информации вручную с последующим переносом в электронные таблицы — является крайне

					ПГУ1.090301.43.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		6

трудоёмким и подверженным ошибкам. Кроме того, такой подход не позволяет получать оперативную информацию в реальном времени и не предоставляет мотивационной поддержки в виде достижений и серий.

Таким образом, назревает объективная необходимость в создании автоматизированной системы, которая позволила бы централизованно управлять всеми показателями здоровья, обрабатывать записи пользователя в автоматическом режиме, контролировать прогресс по привычкам, а также формировать наглядную отчётность в виде графиков и календарей. Внедрение такого программного решения позволит пользователю выйти на новый уровень контроля над своим здоровьем, повысить эффективность формирования полезных привычек и получить мотивационную поддержку для достижения целей.

Целью данной работы является проектирование и разработка современного кроссплатформенного мобильного приложения «StatLife» для комплексного контроля качества жизни. Создаваемая система должна обеспечить удобный и интуитивно понятный интерфейс для пользователя. Необходимо реализовать возможность учёта воды, питания, сна, веса и привычек, а также систему достижений, визуализацию данных в виде графиков и календарей, экспорт информации в форматы CSV и PDF, резервное копирование базы данных и локальные уведомления.

Особое внимание при разработке уделяется вопросам автономности и конфиденциальности. Система должна работать полностью без подключения к интернету, гарантировать сохранность данных на устройстве пользователя, предотвращать потерю информации, обеспечивать хеширование паролей для защиты от несанкционированного доступа, а также сохранять историю всех операций для последующего анализа. Кроме того, предполагается возможность масштабирования разработанного решения — при необходимости функционал может быть расширен за счёт добавления новых трекеров (физическая активность, артериальное давление, настроение),

					ПГУ1.090301.43.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

синхронизации с облачным хранилищем или интеграции с фитнес-браслетами.

Практическая значимость работы заключается в том, что разработанное приложение позволит пользователю значительно сократить временные затраты на ведение учёта показателей здоровья, полностью исключить ошибки, связанные с человеческим фактором, предоставить возможность удобного и быстрого добавления записей в любое время, а также обеспечить наглядной и достоверной аналитической информацией для корректировки образа жизни. Внедрение подобной системы способствует повышению качества жизни, формированию устойчивых полезных привычек и отказу от вредных.

					ПГУ1.090301.43.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

1 Приложение по улучшению качества жизни

1.1 Анализ предметной области

Предметной областью данного исследования является процесс управления качеством жизни человека, включающий в себя организацию и контроль всех аспектов здоровья: от отслеживания потребления воды и калорийности питания до контроля продолжительности сна, динамики веса и прогресса по привычкам. Качество жизни представляет собой сложную комплексную характеристику, в которой взаимодействуют три основные группы показателей: физиологические параметры (вода, калории, сон, вес), поведенческие привычки (курение, алкоголь, сладкое, соцсети) и мотивационные факторы (достижения, серии, рекорды). Каждая из этих групп имеет свои особенности, цели и задачи, которые должны быть учтены при разработке мобильного приложения для улучшения качества жизни. Современный контроль здоровья — это не просто редкие замеры показателей, а систематический комплексный подход, предоставляющий широкий спектр возможностей: отслеживание водного баланса, учёт калорийности питания с детализацией по белкам, жирам и углеводам, мониторинг продолжительности сна, ведение журнала веса, формирование и закрепление полезных привычек, а также отказ от вредных. Успешное функционирование такого контроля требует чёткой координации множества процессов: регулярное добавление записей, отслеживание прогресса, визуализация динамики, получение мотивационной поддержки и многое другое.

В процессе анализа предметной области были выделены следующие ключевые сущности:

Пользователь — это физическое лицо, желающее улучшить качество своей жизни и ведущее учёт показателей здоровья. Пользователь обладает такими характеристиками, как уникальный идентификатор (id), имя пользователя (username), адрес электронной почты (email) и хешированный пароль (password_hash). Пользователь может быть как авторизованным в

					ПГУ1.090301.43.001 ПЗ	Лист
						9
Изм.	Лист	№ докум.	Подпись	Дата		

системе (иметь учётную запись), так и гостем (впервые открывшим приложение). Авторизация обеспечивает сохранность персональных данных и синхронизацию прогресса между сессиями.

Настройки пользователя — это индивидуальные параметры, которые пользователь может изменять в соответствии со своими целями. К ним относятся: дневная норма калорий (`daily_calories`), дневная норма воды (`daily_water_ml`), целевая продолжительность сна (`target_sleep_hours`), выбранная тема оформления (`theme_mode`: светлая, тёмная, системная), язык интерфейса (`language`: русский, английский), статус уведомлений (`enable_notifications`). Возможность настройки целей позволяет адаптировать приложение под конкретные потребности пользователя.

Запись о воде (`water_intake`) фиксирует объём выпитой воды в определённый момент времени. Характеризуется временной меткой (`timestamp`) и объёмом в миллилитрах (`volume_ml`). Позволяет отслеживать прогресс выполнения дневной нормы и анализировать привычку потребления жидкости.

Запись о питании (`calorie_intake`) — это документ, фиксирующий приём пищи пользователем. Содержит информацию о временной метке (`timestamp`), количестве калорий (`calories`), количестве белков (`protein`), жиров (`fat`) и углеводов (`carbs`), текстовом описании (`description`) и типе приёма (`meal_type`: завтрак, обед, ужин, перекус). Позволяет контролировать суточную калорийность рациона и баланс макронутриентов.

Запись о сне (`sleep_record`) фиксирует информацию о сне пользователя. Содержит дату (`date`), время засыпания (`sleep_start`), время пробуждения (`wake_up`) и рассчитанную продолжительность в минутах (`duration_minutes`). Приложение поддерживает активную сессию сна: при нажатии кнопки «Начать сон» фиксируется время засыпания, при нажатии «Проснулся» рассчитывается продолжительность и сохраняется запись.

Запись о весе (weight_record) фиксирует массу тела пользователя на определённую дату. Содержит дату измерения (date) и значение веса в килограммах (weight). Позволяет отслеживать динамику изменения веса с течением времени и визуализировать её на линейном графике.

Привычка (habit) — это поведенческая модель, которую пользователь хочет сформировать (полезная привычка) или от которой хочет отказаться (вредная привычка). Характеризуется названием (name), категорией (category: курение, алкоголь, сладкое, соцсети, другое), датой начала отслеживания (start_date) и целевой длительностью в днях (target_days). Примеры привычек: «не курю», «не пью алкоголь», «отказ от сладкого», «ограничение соцсетей», «утренняя зарядка».

Прогресс привычки (habit_progress) — это документ, фиксирующий текущее состояние выполнения привычки. Содержит текущую серию (current_streak) — количество дней от последнего сброса, рекордную серию (record_streak) — максимальное достигнутое значение, дату последнего сброса (last_reset_date) и общее количество отслеживаемых дней (total_tracking_days). Система автоматически рассчитывает серии на основе даты последнего сброса и текущей даты.

Достижение (achievement) — это награда, автоматически выдаваемая пользователю при выполнении определённых условий. Характеризуется идентификатором достижения (achievement_id), который соответствует одному из 100 predetermined достижений в списке AchievementsDefinitions, и датой получения (date_earned). Примеры достижений: «Первый сон» (добавлена первая запись сна), «Ранняя пташка» (пробуждение до 6:00), «Дневная норма» (выполнение нормы воды), «Золотая середина» (сон ровно 8 часов), «Неделя чистоты» (7 дней без срыва привычки). Рассмотрим основные процессы, протекающие в предметной области.

Процесс учёта воды в традиционном неавтоматизированном подходе выглядит следующим образом: пользователь запоминает или записывает в

блокнот каждый выпитый стакан воды, затем в конце дня пытается подсчитать суммарный объём. Такой подход характеризуется низкой точностью, забывчивостью и отсутствием наглядной обратной связи. Кроме того, пользователь не имеет возможности увидеть прогресс выполнения нормы в реальном времени, а анализ динамики за неделю или месяц становится крайне трудоёмким. В автоматизированном приложении StatLife пользователь добавляет запись о воде одним нажатием кнопки «+250 мл» или через ручной ввод объёма. Система автоматически суммирует записи за день, отображает круговой индикатор прогресса и цветовую индикацию выполнения нормы (синий — норма не выполнена, зелёный — норма выполнена). Календарь воды отображает дни с цветовой маркировкой, позволяя анализировать регулярность потребления жидкости за месяц.

Процесс учёта питания в традиционном подходе требует от пользователя ручного подсчёта калорий с использованием таблиц или сторонних сайтов, что крайне неудобно и занимает много времени. В автоматизированной системе пользователь через простую форму добавляет приём пищи, указывая калории, белки, жиры, углеводы, описание и тип приёма (завтрак, обед, ужин, перекус). Система автоматически суммирует калории за день, отображает прогресс относительно нормы с цветовой индикацией (зелёный — норма, жёлтый — небольшое отклонение, красный — значительное отклонение), а также формирует круговую диаграмму распределения приёмов пищи по типам.

Процесс трекинга сна в традиционном подходе требует от пользователя запоминания времени засыпания и пробуждения, а затем ручного расчёта продолжительности. В автоматизированной системе пользователь нажимает кнопку «Начать сон» перед сном — фиксируется время засыпания (с учётом 20 минут на засыпание). При пробуждении пользователь нажимает «Проснулся» — система автоматически рассчитывает продолжительность сна в минутах, сохраняет запись и отображает всплывающее уведомление с

результатом в часах. Календарь сна отображает дни с цветовой индикацией (зелёный — продолжительность соответствует целевой, красный — отклонение).

Процесс управления привычками в автоматизированной системе позволяет пользователю создавать привычки, указывая название, категорию, дату отказа и целевую длительность. Система автоматически рассчитывает текущую серию (количество дней от последнего сброса) и рекордную серию. При срыве пользователь нажимает кнопку «Сбросить» — текущая серия обнуляется, рекорд остаётся. Если задана целевая длительность, отображается прогресс-бар, показывающий отношение текущей серии к цели. При достижении цели прогресс-бар заполняется полностью, и пользователь получает соответствующее достижение.

Процесс получения достижений полностью автоматизирован. Система после каждого изменения данных (добавление записи, сброс серии, выполнение нормы и т.д.) проверяет условия для всех 100 достижений и автоматически выдаёт те, условия для которых выполнены. Пользователь получает мгновенное уведомление с названием достижения. На экране достижений все награды отображаются в виде сетки 3x3: полученные — цветные, недоступные — серые с замком.

На основе анализа предметной области были сформулированы требования к разрабатываемому мобильному приложению. Система должна обеспечивать регистрацию новых пользователей и аутентификацию существующих с хешированием паролей (SHA-256). Неавторизованные пользователи должны иметь возможность только зарегистрироваться или войти в систему. Авторизованные пользователи должны иметь возможность добавлять, просматривать и удалять записи о воде, питании, сне, весе; создавать привычки, отслеживать серии и сбрасывать их при срыве; получать достижения; просматривать графики и календари; экспортировать данные в CSV и PDF; создавать резервные копии базы данных и восстанавливать их;

настраивать индивидуальные цели, тему оформления, язык и уведомления. Система должна предотвращать некорректный ввод (калории от 0 до 10000, объём воды от 1 до 5000 мл, вес от 20 до 300 кг, имя пользователя от 3 до 20 символов, пароль от 6 символов). К нефункциональным требованиям относятся работа на платформах Android и iOS, время отклика не более одной секунды, обеспечение целостности и сохранности данных (локальное хранение SQLite), интуитивно понятный интерфейс в стиле Material Design, полная автономность (без подключения к интернету) и безопасность данных с хешированием паролей.

Проведённый анализ показал, что разрабатываемое приложение StatLife призвано решить выявленные проблемы за счёт автоматизации ключевых процессов, позволяя пользователю сократить временные затраты, повысить точность учёта показателей здоровья, получить наглядную визуализацию прогресса и мотивационную поддержку через систему достижений и серий.

1.2 Обзор аналогов

Обзор аналогов является важной частью разработки мобильного приложения, так как позволяет изучить подобные решения и определить их преимущества и недостатки. В рамках данного исследования был проведен анализ рынка мобильных приложений для трекинга здоровья и формирования привычек с целью выявления существующих аналогов и определения их сильных и слабых сторон. Следует отметить, что специализированных программных решений для комплексного контроля здоровья на российском рынке представлено не так много, а многие существующие разработки ориентированы преимущественно на один аспект здоровья (только питание, только воду или только сон) и не предоставляют комплексного подхода. Тем не менее, можно выделить несколько категорий аналогов, каждая из которых имеет свои особенности.

Первым аналогом является приложение «MyFitnessPal». Данный сервис представлен на рисунке 1. MyFitnessPal представляет собой многофункциональную платформу, предназначенную для учёта калорийности питания и физической активности. Система объединяет в себе базу данных продуктов (более 14 млн позиций), возможность сканирования штрих-кодов, учёт белков, жиров и углеводов, а также синхронизацию с фитнес-трекерами и умными весами. MyFitnessPal включает в себя дневник питания, дневник тренировок, формирование отчётов по дням, неделям и месяцам, а также возможность установки целей по снижению или поддержанию веса. Приложение ориентировано на пользователей, желающих контролировать свой рацион и физическую активность. Основными преимуществами являются широкая функциональность, огромная база продуктов. К недостаткам можно отнести отсутствие встроенных трекеров воды и сна, отсутствие системы достижений и привычек, а также наличие платной подписки для доступа к расширенной статистике и планам питания.

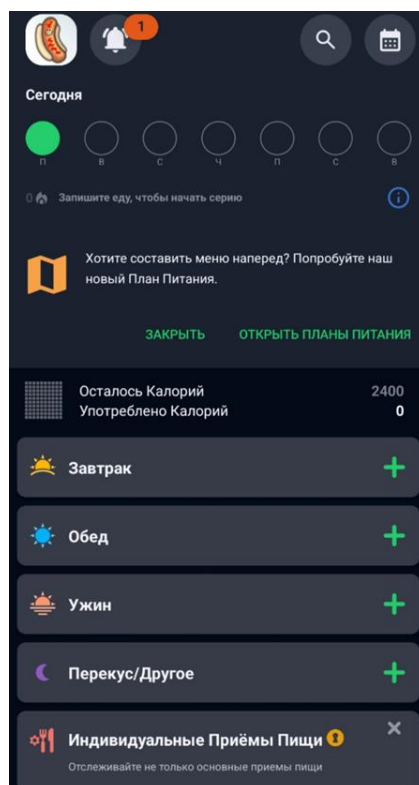


Рисунок 1 — «MyFitnessPal»

Вторым аналогом является приложение «WaterMinder». Данный сервис представлен на рисунке 2. WaterMinder представляет собой специализированный трекер потребления воды, предназначенный для контроля водного баланса. Приложение позволяет быстро добавлять выпитую воду стандартными объёмами или через ручной ввод, рассчитывает индивидуальную норму воды на основе веса, возраста и активности пользователя, отправляет напоминания о необходимости пить воду, а также предоставляет виджеты для экрана блокировки и домашнего экрана. Приложение ориентировано на пользователей, желающих следить за своим водным балансом. Основным преимуществом является простой и понятный интерфейс, а также качественная визуализация прогресса в виде анимированной капли. К недостаткам можно отнести узкую специализацию (только вода), отсутствие учёта питания, сна и привычек, а также отсутствие игрофикации (достижений, серий). Кроме того, бесплатная версия ограничена по времени, а полная версия требует единоразовой покупки.

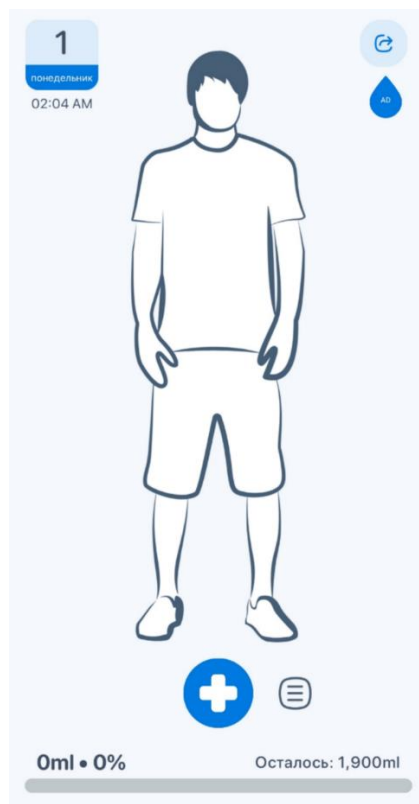


Рисунок 2 — «WaterMinder»

Третьим аналогом является приложение «Sleep Cycle». Данный сервис представлен на рисунке 3. Sleep Cycle представляет собой приложение для анализа сна с «умным» будильником, предназначенное для контроля качества сна. Приложение отслеживает фазы сна через акселерометр или микрофон смартфона, пробуждает пользователя в оптимальную фазу (лёгкий сон) в заданном временном окне, записывает звуки во сне (храп, разговоры), а также предоставляет долгосрочную статистику качества сна. Приложение ориентировано на пользователей, желающих улучшить качество своего сна и просыпаться более отдохнувшими. Основными преимуществами являются высокая точность определения фаз сна, приятный минималистичный дизайн и интеграция с Apple Health и Google Fit. К недостаткам можно отнести отсутствие учёта воды, питания и привычек, необходимость платной подписки для получения долгосрочных трендов и экспорта данных, а также отсутствие поддержки ручного ввода сна (только автоматический анализ), что ограничивает применение для людей, спящих не в одиночку или с домашними животными.

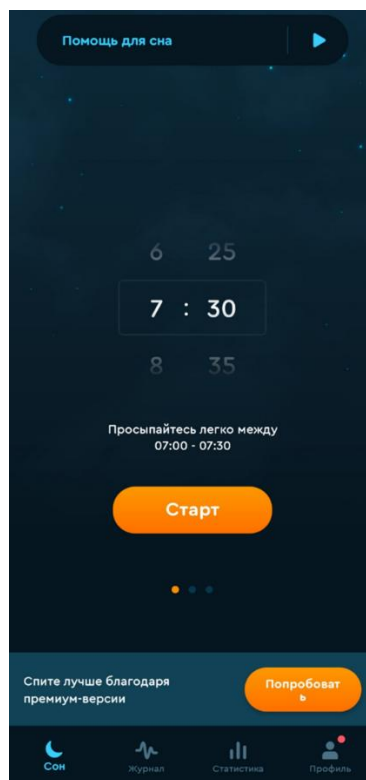


Рисунок 3 — «Sleep Cycle»

Сравнение преимуществ и недостатков рассмотренных аналогов представлено в таблице 1.

Таблица 1 — Сравнительная таблица аналогов

Аналог	Преимущества	Недостатки
MyFitnessPal	Огромная база продуктов (, учёт белков, жиров и углеводов (БЖУ), синхронизация с фитнес-трекерами и умными весами, формирование отчётов по дням, неделям и месяцам	Отсутствие встроенного трекера воды и сна, отсутствие системы достижений и привычек, платная подписка
WaterMinder	Простой и понятный интерфейс, напоминания о воде, виджеты на экран блокировки, визуализация прогресса в виде анимированной капли, расчёт индивидуальной нормы воды на основе веса, возраста и активности	Узкая специализация , отсутствие учёта питания, сна и привычек, отсутствие геймификации (достижений, серий), бесплатная версия ограничена по времени, полная версия требует единовременной покупки
Sleep Cycle	Автоматический анализ фаз сна через акселерометр или микрофон, «умный» будильник с пробуждением в оптимальную фазу лёгкого сна, долгосрочная статистика качества сна, интеграция с Apple Health и Google Fit	Отсутствие учёта воды, питания и привычек, платная подписка для получения долгосрочных трендов и экспорта данных, отсутствие поддержки ручного ввода сна (автоматически), что ограничивает применение для людей, спящих не в одиночку

На основании проведённого анализа был сделан вывод, что на рынке мобильных приложений для трекинга здоровья имеется потенциал для разработки новых решений, объединяющих в себе комплексный подход: учёт воды, питания, сна, веса и привычек в одном приложении, с системой достижений, визуализацией данных и экспортом информации.

1.3 Анализ требований к разработке

В рамках проекта разрабатывается кроссплатформенное мобильное приложение «StatLife» для улучшения качества жизни, предназначенное для сбора, обработки и анализа данных о физиологических показателях пользователя и прогрессе по привычкам. Программа ориентирована на широкий круг пользователей, которые нуждаются в контроле своего здоровья и формировании полезных привычек, что критически важно для эффективного улучшения качества жизни и повышения мотивации. Разрабатываемая система призвана заменить устаревшие ручные методы ведения учёта, основанные на бумажных дневниках, заметках в телефоне или электронных таблицах, которые характеризуются низкой точностью, высокими временными затратами и неизбежными ошибками при внесении данных. Помимо функциональных требований, в связи с тем, что разрабатываемое приложение имеет локальную архитектуру (все данные хранятся на устройстве пользователя), целесообразно разделить описание требований на клиентскую часть (пользовательский интерфейс) и слой бизнес-логики с базой данных. Клиентская часть отвечает за отображение информации, взаимодействие с пользователем и вызов методов бизнес-логики. Слой бизнес-логики обрабатывает запросы от клиентской части, взаимодействует с базой данных, реализует расчёты (продолжительность сна, серии привычек, проверка условий достижений) и возвращает обработанные данные в интерфейс.

					ПГУ1.090301.43.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		19

Для клиентской части разрабатываемой системы выдвигаются следующие требования:

интуитивно понятный интерфейс, не требующий специального обучения для конечных пользователей, с единым стилем оформления всех экранов и адаптивной вёрсткой для корректного отображения на различных устройствах (Android и iOS);

- главный экран (сводка) с карточками трекеров, отображающими краткую информацию по каждому показателю: количество выпитой воды за день, суммарные калории, продолжительность последнего сна, количество отслеживаемых привычек, количество полученных достижений и быстрый переход к журналу веса;

- экран трекера воды с круговым индикатором прогресса выполнения дневной нормы, кнопками для быстрого добавления воды (+250 мл) и ручного ввода объёма, списком записей за текущий день с возможностью удаления свайпом, а также календарём с цветовой индикацией дней, когда норма воды была выполнена;

- экран трекера питания с отображением суммарных калорий за день и прогресса относительно нормы, тремя круговыми индикаторами для белков, жиров и углеводов, формой добавления приёма пищи (калории, БЖУ, описание, тип приёма), списком записей за день с возможностью удаления свайпом, а также календарём с цветовой индикацией выполнения нормы калорий;

- экран трекера сна с кнопками «Начать сон» и «Проснулся», автоматическим расчётом продолжительности сна, календарём с цветовой индикацией дней, где продолжительность сна соответствовала целевой, а также возможностью установки будильника;

- экран трекера веса с формой добавления новой записи (вес, дата), линейным графиком изменения веса и списком всех записей;

– экран трекера привычек с возможностью создания новой привычки (название, категория, дата отказа, целевая длительность), отображением списка привычек с указанием текущей и рекордной серий, прогресс-бара цели (при наличии) и кнопок сброса серии и удаления привычки;

– экран достижений с отображением всех 100 достижений в виде сетки 3×3, gdzie полученные достижения имеют цветную иконку и название, а недоступные — серую иконку с замком, а также счётчиком «Собрано: X / 100»;

– экран графиков и календарей с линейными графиками потребления воды и калорий за последние 7 дней и круговой диаграммой распределения приёмов пищи по типам;

– экран настроек с возможностью изменения нормы воды, нормы калорий, целевой продолжительности сна, переключения темы оформления (светлая, тёмная, системная), выбора языка интерфейса (русский, английский), включения/отключения уведомлений, а также кнопками для экспорта данных в CSV и PDF, резервного копирования и восстановления базы данных;

– экран авторизации и регистрации с полями для ввода имени пользователя, email и пароля, а также проверкой корректности вводимых данных (имя пользователя od 3 do 20 символов, пароль не менее 6 символов).

Для слоя бизнес-логики и базы данных разрабатываемой системы выдвигаются следующие требования:

– обработка запросов от клиентской части, реализация бизнес-логики приложения (расчёт продолжительности сна, серий привычек, проверка условий достижений) и взаимодействие с базой danych przez DatabaseHelper;

– регистрация и аутентификация пользователей с хешированием паролей по алгоритму SHA-256 и автоматическим восстановлением сессии przez SharedPreferences;

– добавление, удаление и получение записей о воде (WaterIntake): идентификатор, идентификатор пользователя, временная метка, объём w миллилитрах;

– добавление, удаление и получение записей о питании (CalorieIntake): идентификатор, идентификатор пользователя, временная метка, калории, белки, жиры, углеводы, описание, тип приёма (завтрак, обед, ужин, перекус);

– добавление, удаление и получение записей о сне (SleepRecord): идентификатор, идентификатор пользователя, дата, время засыпания, время пробуждения, продолжительность w минутах;

– добавление, удаление и получение записей о весе (WeightRecord): идентификатор, идентификатор пользователя, дата, вес;

– создание, удаление и получение привычек (Habit): идентификатор, идентификатор пользователя, название, категория, дата начала, целевая длительность;

– автоматический расчёт текущих и рекордных серий по привычкам na podstawie даты последнего сброса и текущей даты;

– сброс серии привычки с обнулением текущей серии и сохранением рекорда;

– система достижений (100 уникальных наград) с автоматической проверкой условий przy każdym изменении данных и выдачей достижений с уведомлением;

– проверка корректности вводимых данных: калории od 0 do 10000, объём воды od 1 do 5000 ml, вес od 20 do 300 kg, имя пользователя od 3 do 20 символов (только буквы и цифры), пароль не менее 6 символов;

– валидация всех входящих данных для предотвращения некорректных запросов.

Для базы данных разрабатываемой системы выдвигаются следующие требования:

					ПГУ1.090301.43.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		22

- хранение информации о пользователях (users): идентификатор, имя пользователя, email, хеш пароля;
- хранение настроек пользователя (user_settings): идентификатор пользователя, дневная норма калорий, дневная норма воды, целевая продолжительность сна, тема оформления, язык, статус уведомлений;
- хранение записей о воде (water_intake): идентификатор, идентификатор пользователя, временная метка, объём w миллилитрах;
- хранение записей о питании (calorie_intake): идентификатор, идентификатор пользователя, временная метка, калории, белки, жиры, углеводы, описание, тип приёма;
- хранение записей о сне (sleep_records): идентификатор, идентификатор пользователя, дата, время засыпания, время пробуждения, продолжительность w минутах;
- хранение записей о весе (weight_records): идентификатор, идентификатор пользователя, дата, вес;
- хранение привычек (habits): идентификатор, идентификатор пользователя, название, категория, дата начала, целевая длительность;
- хранение прогресса привычек (habit_progress): идентификатор привычки, текущая серия, рекордная серия, дата последнего сброса, общее количество отслеживаемых дней;
- хранение полученных достижений (achievements): идентификатор, идентификатор пользователя, идентификатор достижения, дата получения;
- обеспечение целостности ссылок с помощью внешних ключей i kaskadowego удаления (ON DELETE CASCADE).

Исходя из целевого назначения, были сформированы диаграммы прецедентов для пользователя, изображённые на рисунках 4–6.

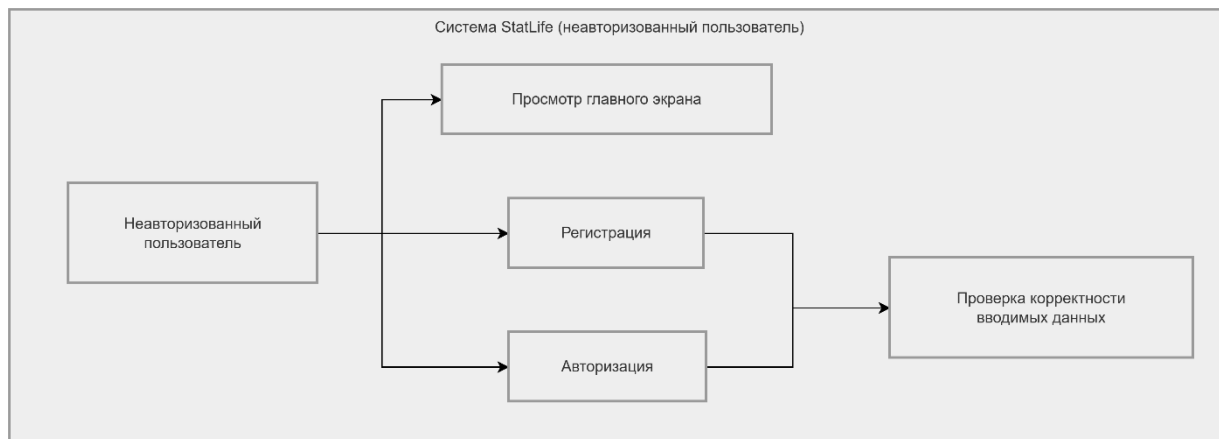


Рисунок 4 — Диаграмма прецедентов неавторизованного пользователя

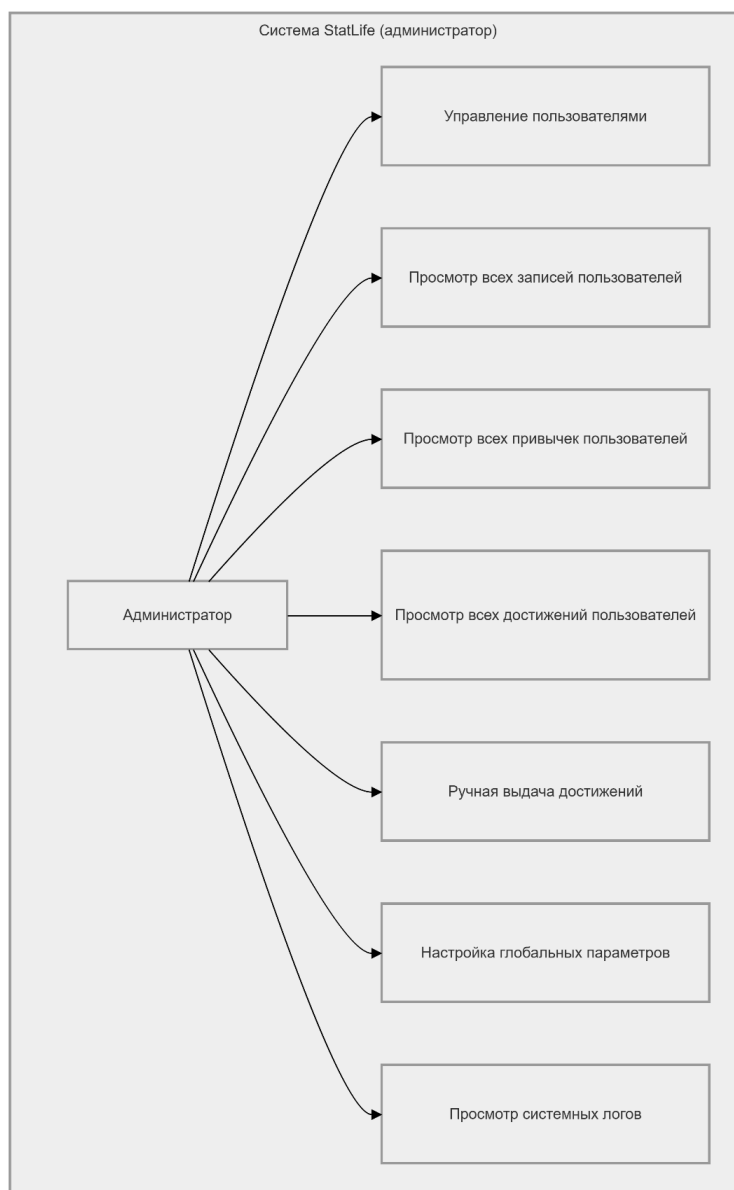


Рисунок 5 — Диаграмма прецедентов администратора

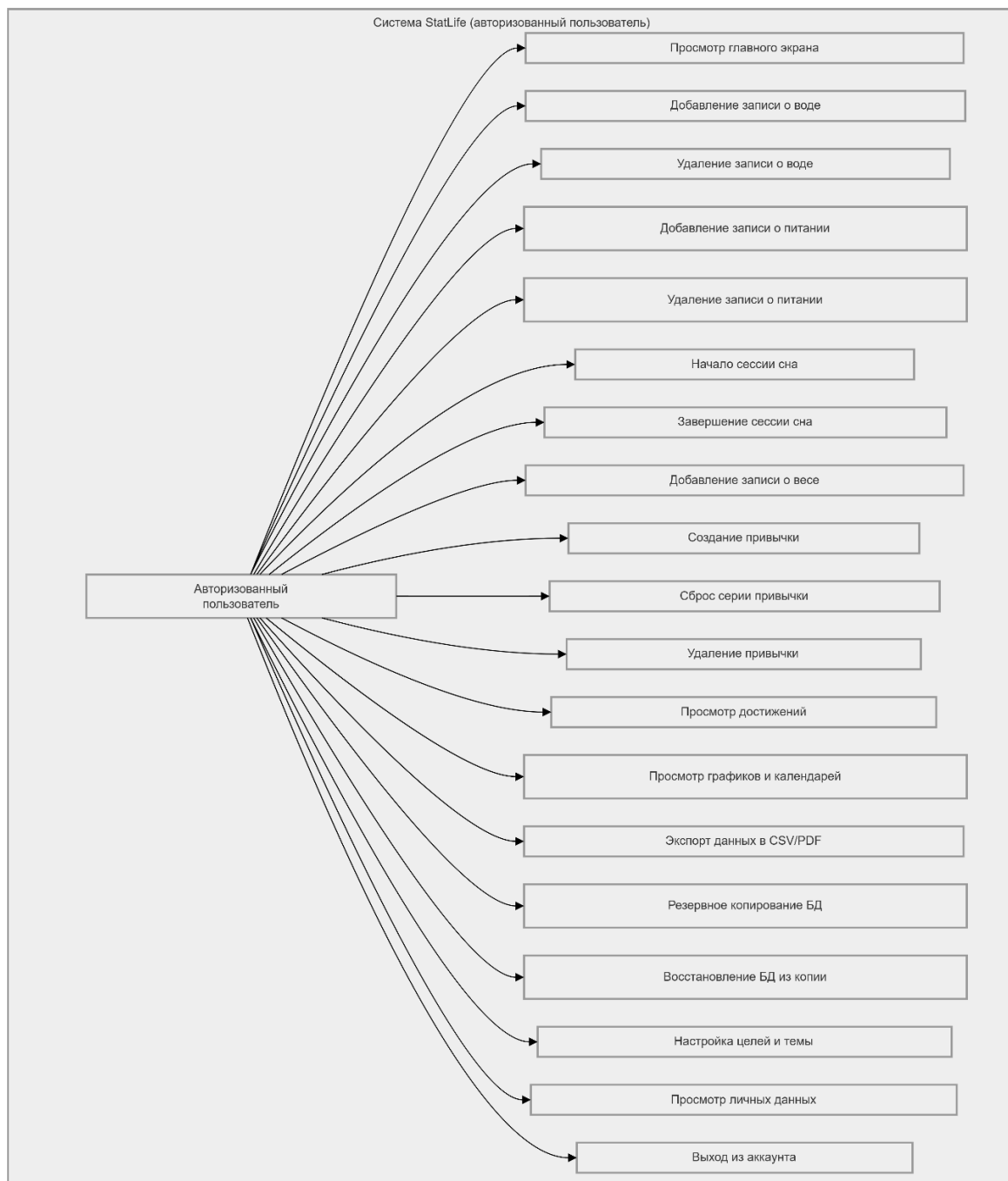


Рисунок 6 — Диаграмма прецедентов авторизованного пользователя

1.4 Выбор архитектуры системы

Для организации корректного и отказоустойчивого взаимодействия между пользователем и приложением, работа приложения StatLife построена на локальной архитектуре с асинхронными операциями. В отличие от клиент-серверной архитектуры, все данные хранятся и обрабатываются

непосредственно на устройстве пользователя, что обеспечивает полную автономность и конфиденциальность данных.

Основной рабочий цикл программы можно разбить на следующие этапы:

- ожидание и обработка пользовательского взаимодействия с элементами графического интерфейса мобильного приложения. Пользователь нажимает на кнопки, вводит данные в текстовые поля или выбирает элементы списков, что генерирует событие, которое перехватывается соответствующим обработчиком на стороне клиента;
- в обработчике события инициируется асинхронный вызов метода `AppProvider`. Это позволяет не блокировать поток пользовательского интерфейса на время выполнения длительных операций, таких как добавление записи в базу данных, расчёт продолжительности сна или проверка условий достижений;
- слой бизнес-логики (`AppProvider`) принимает запрос, выполняет валидацию входящих данных (проверка диапазонов калорий, объёма воды, веса), при необходимости обращается к `DatabaseHelper` для выполнения операций с базой данных `SQLite`. `DatabaseHelper` выполняет чтение или запись информации о пользователях, воде, питании, сне, весе, привычках и достижениях;
- извлечённые или сохранённые данные возвращаются в `AppProvider`, который обновляет своё состояние (изменяет значения переменных, списков записей) и вызывает метод `notifyListeners()`, уведомляя все виджеты, использующие `Consumer`, об изменении состояния;
- после получения уведомления от `AppProvider` соответствующие виджеты перерисовываются, отображая обновлённую информацию: круговой индикатор прогресса воды, суммарные калории за день, список записей, календарь сна или график веса. При необходимости отображаются всплывающие уведомления через `SnackBar`, информирующие о результате

выполненного действия — успешном добавлении записи, ошибке валидации или получении нового достижения;

– приложение вновь переходит в режим ожидания следующего пользовательского события, готовое к выполнению нового цикла. Время выполнения цикла зависит от сложности операции, количества записей в базе данных и производительности устройства.

Функциональная схема выполнения программы показана на рисунке 7.

					ПГУ1.090301.43.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		27

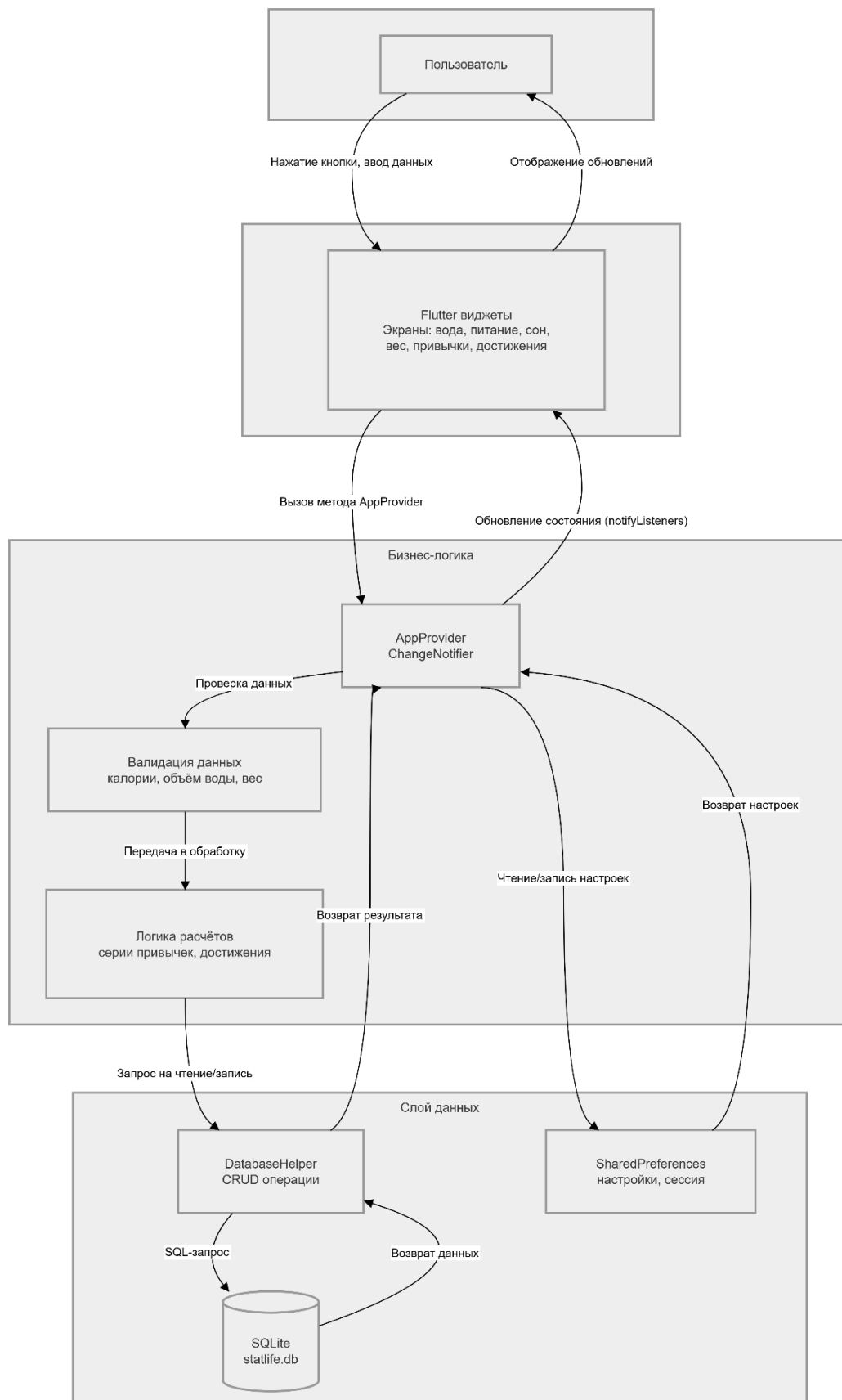


Рисунок 7 — Функциональная схема проекта

Детально процесс авторизации и последующего сбора данных показан на диаграммах последовательностей (рисунки 8 и 9).

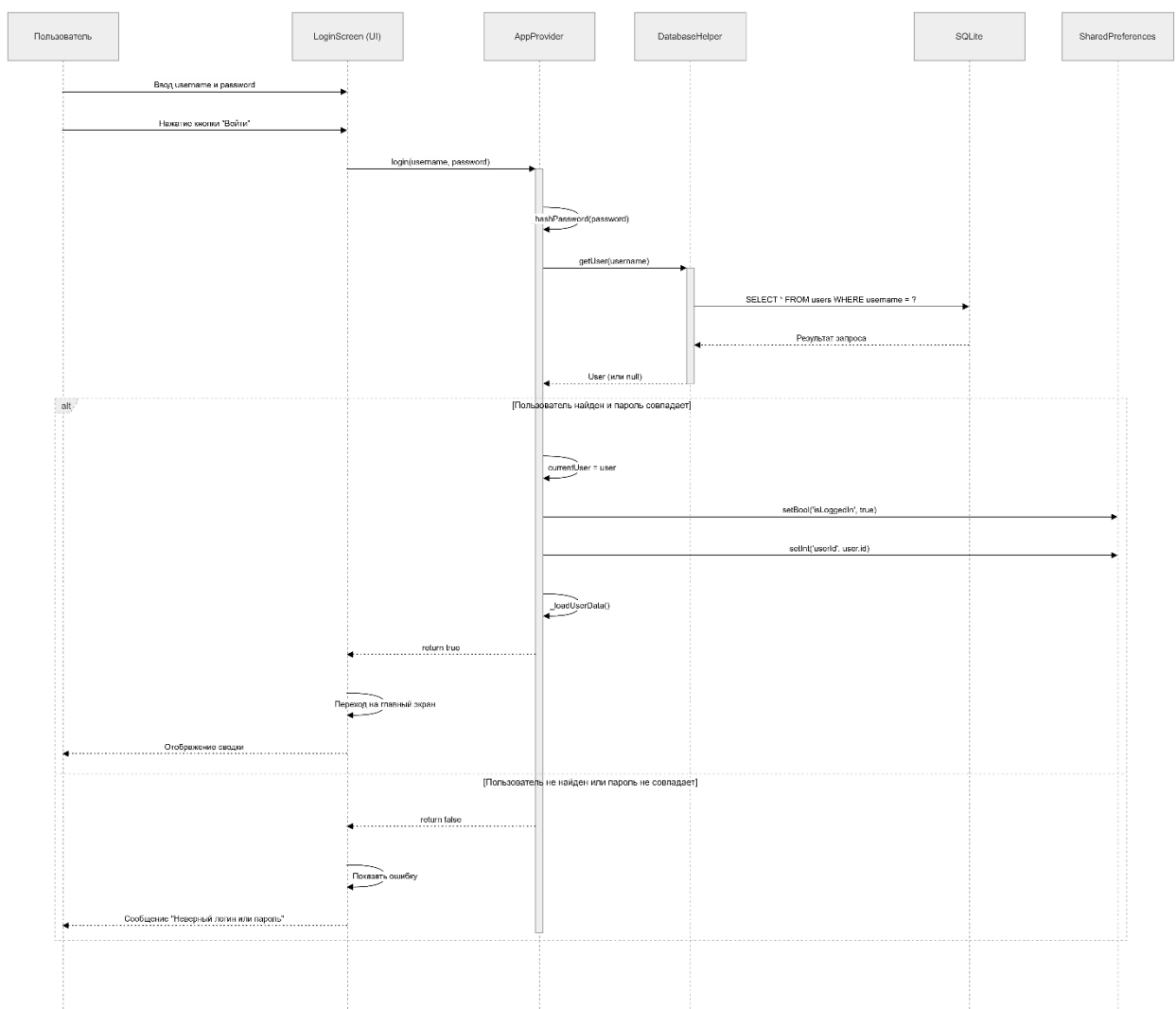


Рисунок 8 — Диаграмма последовательностей «Авторизация»

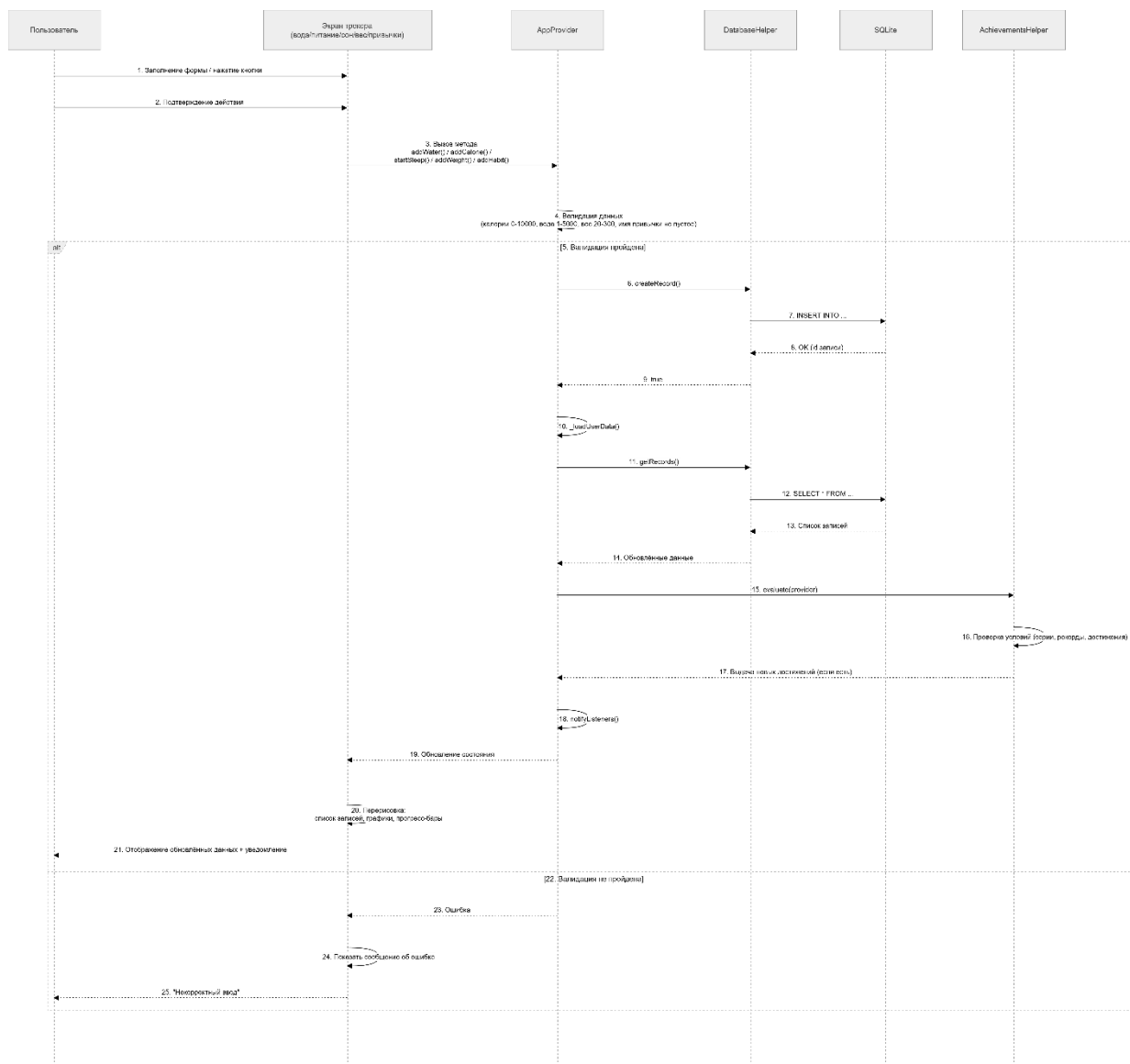


Рисунок 9 — Обобщённая диаграмма последовательности добавления записи в трекарах StatLife

Развертывание приложения StatLife не требует серверного оборудования или облачного хостинга, так как приложение является полностью локальным и работает непосредственно на мобильном устройстве пользователя. Все компоненты системы (пользовательский интерфейс, бизнес-логика и база данных) функционируют в изолированной среде операционной системы мобильного устройства (Android или iOS).

Для установки приложения пользователю необходимо скачать установочный файл (.apk для Android или .ipa для iOS) и выполнить установку на устройство. Приложение не требует подключения к интернету для выполнения основных функций, за исключением экспорта данных (CSV, PDF) и резервного копирования, которые используют файловую систему устройства.

Клиентская часть представляет собой скомпилированное Flutter-приложение, которое выполняется на устройстве пользователя. База данных SQLite создаётся автоматически при первом запуске приложения и хранится во внутреннем хранилище устройства. Пользователь взаимодействует с приложением через сенсорный экран. Все данные сохраняются локально на устройстве, что обеспечивает полную конфиденциальность и возможность работы без интернета.

Архитектура развертывания системы отражена на рисунке 10.

					ПГУ1.090301.43.001 ПЗ	Лист
						31
Изм.	Лист	№ докум.	Подпись	Дата		

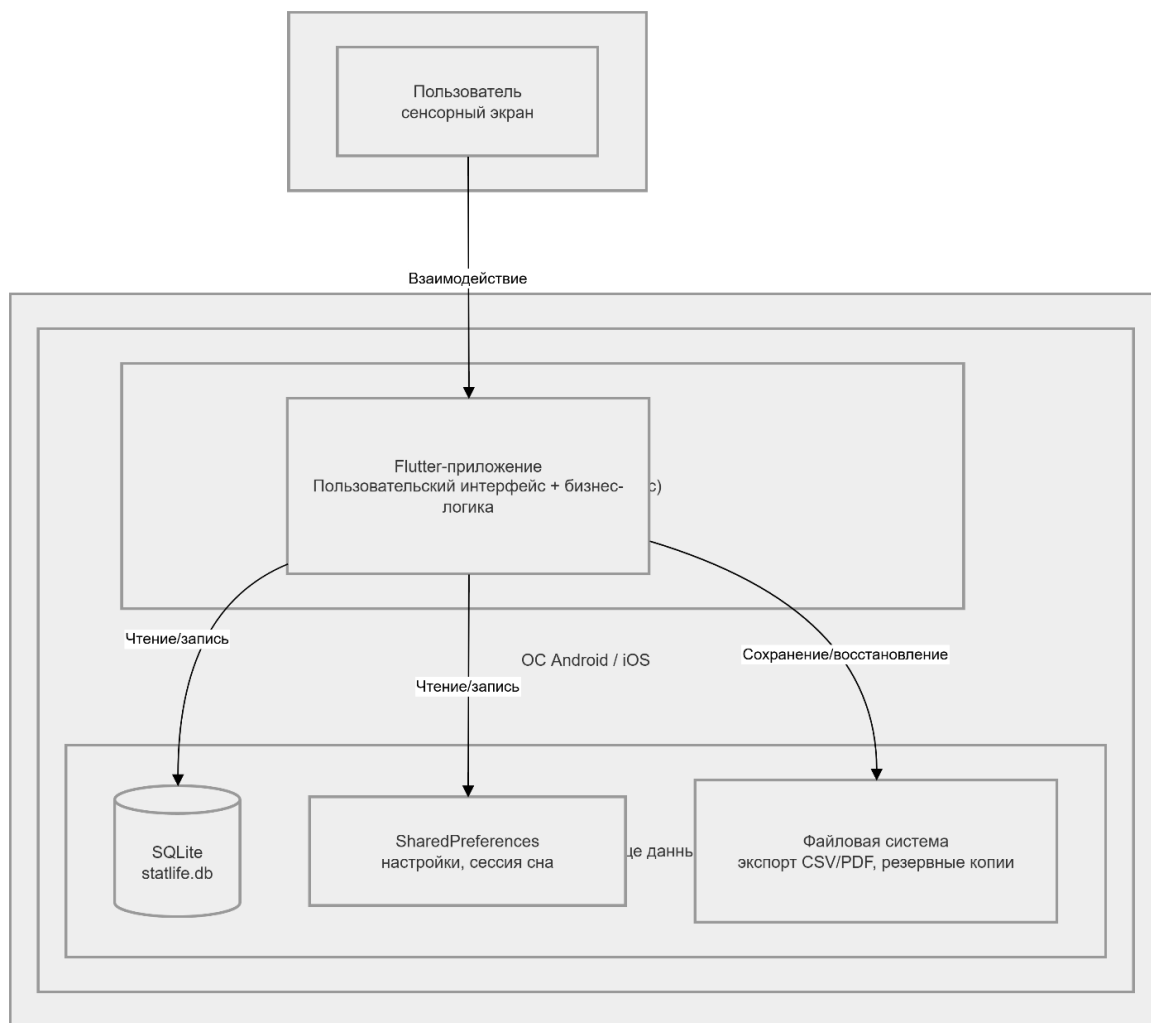


Рисунок 10 — Диаграмма развертывания приложения

Исходя из требуемого функционала, планируемой архитектуры проекта и выбранной целевой платформы, был определен следующий набор инструментов разработки: для клиентской части — язык программирования Dart, фреймворк Flutter, среда разработки Visual Studio Code; для слоя бизнес-логики — язык Dart, библиотека Provider для управления состоянием; для хранения данных — СУБД SQLite, пакет sqflite для работы с базой данных, пакет shared_preferences для хранения настроек и состояния сессии; для визуализации данных — библиотеки fl_chart (графики и диаграммы) и table_calendar (календарь с цветовой индикацией); для локальных уведомлений — пакет flutter_local_notifications; для экспорта данных — пакеты pdf и csv; для резервного копирования и работы с файловой системой

— пакеты `file_picker` и `path_provider`; для системы контроля версий — `Git`; для управления зависимостями — `pub` (стандартный менеджер пакетов экосистемы `Dart/Flutter`). Операционная система разработки не имеет принципиального значения, так как все используемые инструменты являются кроссплатформенными и могут работать как на `Windows`, так и на `macOS` или `Linux`.

1.5 Выбор языка программирования

`Dart` — статически типизированный язык программирования, разработанный компанией `Google` как основной язык для фреймворка `Flutter`. Он предназначен для разработки кроссплатформенных мобильных приложений для `Android` и `iOS`, включая клиентскую часть и бизнес-логику, благодаря возможности выявлять ошибки на этапе компиляции. `Dart` добавляет в язык такие возможности, как статическая типизация с выводом типов (`type inference`), `null safety` (встроенная защита от `null`-ссылок), классы, интерфейсы, дженерики, перечисления, асинхронное программирование (`async/await`, `Future`, `Stream`) и модули. Программа на `Dart` представляет собой набор классов, функций и переменных, которые компилируются в нативный код для целевой платформы (`AOT`-компиляция для `iOS` и `Android`) или выполняются в виртуальной машине `Dart` с поддержкой горячей перезагрузки (`JIT`-компиляция для ускорения разработки). Классы могут наследоваться от базовых классов и реализовывать интерфейсы, что позволяет создавать гибкую и расширяемую архитектуру, удобную для поддержки и масштабирования проекта.

При программировании используются встроенные типы и наборы библиотечных модулей из экосистемы `Flutter` и `Dart`, а также собственные классы и виджеты, написанные на `Dart`. Класс — это шаблон для создания объектов, объединяющий данные и поведение. Встроенные библиотеки предоставляют широкий набор готовых компонентов: классы для работы с

HTTP-запросами, файловой системой, базами данных через sqflite, коллекциями и датами. В клиентской части Flutter-виджеты представляют собой классы, расширяющие StatelessWidget или StatefulWidget, и возвращают элемент интерфейса для отображения на экране мобильного устройства. В слое бизнес-логики класс AppProvider, наследующий ChangeNotifier, управляет состоянием приложения и обрабатывает запросы от пользовательского интерфейса.

Каждый отдельный метод представляет собой последовательность инструкций, выполняемых строго в порядке их написания. Методы могут вызывать другие методы, создавая иерархию вызовов. Внутри метода инструкции выполняются последовательно сверху вниз, за исключением случаев использования управляющих конструкций: условные операторы, циклы, обработка исключений. Результат выполнения метода возвращается в вызывающий код либо сохраняется в полях класса для дальнейшего использования. Асинхронные методы, помеченные ключевым словом async, позволяют выполнять длительные операции (например, запросы к базе данных или файловой системе) без блокировки основного потока выполнения.

Ключевой особенностью Dart является наличие строгой системы типов и null safety, позволяющих выявлять ошибки на этапе разработки до выполнения программы. В сочетании с реактивной моделью Flutter, где действия пользователя (нажатие кнопок, ввод текста) генерируют события, которые обрабатываются методами AppProvider с последующим уведомлением интерфейса через notifyListeners(), это позволяет создавать интерактивные мобильные приложения, реагирующие на действия пользователя в реальном времени без задержек и потери данных.

1.6 Выбор среды разработки

В качестве среды разработки для создания кроссплатформенного мобильного приложения StatLife была выбрана среда разработки Visual Studio

					ПГУ1.090301.43.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		34

Code. Данный выбор обусловлен тем, что VS Code идеально подходит для разработки на языке Dart, поддерживает работу с фреймворком Flutter, а также предоставляет богатую экосистему расширений, упрощающих процесс создания мобильных приложений для Android и iOS.

Visual Studio Code — это легковесная интегрированная среда разработки (IDE) от компании Microsoft, предназначенная для разработки современных приложений на различных языках программирования. VS Code работает на всех основных операционных системах: Windows, macOS и Linux, что обеспечивает кроссплатформенность разработки и не привязывает проект к конкретной ОС.

Возможности Visual Studio Code, используемые в проекте:

- встроенная поддержка Dart и Flutter обеспечивает подсветку синтаксиса, автодополнение кода IntelliSense, проверку типов в реальном времени и быструю навигацию по проекту. Расширения Flutter и Dart, устанавливаемые из официального маркетплейса, предоставляют возможность горячей перезагрузки (hot reload), позволяющей мгновенно видеть изменения кода на эмуляторе или реальном устройстве без перекомпиляции всего приложения;
- редактор кода с подсветкой синтаксиса обеспечивает удобное написание программного кода на Dart с автоматическим дополнением, подсказками по параметрам методов и функций, а также возможностью быстрого рефакторинга (переименование символов, выделение в отдельную функцию, импорт модулей);
- встроенный терминал позволяет выполнять команды Flutter (flutter run, flutter build apk, flutter pub get), управлять зависимостями, запускать приложение на эмуляторе или реальном устройстве без переключения на отдельное окно терминала;
- отладчик позволяет выполнять программу по шагам, устанавливать точки останова, отслеживать значения переменных в реальном

времени, анализировать состояние виджетов и потоки выполнения асинхронных операций, что необходимо для поиска и устранения ошибок;

- менеджер пакетов pub интегрирован через встроенный терминал и позволяет легко подключать сторонние библиотеки, такие как provider, sqflite, fl_chart, table_calendar, flutter_local_notifications, pdf, csv, file_picker, path_provider и другие, путём добавления зависимостей в файл pubspec.yaml с последующим выполнением команды flutter pub get;

- расширения предоставляют возможность добавлять дополнительный функционал, улучшающий процесс разработки. Ключевые расширения для проекта: Flutter — основное расширение для разработки, предоставляющее автодополнение, горячую перезагрузку и отладку; Dart — поддержка языка программирования Dart; Pubspec Assist — вспомогательное расширение для быстрого добавления зависимостей в pubspec.yaml;

- Flutter Tree — визуализация дерева виджетов; Error Lens — подсветка ошибок непосредственно в строке кода;

- встроенная система контроля версий Git позволяет отслеживать изменения кода, создавать коммиты, ветки и синхронизироваться с удалёнными репозиториями (например, GitHub) без выхода из среды разработки. Это удобно для ведения проекта и сохранения истории изменений;

- интеграция с эмулятором Android через Android Studio и подключение реальных устройств позволяют тестировать приложение на различных конфигурациях без дополнительных настроек. Выбор устройства для запуска осуществляется через панель состояния VS Code.

Пример фрагмента программы в среде Visual Studio Code представлен на рисунке 11.

```

main.dart
61 builder: (context) {
62   if (provider.isInitializing) {
63     return const Scaffold(body: Center(child: CircularProgressIndicator()));
64   }
65   if (provider.currentUser == null) {
66     return const LoginScreen();
67   }
68   return const MainNavigator();
69 },
70 ),
71 ),
72 ),
73 ),
74 ),
75 ),
76 ),
77 class MainNavigator extends StatefulWidget {
78   const MainNavigator({Key? key}) : super(key: key);
79
80   @override
81   State<MainNavigator> createState() => _MainNavigatorState();
82 }
83
84 class _MainNavigatorState extends State<MainNavigator> {
85   int _currentIndex = 0;
86
87   @override
88   Widget build(BuildContext context) {
89     final screens = [
90       HomeScreen(onNavigate: (index) => setState(() => _currentIndex = index)),
91       const SleepScreen(),
92       const NutritionScreen(),
93       const WaterScreen(),
94       const HabitsScreen(),
95       const AchievementsScreen(),
96     ];
97
98     return Scaffold(
99       body: screens[_currentIndex],
100       bottomNavigationBar: NavigationBar(
101         labelBehavior: NavigationDestinationLabelBehavior.alwaysHide,
102         selectedIndex: _currentIndex,
103         onDestinationSelected: (index) {
104           setState(() {
105             _currentIndex = index;
106           });
107         },
108         destinations: const [
109           NavigationDestination(
110             icon: Icon(Icons.dashboard_outlined),
111             selectedIcon: Icon(Icons.dashboard),
112             label: 'Главная',
113           ),
114           NavigationDestination(
115             icon: Icon(Icons.bedtime_outlined),
116             selectedIcon: Icon(Icons.bedtime),
117             label: 'Сон',

```

Рисунок 11 — пример фрагмента алгоритма

Visual Studio Code оказался довольно простым в освоении — интерфейс минималистичный, ничего лишнего, при этом всё нужное под рукой. Для новичка это плюс: не надо разбираться в десятках кнопок и панелей. Плюс обучающих материалов в интернете хватает, да и сам редактор подсказывает по командам, если что.

Подключение нужных расширений заняло пару минут — поставил Flutter, Dart, пару вспомогательных — и можно сразу начинать писать код, не тратя время на долгую настройку.

Отладка тоже удобная. Ставишь точку остановки, смотришь значения переменных, проверяешь, где ошибка. В консоли отладки видно, что происходит в процессе выполнения. Это реально помогает быстрее найти проблему, чем гадать по логам.

Редактор хорошо показывает связи между файлами. Кликаешь по импорту — сразу переходишь в нужный файл, ссылки подсвечиваются, понятно, откуда что вызывается. Это особенно выручает, когда проект разрастается и файлов становится много.

Если архитектура приложения грамотно продумана и логика отделена от интерфейса, то разрабатывать в VS Code заметно удобнее, чем в обычном блокноте, где ничего не подсвечивается и приходится всё искать руками. И ещё важный момент — VS Code полностью бесплатный. Сайт Visual Studio Code представлен на рисунке 12.

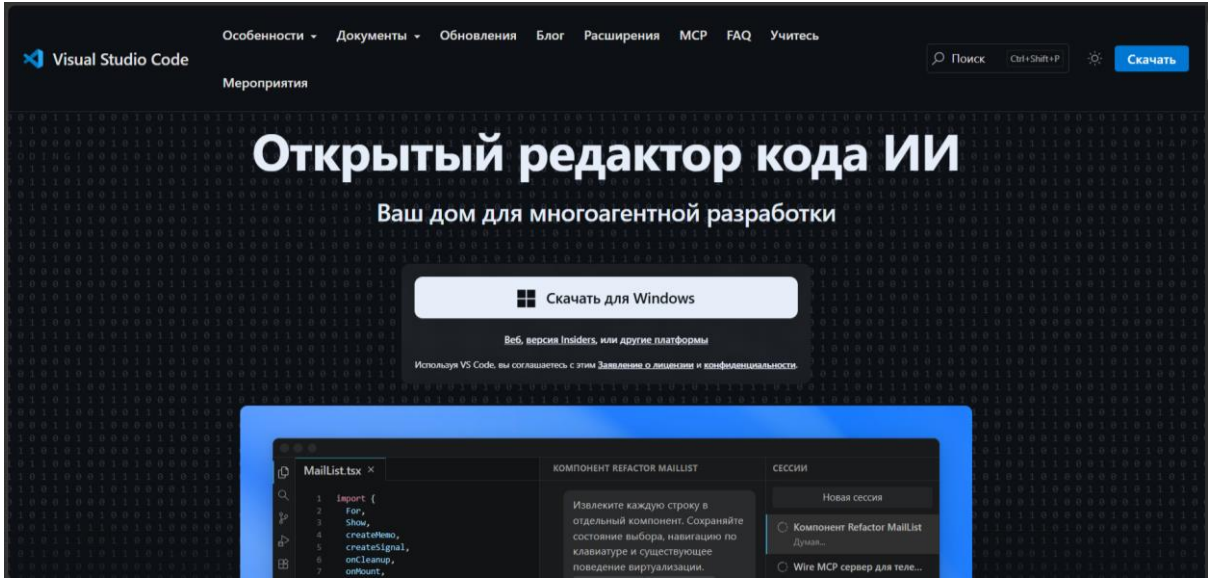


Рисунок 12 — Сайт Visual Studio

Процесс установки и настройки выполнялся в следующей последовательности.

На первом этапе необходимо загрузить установочный файл Visual Studio Code с официального сайта. На главной странице ресурса расположена область загрузки, соответствующая операционной системе пользователя. Данный этап проиллюстрирован на рисунке 13.

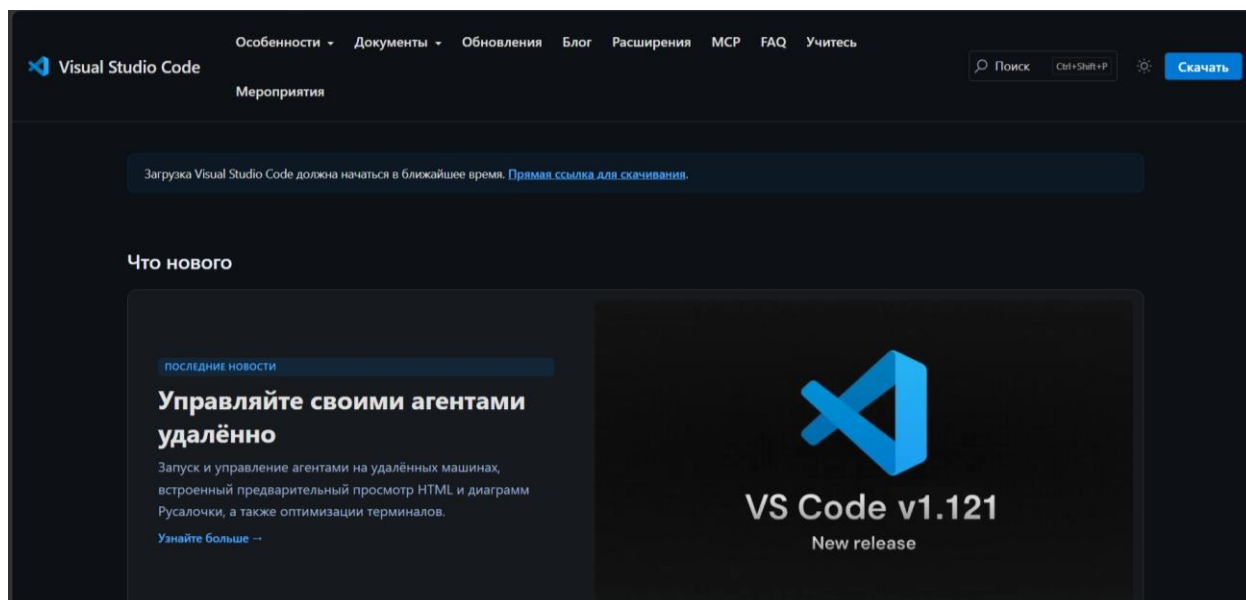


Рисунок 13 — Область загрузки Visual Studio Code с официального сайта

После нажатия кнопки загрузки запускается получение исполняемого файла установщика. Ход загрузки отображается в интерфейсе браузера. По завершении загрузки получен файл VSCodeSetup.exe для ОС Windows.

Далее был запущен установочный файл. На начальном этапе установки открывается окно с лицензионным соглашением, после принятия которого предлагается выбрать путь установки и компоненты. Для работы проекта были оставлены параметры по умолчанию. Внешний вид окна установки показан на рисунке 14.

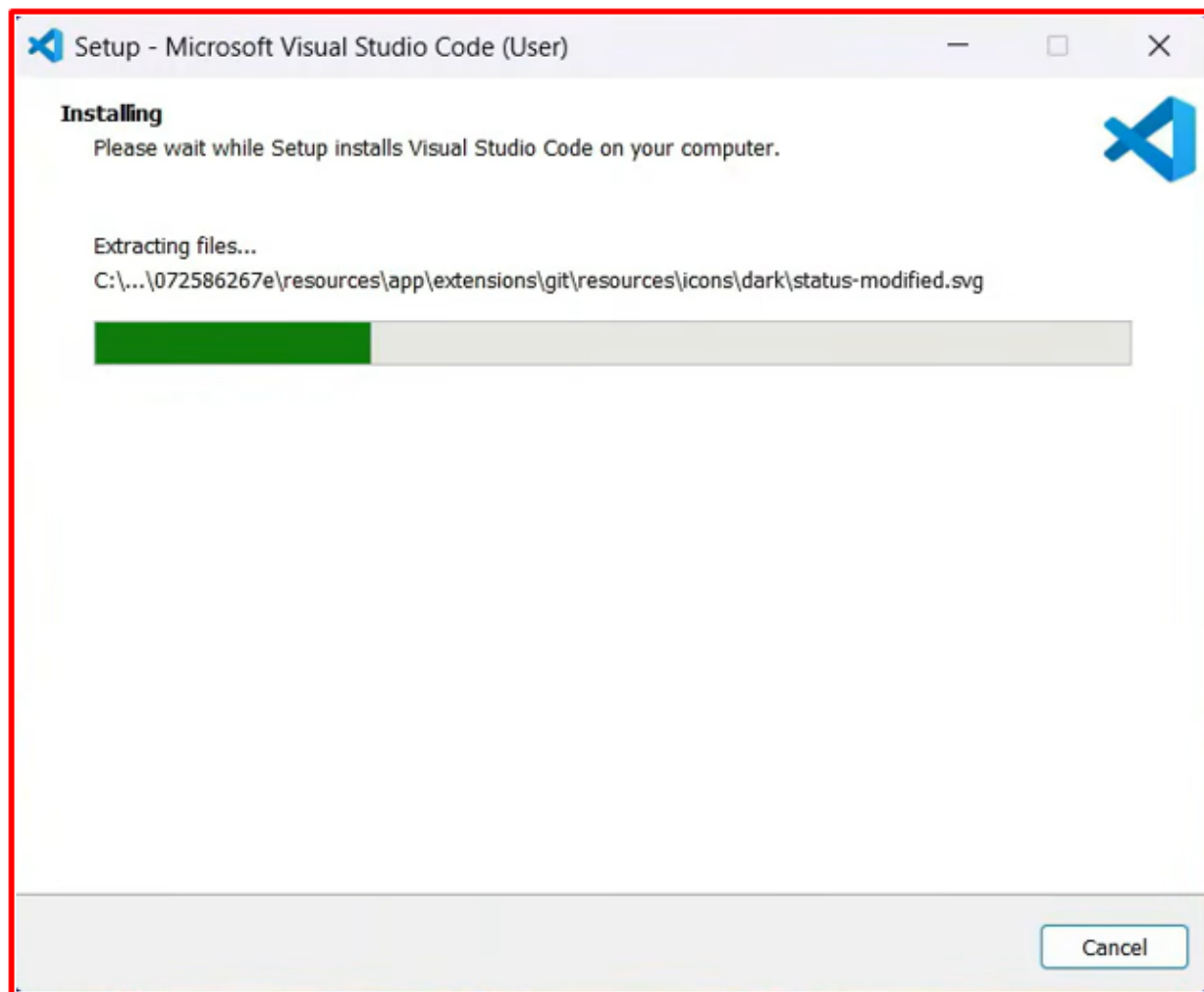


Рисунок 14 — Процесс установки Visual Studio Code

После завершения инсталляции среда становится доступной для запуска через главное меню или рабочий стол.

Затем был выполнен запуск Visual Studio Code. При первом открытии интерфейс имеет минималистичный вид, что удобно для дальнейшей настройки под нужды проекта. Главное окно VS Code изображено на рисунке 15.

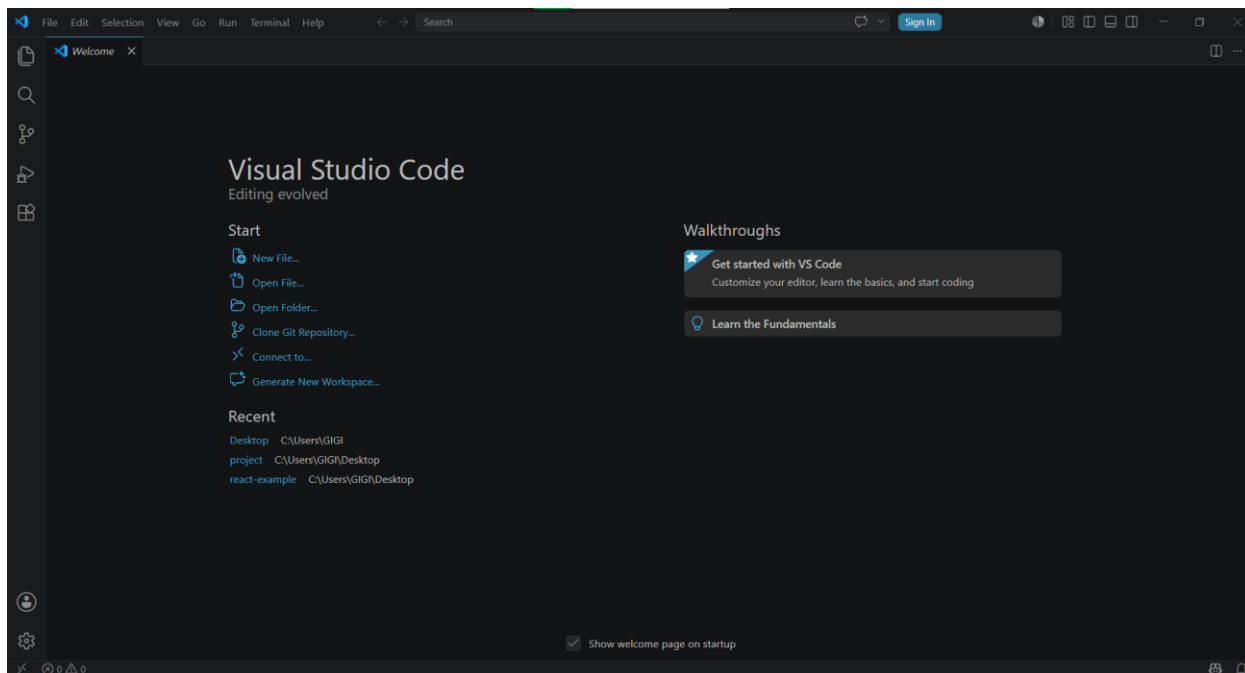


Рисунок 15 — Главное окно Visual Studio Code после установки

После завершения установки была выполнена проверка корректности работы. Для этого в терминале Visual Studio Code были последовательно введены команды. Для разработки дипломного проекта в Visual Studio Code были дополнительно установлены следующие расширения через встроенный магазин. Процесс установки расширений в VS Code показан на рисунке 16.

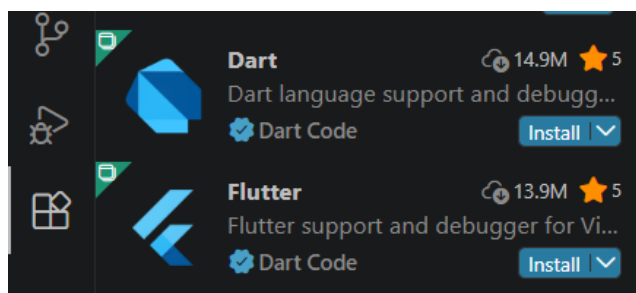


Рисунок 16 — Управление расширениями в Visual Studio Code

1.7 Проектирование и разработка

Проект представляет собой кроссплатформенное мобильное приложение «StatLife», разработанное для улучшения качества жизни пользователя путём комплексного контроля физиологических показателей и

поведенческих привычек. Приложение включает в себя учёт потребления воды, калорийности питания, продолжительности сна, динамики веса, а также отслеживание прогресса по привычкам с системой достижений и серий. Приложение создано с использованием локальной архитектуры: клиентская часть реализована на Flutter с использованием языка Dart, данные хранятся в локальной базе данных SQLite, что обеспечивает полную автономность, конфиденциальность данных и возможность работы без подключения к интернету.

Приложение состоит из следующих основных программных модулей: модуль авторизации, трекер воды, трекер питания, трекер сна, трекер веса, трекер привычек, система достижений, модуль визуализации данных (графики и календари), модуль экспорта данных (CSV, PDF) и резервного копирования, а также модуль настроек и уведомлений. Структурная организация программы отражена на диаграмме классов, представленной на рисунке 17.

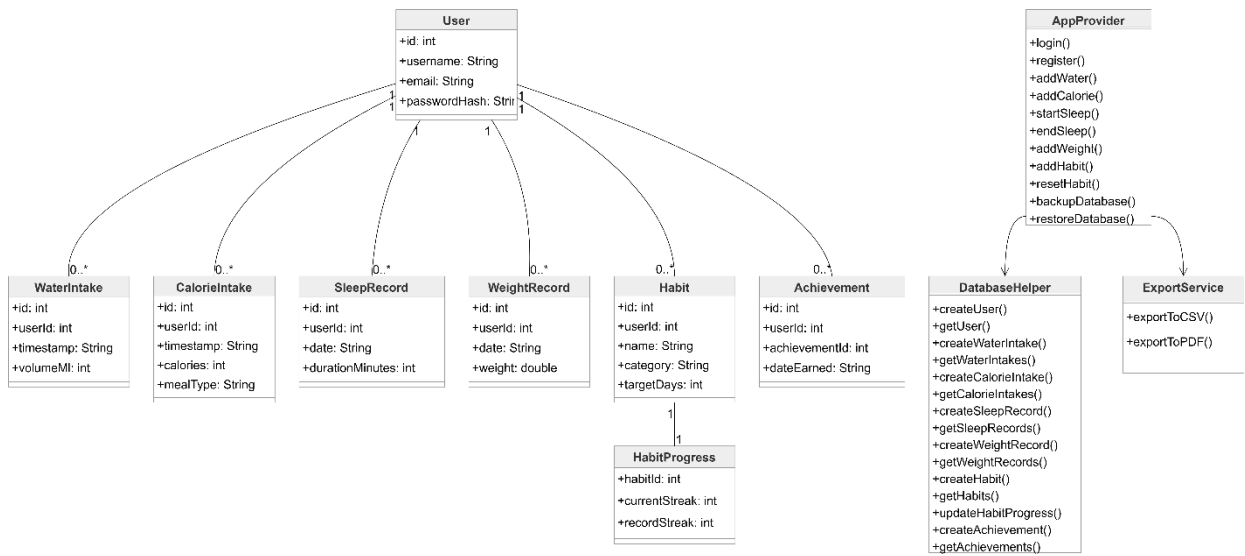


Рисунок 17 — Диаграмма классов

Графический интерфейс приложения StatLife представляет собой экраны мобильного приложения, доступные через сенсорное управление на устройствах Android и iOS. Интерфейс включает следующие основные экраны: главный экран (сводка) с карточками трекеров, отображающими

краткую информацию по каждому показателю; экран трекера воды с круговым индикатором прогресса, кнопками добавления и списком записей; экран трекера питания с отображением суммарных калорий, индикаторами БЖУ и формой добавления приёмов пищи; экран трекера сна с кнопками «Начать сон» и «Проснулся», а также календарём; экран трекера веса с графиком изменения веса и списком записей; экран трекера привычек с созданием привычек, отображением серий и рекордов; экран достижений с сеткой всех 100 наград; экран графиков и календарей; экран настроек; экран авторизации и регистрации.

На экранах приложения выводятся следующие данные:

- для трекера воды: количество выпитой воды за день, прогресс выполнения нормы, список записей с временем и объёмом, календарь с цветовой индикацией;
 - для трекера питания: суммарные калории за день, белки, жиры, углеводы, список приёмов пищи с указанием калорий и типа приёма, календарь с цветовой индикацией;
 - для трекера сна: продолжительность последнего сна, календарь с цветовой индикацией дней, где продолжительность сна соответствовала целевой;
 - для трекера привычек: текущая серия, рекордная серия, прогресс-бар цели, кнопки сброса серии и удаления привычки;
- для экрана достижений: список всех 100 достижений с цветовой индикацией полученных и недоступных.

Пользователь взаимодействует с программой через набор управляющих элементов сенсорного интерфейса, инициируя такие операции, как регистрация и вход в систему, добавление воды, добавление приёма пищи, начало и завершение сессии сна, добавление веса, создание привычек, сброс серии привычки, просмотр достижений, экспорт данных в CSV и PDF,

резервное копирование и восстановление базы данных, настройка целей и темы оформления.

Автоматизация сбора и обработки данных реализована с применением локальной архитектуры: бизнес-логика на Dart (AppProvider), база данных SQLite с обёрткой sqflite, а также SharedPreferences для хранения настроек и состояния сессии. AppProvider обрабатывает запросы от пользовательского интерфейса, выполняет валидацию данных, расчёт продолжительности сна, серий привычек и проверку условий достижений, взаимодействует с базой данных через DatabaseHelper. После получения данных экраны перерисовываются, отображая обновлённую информацию. Предусмотрена цветовая индикация прогресса выполнения норм и статусов (вода, калории, сон), что упрощает визуальный анализ.

Структура базы данных приложения отражена на диаграмме, представленной на рисунке 18.

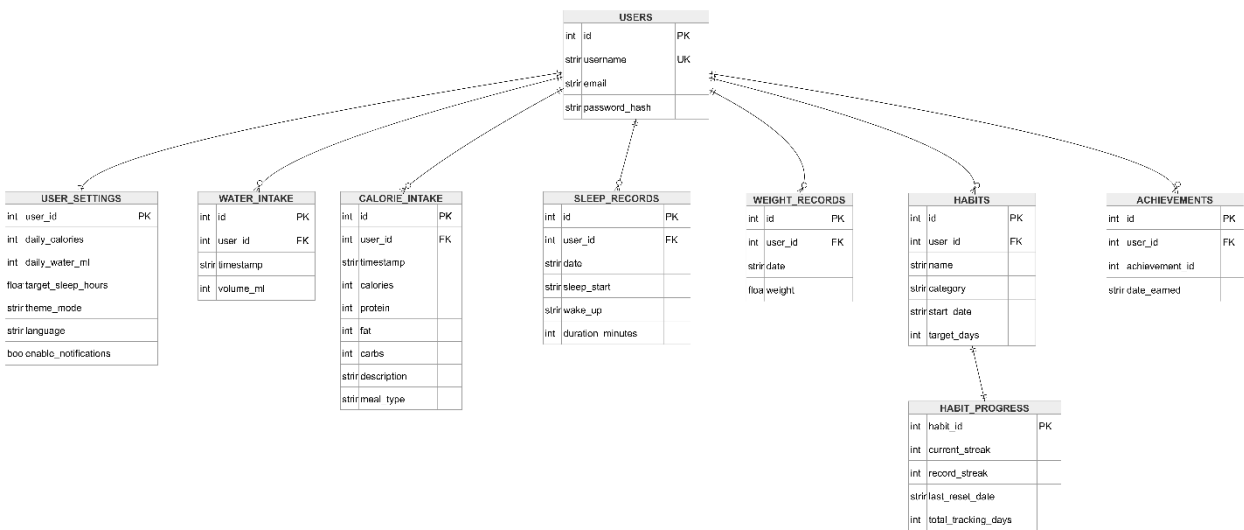


Рисунок 18 — Диаграмма базы данных

Пользователь имеет возможность просмотреть детальную информацию по любому трекеру (вода, питание, сон, вес, привычки), а также по каждому достижению в отдельности, включая его полное описание. Для привычек

доступен просмотр текущей и рекордной серий, а также прогресса выполнения целевой длительности.

Дипломный проект автоматизирует процессы учёта воды, питания, сна, веса и привычек, а также формирование наглядной статистики и мотивацию пользователя через систему достижений и серий, что способствует повышению эффективности самоконтроля здоровья, сокращению временных затрат на ведение учёта и улучшению качества жизни пользователя.

1.8 Разработка управляющей программы

Разработанное в рамках дипломного проекта мобильное приложение StatLife построено на локальной архитектуре без использования серверной части. Все данные хранятся непосредственно на устройстве пользователя в базе данных SQLite, а бизнес-логика реализована на языке Dart с использованием фреймворка Flutter. Центральными элементами архитектуры являются класс AppProvider, управляющий состоянием приложения, и класс DatabaseHelper, обеспечивающий взаимодействие с базой данных. Модели данных представлены классами-сущностями User, UserSettings, WaterIntake, CalorieIntake, SleepRecord, WeightRecord, Habit, HabitProgress и Achievement, которые преобразуются в таблицы базы данных через пакет sqflite.

Инициализация приложения. При запуске приложения вызывается функция main() в файле main.dart. Она выполняет следующие действия: принудительно устанавливает портретную ориентацию экрана, инициализирует локализацию для русского языка, после чего запускает приложение, обернутое в MultiProvider, где создаётся экземпляр AppProvider. Корневой компонент StatLifeApp в зависимости от состояния авторизации пользователя отображает либо экран входа (LoginScreen), либо главный экран с навигацией (MainNavigator).

Управление состоянием. Класс AppProvider наследуется от ChangeNotifier и выступает центральным хранилищем всех данных

пользователя. Он содержит списки записей о воде (waterIntakes), питании (calorieIntakes), сне (sleepRecords), весе (weightRecords), привычках (habits), прогрессе привычек (habitProgresses), а также список полученных достижений (earnedAchievements) и настройки пользователя (settings). При загрузке приложения вызывается метод checkLoginStatus, который считывает из SharedPreferences статус авторизации и, если пользователь был авторизован, загружает все его данные через метод _loadUserData. После загрузки данных вызывается метод AchievementsHelper.evaluate для проверки условий достижений и notifyListeners для обновления интерфейса.

Взаимодействие с базой данных. Класс DatabaseHelper реализован как синглтон. При первом обращении к базе данных создаётся файл statlife.db в директории приложения, а также выполняются SQL-запросы на создание всех таблиц: users, user_settings, water_intake, calorie_intake, sleep_records, weight_records, habits, habit_progress, achievements. Все методы DatabaseHelper являются асинхронными и возвращают Future, что позволяет не блокировать поток пользовательского интерфейса. Например, метод getWaterIntakes получает все записи о воде для конкретного пользователя, выполняя SELECT-запрос с фильтрацией по user_id. Метод createWaterIntake вставляет новую запись через INSERT-запрос.

Аутентификация и авторизация. Процесс аутентификации реализован в классе AppProvider. При регистрации пользователя метод register принимает имя пользователя, пароль и email. Пароль хешируется с помощью алгоритма SHA-256, после чего создаётся запись в таблице users через DatabaseHelper. При входе в систему метод login ищет пользователя по имени, сверяет хеш пароля и при успешной проверке сохраняет статус авторизации в SharedPreferences. Сессия пользователя автоматически восстанавливается при повторном запуске приложения. В приложении StatLife отсутствует разделение на роли (пользователь и администратор), так как все данные хранятся локально на устройстве.

Работа с трекерами. Добавление записи о воде выполняется методом `addWater`, который принимает объём в миллилитрах, проверяет его корректность (от 1 до 5000 мл), создаёт объект `WaterIntake` с текущей временной меткой и сохраняет его через `DatabaseHelper`. После сохранения вызывается `_loadUserData` для перезагрузки всех данных и `notifyListeners` для обновления интерфейса. Аналогично работают методы `addCalorie` (добавление приёма пищи с указанием калорий, белков, жиров, углеводов, типа приёма и описания), `startSleep` и `endSleep` (начало и завершение сессии сна с автоматическим расчётом продолжительности), `addWeight` (добавление записи о весе с проверкой диапазона от 20 до 300 кг).

Работа с привычками. Создание привычки выполняется методом `addHabit`, который принимает название, категорию, дату отказа и целевую длительность. При создании привычки автоматически создаётся запись в таблице `habit_progress` с нулевыми сериями. Метод `resetHabit` сбрасывает текущую серию привычки при срыве: текущая серия обнуляется, рекордная серия сохраняется, а дата последнего сброса обновляется. Метод `_recalculateHabitStreaks` вызывается при каждой загрузке данных и пересчитывает текущие серии на основе даты последнего сброса и текущей даты.

Система достижений. Система достижений реализована в классе `AchievementsHelper` статическим методом `evaluate`. Этот метод вызывается после каждого обновления данных пользователя. Он получает список уже полученных достижений из базы данных и последовательно проверяет условия для каждого из 100 достижений. Проверка обёрнута во вспомогательную функцию `check`, которая принимает идентификатор достижения и логическое условие. Если условие истинно и достижение ещё не получено, вызывается метод `awardAchievement` из `AppProvider`, который сохраняет достижение в базу данных, добавляет его в список `_earnedAchievements` и отправляет локальное уведомление с названием

достижения. Примеры условий: пробуждение до 6:00 (достижение №2 «Ранняя пташка»), сон ровно 8 часов (№4 «Золотая середина»), выполнение дневной нормы воды (№18 «Дневная норма»), 7 дней без срыва у любой привычки (№48 «Неделя чистоты»), одновременное выполнение норм воды, калорий и сна (№61 «Здоровый день»).

Экспорт данных и резервное копирование. Экспорт данных реализован в классе `ExportService`. Метод `exportToCSV` запрашивает у пользователя директорию через `FilePicker`, затем для каждой таблицы (`water_intake`, `calorie_intake`, `sleep_records`, `habits`, `habit_progress`, `achievements`) получает все записи из базы данных, формирует строки CSV (первая строка — заголовки, последующие — значения полей) и сохраняет файлы с именами `statlife_имя_таблицы.csv`. Метод `exportToPDF` генерирует отчет в формате PDF, содержащий имя пользователя и количество записей по каждому треке. Резервное копирование реализовано методом `backupDatabase` в `AppProvider`: он получает путь к файлу `statlife.db`, копирует его в выбранную пользователем директорию с именем, содержащим временную метку. Восстановление выполняется методом `restoreDatabase`: после выбора файла `.db` текущее соединение с базой данных закрывается, выбранный файл копируется на место `statlife.db`, после чего выполняется выход из аккаунта.

Уведомления. Уведомления реализованы в классе `NotificationService` с использованием пакета `flutter_local_notifications`. При инициализации сервиса настраиваются каналы уведомлений для Android и iOS. Метод `scheduleDailyNotification` планирует ежедневное уведомление на указанное время (например, напоминание о воде в 10:00 или будильник). При получении нового достижения вызывается метод `showNotification`, который мгновенно отображает уведомление с текстом «Поздравляем! Вы получили достижение: название». При отключении уведомлений в настройках все запланированные уведомления отменяются через метод `cancelAllNotifications`.

2 Демонстрация работы приложения

2.1 Регистрация и вход

При первом запуске приложение показывает экран входа с двумя вкладками: «Вход» и «Регистрация». Пользователь вводит имя, email и пароль (не менее 6 символов), нажимает «Зарегистрироваться». Система проверяет уникальность имени, хеширует пароль и сохраняет пользователя в БД. После успешной регистрации происходит автоматический вход, открывается главный экран «Сводка». При повторном запуске приложение восстанавливает сессию через SharedPreferences.

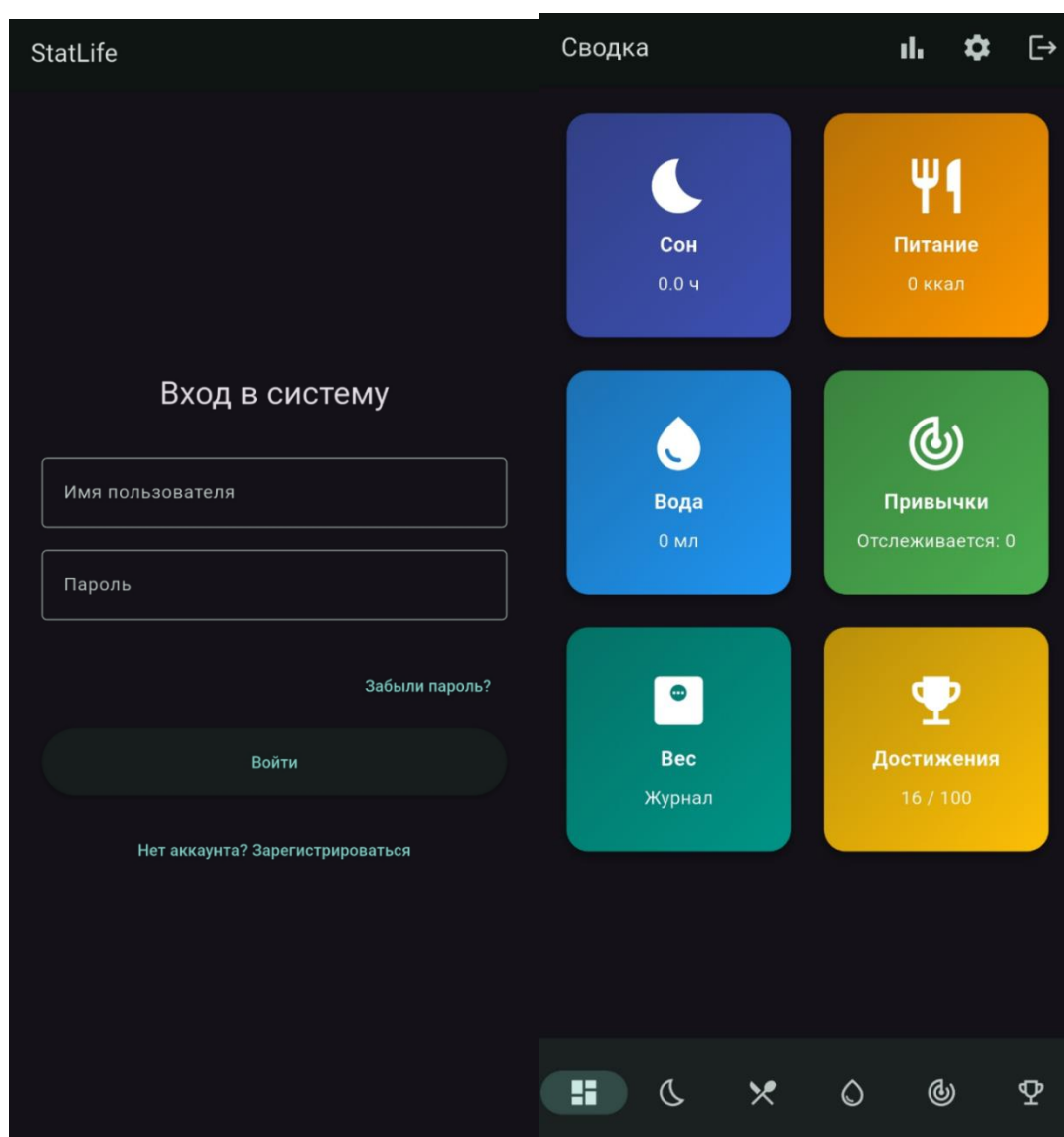


Рисунок 19 — Регистрация и главный экран

2.2 Добавление записи о воде

Пользователь переходит на экран «Вода» через нижнюю навигацию. Нажимает кнопку «+250 мл». Приложение добавляет запись в БД с текущей временной меткой. Круговой индикатор обновляется, показывая прогресс (например, 500/2500 мл). При желании пользователь может добавить произвольный объём через кнопку «Свой объём». Записи можно удалить свайпом вправо.

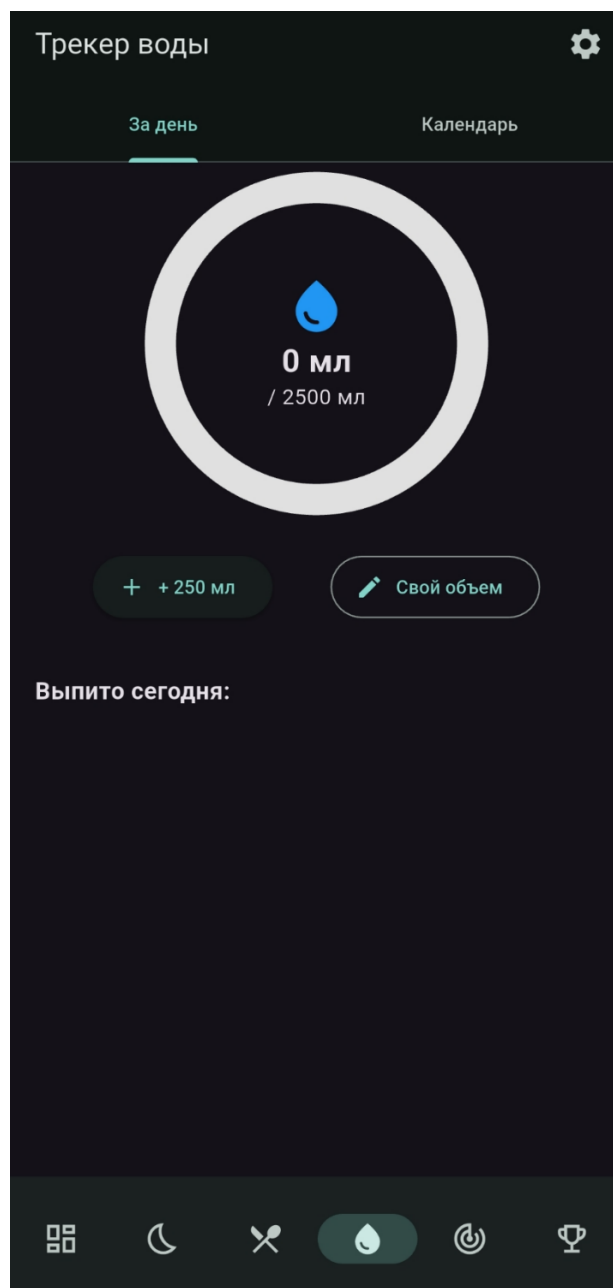


Рисунок 20 — Модуль отслеживания воды

2.3 Добавление приёма пищи

Пользователь переходит на экран «Питание». Нажимает кнопку «+» (FloatingActionButton). Заполняет форму: выбирает тип приёма (завтрак/обед/ужин/перекус), вводит калории, опционально – БЖУ и описание. Нажимает «Добавить». Запись сохраняется, на экране появляется карточка с информацией. Суммарные калории за день обновляются, прогресс-бар пересчитывается. Пользователь может перейти на вкладку «Календарь», чтобы увидеть цветовую индикацию выполнения нормы по дням.

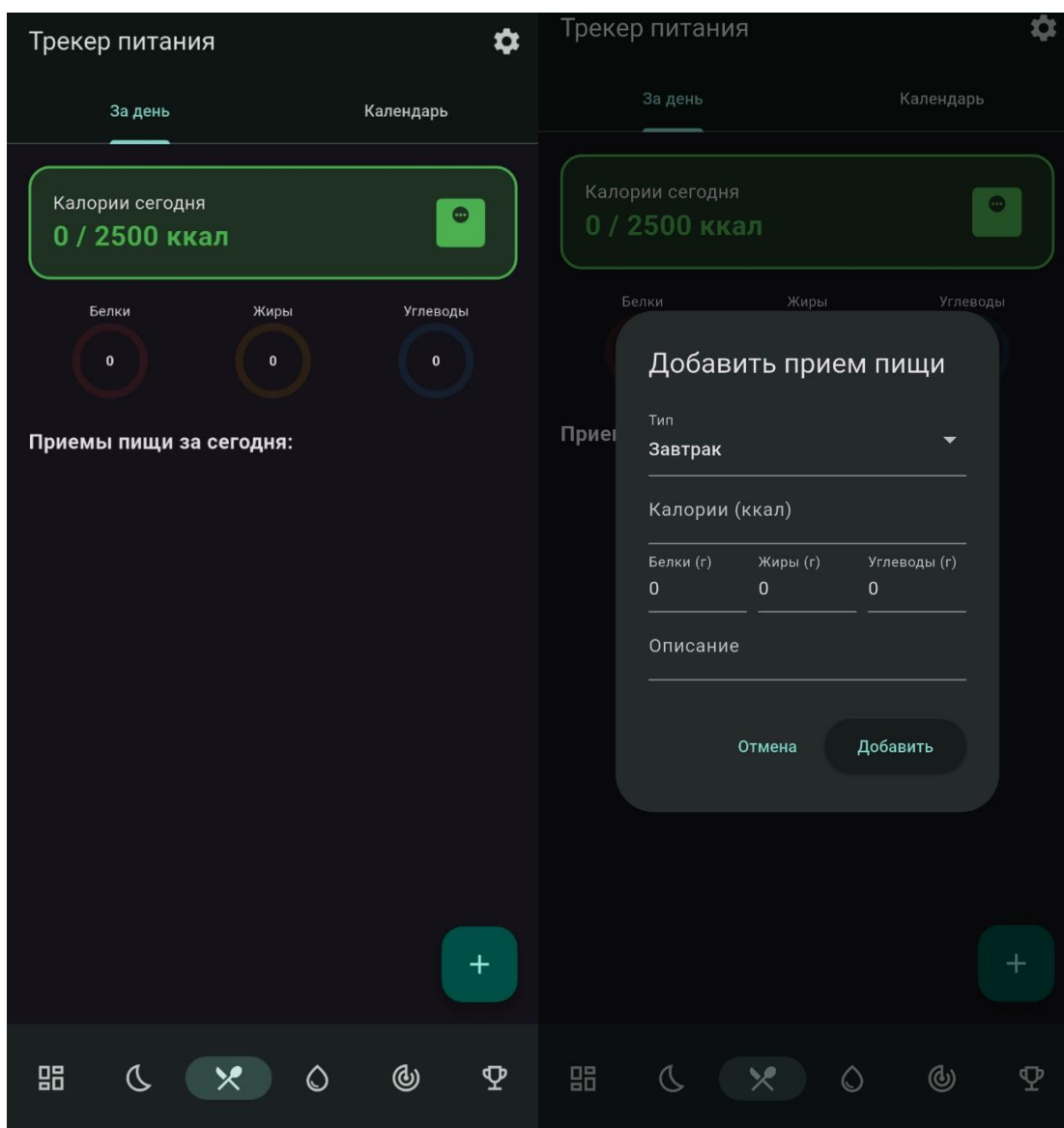


Рисунок 21 — Модуль отслеживания ккалорий

Изм.	Лист	№ докум.	Подпись	Дата

2.4 Отслеживание сна

Пользователь переходит на экран «Сон». Нажимает кнопку «Начать сон» перед сном. Приложение фиксирует время засыпания (с учётом +20 мин на засыпание). Утром пользователь нажимает «Проснулся». Приложение автоматически рассчитывает продолжительность сна (разница между пробуждением и засыпанием), сохраняет запись. На календаре день окрашивается в зелёный цвет, если продолжительность соответствует целевой (± 30 мин), или в красный – если нет.

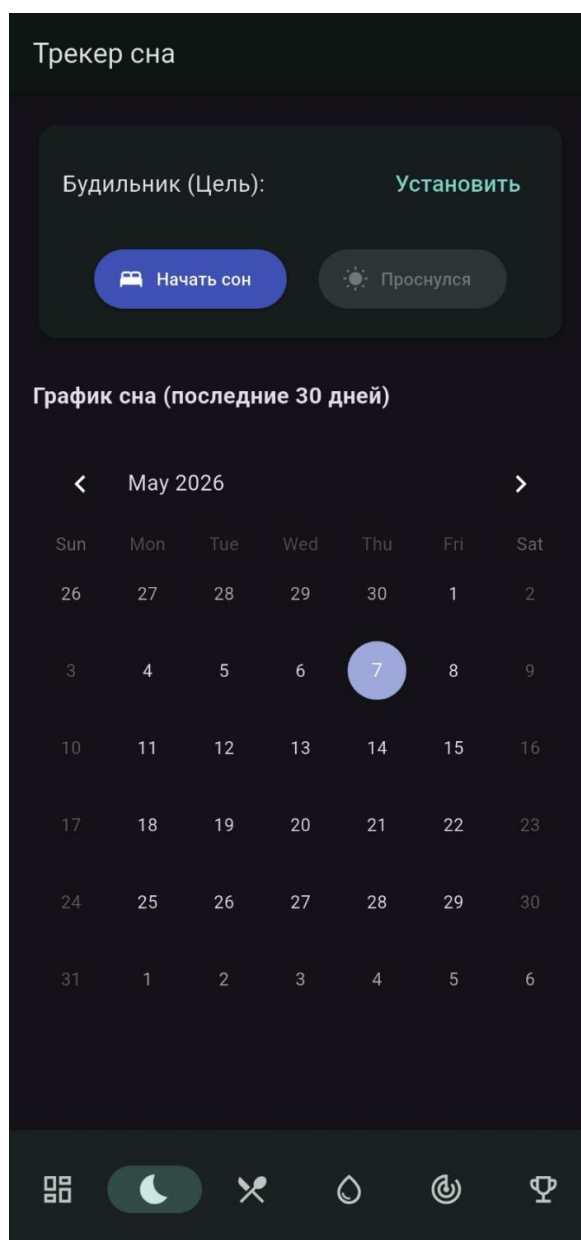


Рисунок 22 — Модуль отслеживания сна

2.5 Работа с привычками

Пользователь переходит на экран «Привычки». Нажимает кнопку «+» для создания новой привычки (например, «Не курю», категория «Курение»). Указывает дату отказа (сегодня) и цель (опционально, например, 30 дней). Привычка появляется в списке с текущей серией = 0. Каждый день серия автоматически увеличивается (на основе даты последнего сброса). При срыве пользователь нажимает кнопку «Сбросить», текущая серия обнуляется, рекорд сохраняется. Если целевая длительность задана, отображается прогресс-бар.

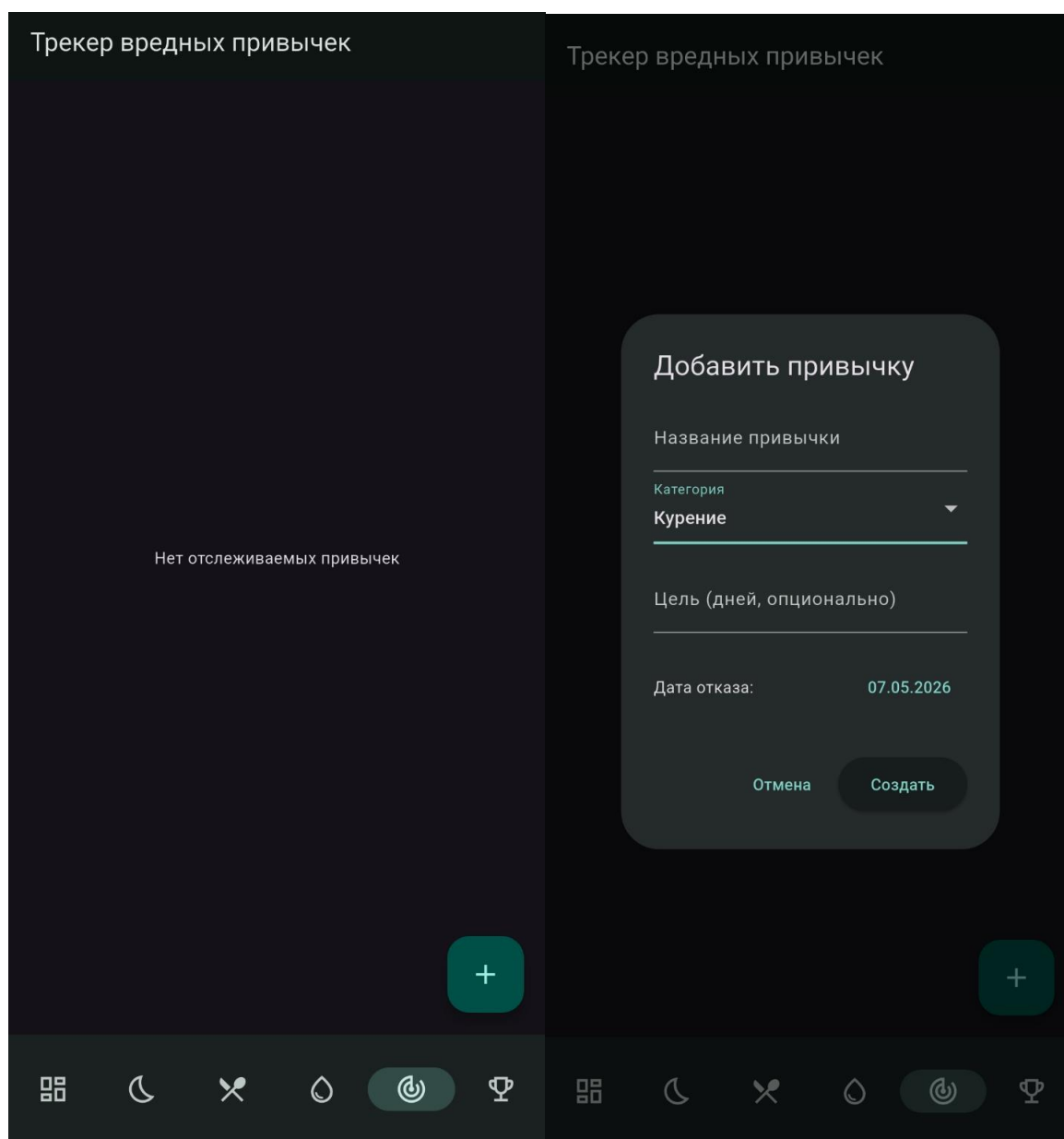


Рисунок 23 — Модуль отслеживания привычек

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.43.001 ПЗ

Лист

53

2.6 Получение достижения

Пользователь выполняет условие для достижения (например, выпивает 2500 мл воды, что соответствует норме). AchievementsHelper автоматически проверяет условия после добавления записи. При выполнении условия выдаётся достижение (например, «Дневная норма»). Пользователь видит всплывающее уведомление. На экране «Достижения» иконка достижения становится цветной и активной.

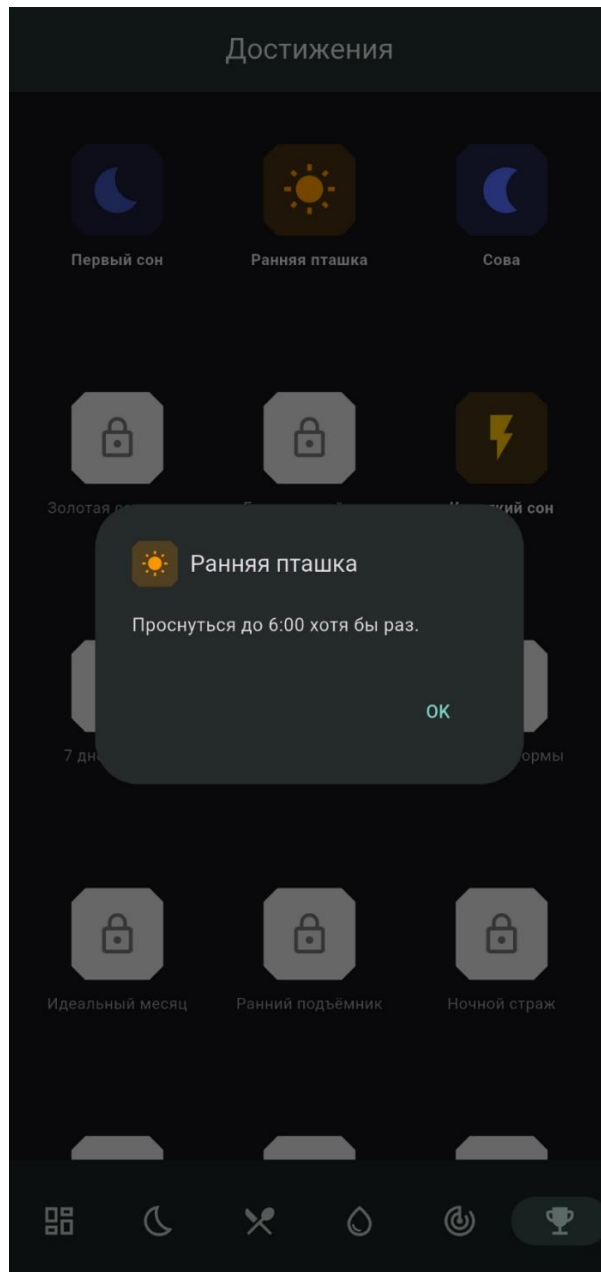


Рисунок 24 — Достижения

2.7 Экспорт данных

Пользователь переходит в раздел «Настройки». Выбирает «Экспорт в CSV» или «Отчёт в PDF». Система через `file_picker` предлагает выбрать директорию для сохранения. После выбора папки генерируются файлы (отдельные CSV для каждой таблицы или PDF-отчёт). Пользователь видит сообщение об успешном экспорте.

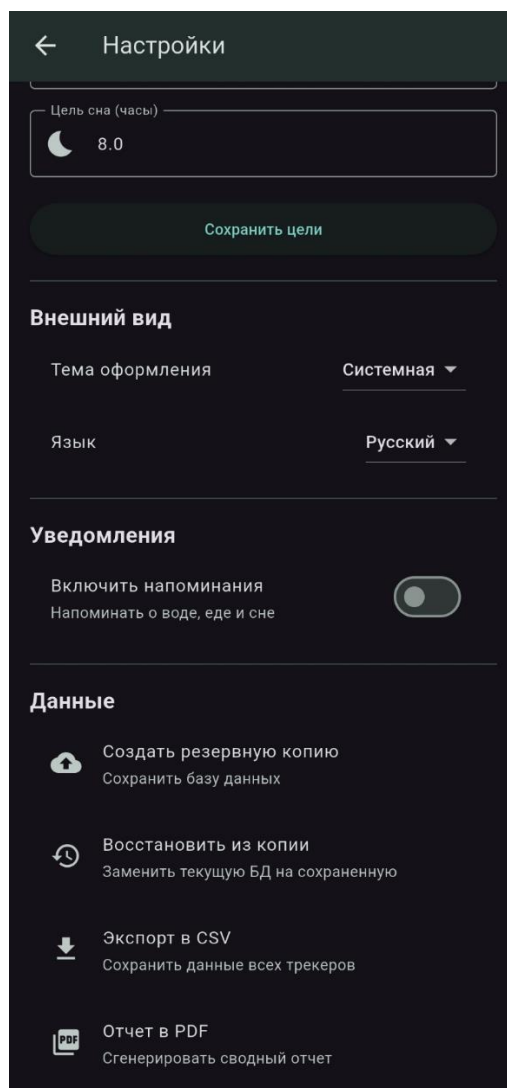


Рисунок 25 — Возможные действия с данными

2.8 Заключение по демонстрации

Все реализованные сценарии демонстрируют полную работоспособность приложения StatLife. Интерфейс интуитивно понятен, все

					ПГУ1.090301.43.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		55

действия пользователя обрабатываются корректно, данные сохраняются и извлекаются без ошибок. Система достижений и уведомлений успешно мотивирует пользователя к регулярному использованию приложения.

2 Требования безопасности по охране труда

					ПГУ1.090301.43.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		56

3 Общие требования безопасности

При разработке и использовании мобильного приложения StatLife необходимо соблюдать общие требования охраны труда, направленные на предотвращение профессиональных заболеваний и травматизма. К основным общим требованиям относятся:

- соблюдение режима труда и отдыха, исключающего длительную непрерывную работу без перерывов;
- обеспечение правильной организации рабочего места в соответствии с эргономическими нормами;
- поддержание оптимального микроклимата в помещении (температура 22-24°C, влажность 40-60%);
- регулярное проветривание помещения для предотвращения накопления углекислого газа;
- обеспечение достаточного уровня освещённости рабочей зоны (не менее 300-500 лк);
- прохождение обязательных медицинских осмотров при длительной работе за компьютером;
- своевременное информирование руководителя о любых неисправностях оборудования;
- соблюдение правил пожарной безопасности, наличие средств пожаротушения.

3.1 Требования безопасности при работе за ПК

Длительная работа за компьютером может привести к развитию таких заболеваний, как остеохондроз, синдром запястного канала, ухудшение зрения, хроническая усталость. Для минимизации рисков необходимо соблюдать следующие требования:

- правильная посадка: спина прямая, опирается на спинку стула, плечи расслаблены, локти согнуты под углом 90°, запястья прямые;

					ПГУ1.090301.43.001 ПЗ	Лист
						57
Изм.	Лист	№ докум.	Подпись	Дата		

- расстояние от глаз до монитора должно составлять 50-70 см, верхний край экрана должен находиться на уровне глаз или чуть ниже;
- стул должен иметь регулируемую высоту, подлокотники и поясничную поддержку для снижения нагрузки на позвоночник; клавиатура и мышь должны располагаться на одной поверхности на расстоянии 10-15 см от края стола;
- каждые 45-60 минут работы необходимо делать перерыв на 5-10 минут для разминки шеи, плеч, спины и кистей рук; для предотвращения зрительного утомления каждые 20 минут следует отводить взгляд от экрана на 20 секунд, фокусируясь на отдалённых предметах;
- яркость и контрастность монитора должны быть настроены комфортно для глаз, в вечернее время рекомендуется использовать режим «тёплый свет»;
- экран монитора должен регулярно очищаться от пыли специальными салфетками без спирта.

3.2 Требования безопасности при работе с мобильным телефоном

Разработка и тестирование мобильного приложения StatLife осуществлялась с использованием смартфонов на операционной системе Android. При длительной работе с мобильным устройством необходимо соблюдать следующие требования безопасности:

- при работе с телефоном следует держать его на расстоянии 30-40 см от глаз, избегая длительного приближения экрана к лицу; яркость экрана должна быть адаптирована к условиям окружающего освещения — не слишком тусклой, чтобы не напрягать зрение, и не слишком яркой, чтобы не вызывать дискомфорт;
- каждые 30-40 минут использования мобильного устройства рекомендуется делать перерыв на 3-5 минут, чтобы дать глазам отдохнуть;

при длительном тестировании приложения (более 2 часов) следует использовать подставку для телефона, чтобы избежать статического напряжения в шее и плечах;

- не рекомендуется использовать мобильное устройство во время зарядки, так как это может привести к перегреву батареи и снижению безопасности;

- следует избегать падений и механических повреждений устройства, использовать защитные чехлы и стёкла;

- при работе с телефоном в положении лёжа или сидя со сгорбленной спиной нагрузка на позвоночник возрастает в несколько раз, поэтому рекомендуется держать устройство на уровне глаз;

- регулярно очищать экран и корпус устройства от загрязнений, так как накопление бактерий и пыли может вызывать раздражение кожи и аллергические реакции;

- не следует использовать мобильное устройство непрерывно более 2 часов без перерыва на физическую активность и разминку.

3.3 Меры пожарной безопасности

Помещения, в которых осуществляется разработка программного обеспечения, относятся к категории пожароопасных, так как содержат электрооборудование. Основные требования пожарной безопасности:

- запрещается использовать повреждённые розетки, удлинители и сетевые фильтры;

- не допускается оставлять включённое электрооборудование без присмотра на длительное время;

- запрещается закрывать вентиляционные отверстия системного блока и мобильных устройств;

- в помещении должен находиться углекислотный или порошковый огнетушитель (марки ОУ или ОП);

					ПГУ1.090301.43.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		59

- при обнаружении запаха гари необходимо немедленно обесточить оборудование и сообщить ответственному лицу;
- пути эвакуации должны быть свободны от посторонних предметов;
- запрещается использовать для тушения электрооборудования воду и пенные огнетушители.

					ПГУ1.090301.43.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		60

Заключение

Разработанное в рамках выпускной квалификационной работы кроссплатформенное мобильное приложение «StatLife» для улучшения качества жизни представляет собой современное и востребованное решение в области цифрового здравоохранения и трекинга здоровья. В условиях современного общества, где время пользователя становится главной ценностью, а забота о здоровье и формирование полезных привычек выходят на первый план, подобное приложение может стать надёжным инструментом для повышения качества жизни, контроля физиологических показателей и мотивации пользователя к здоровому образу жизни.

Ожидается, что внедрение разработанной системы не только упростит процесс ведения учёта воды, питания, сна, веса и привычек, но и позволит пользователю получать наглядную аналитику о прогрессе, отслеживать динамику изменения показателей и получать мотивационную поддержку через систему достижений и серий. Система способна показать те нюансы в образе жизни пользователя, над которыми стоит поработать для улучшения самочувствия, повышения работоспособности и укрепления здоровья.

В дальнейшем проект может быть расширен путём добавления дополнительных функций, например, интеграции с облачным хранилищем для синхронизации данных между устройствами, подключения фитнес-браслетов и умных весов, добавления новых трекеров (физическая активность, артериальное давление, настроение), расширения системы достижений, а также создания веб-версии для просмотра статистики на персональном компьютере.

Список использованных источников

- 1 Flutter documentation [Электронный ресурс]. – URL: <https://docs.flutter.dev> (дата обращения: 01.06.2026)..
- 2 Dart language overview [Электронный ресурс]. – URL: <https://dart.dev/overview> (дата обращения: 01.06.2026).
- 3 SQLite documentation [Электронный ресурс]. – URL: <https://www.sqlite.org/docs.html> (дата обращения: 01.06.2026).
- 4 Михайлов А.Ю. Разработка кроссплатформенных мобильных приложений на Flutter / А.Ю. Михайлов. – М.: ДМК Пресс, 2024. – 380 с.
- 5 Вайл С. Dart. Разработка приложений на языке Google Dart / С. Вайл. – М.: Вильямс, 2022. – 512 с.
- 6 World Health Organization. Noncommunicable diseases. Fact sheet [Электронный ресурс]. – URL: <https://www.who.int/news-room/fact-sheets/detail/noncommunicable-diseases> (дата обращения: 01.06.2026).
- 7 Grand View Research. mHealth Market Size Report, 2024–2030 [Электронный ресурс]. – URL: <https://www.grandviewresearch.com> (дата обращения: 01.06.2026).

Приложение А

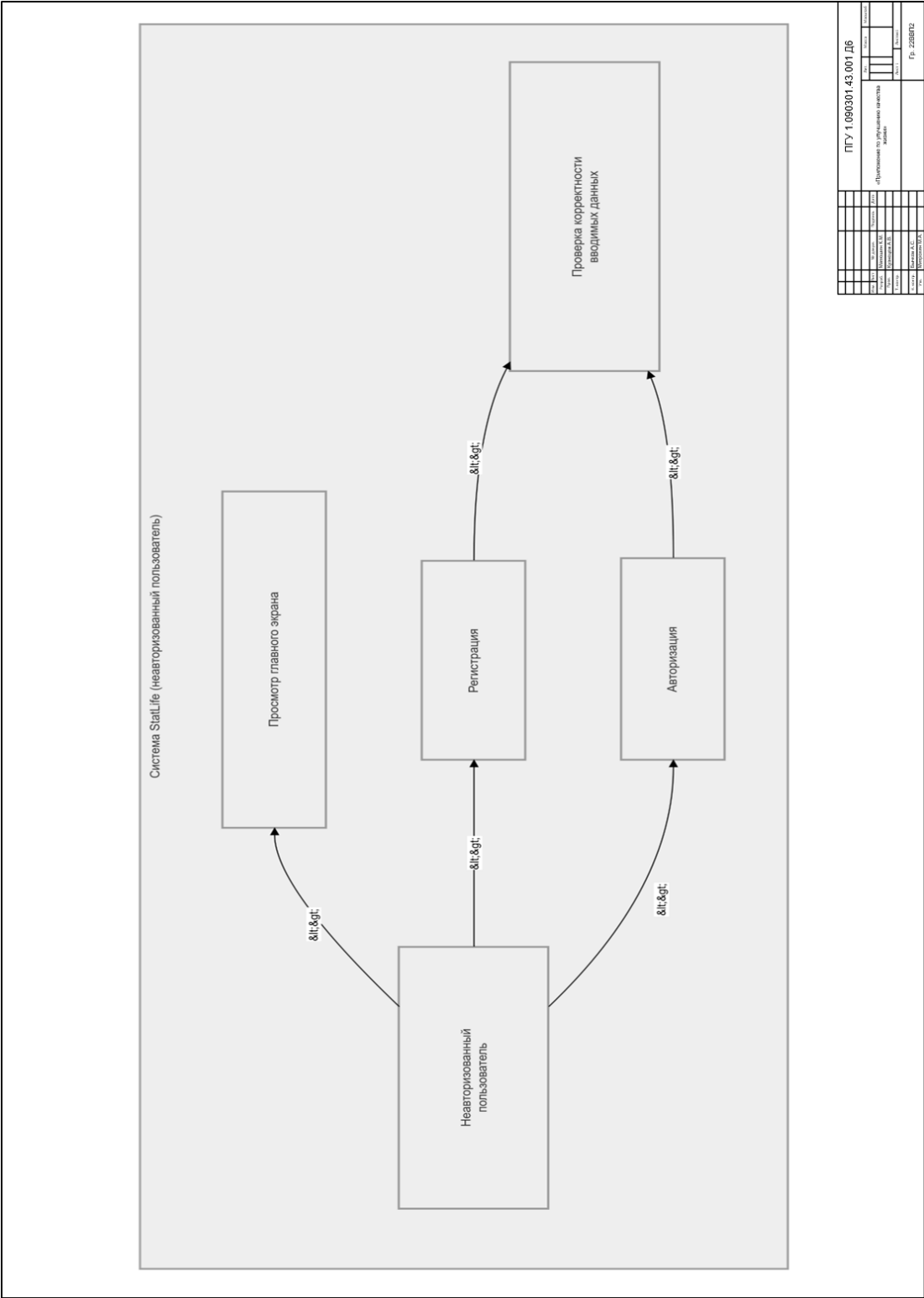
(обязательное)

Веб-приложение фитнес-центра

UML диаграммы и схемы

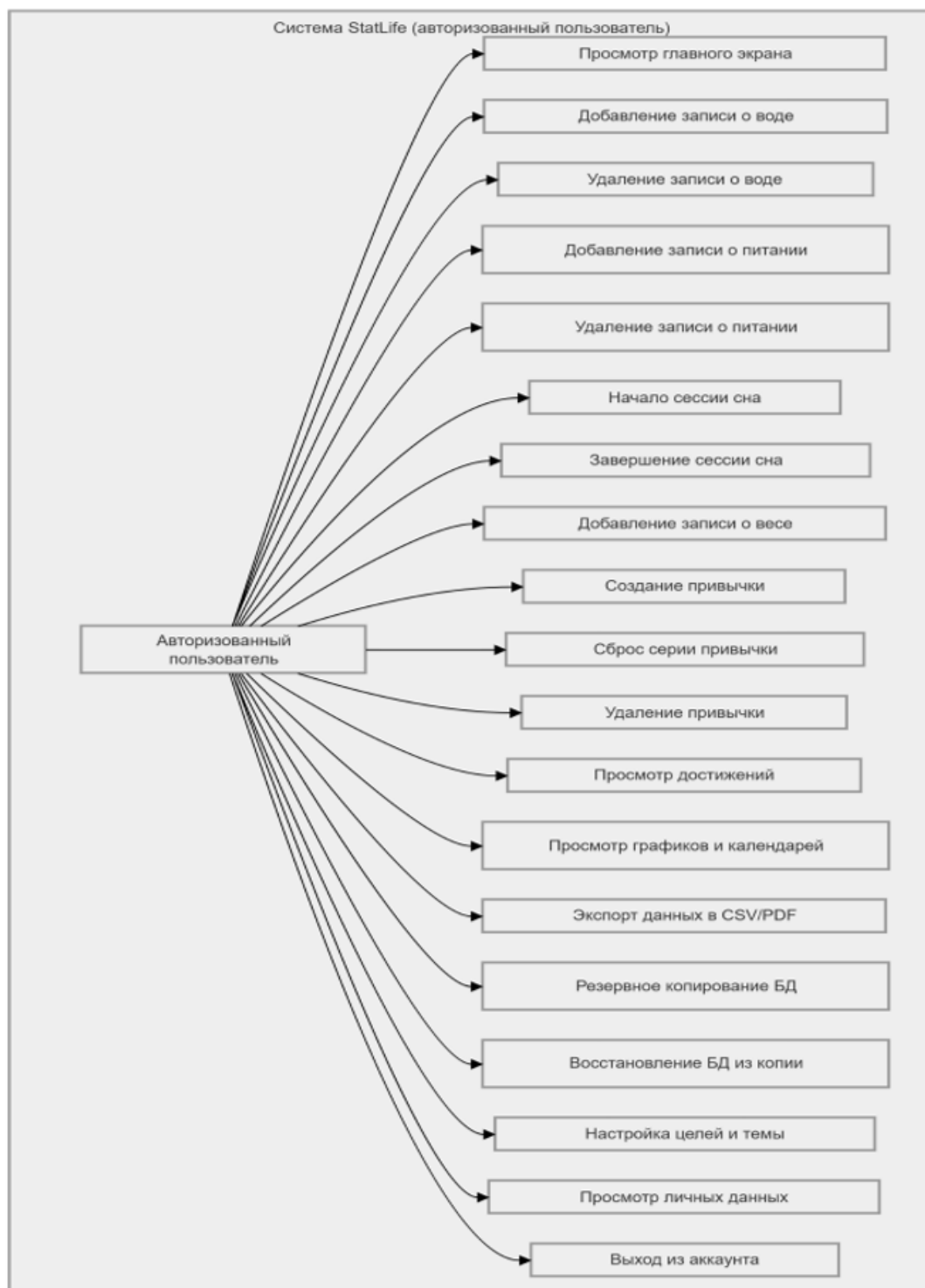
					ПГУ1.090301.43.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		63

Приложение А.1 – Диаграмма прецедентов неавторизованного пользователя



ПГУ 1.090301.43.001 ДБ									
«Программа по управлению активами»									
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Гр. 220802									

Приложение А.2 – Диаграмма прецедентов авторизованного пользователя

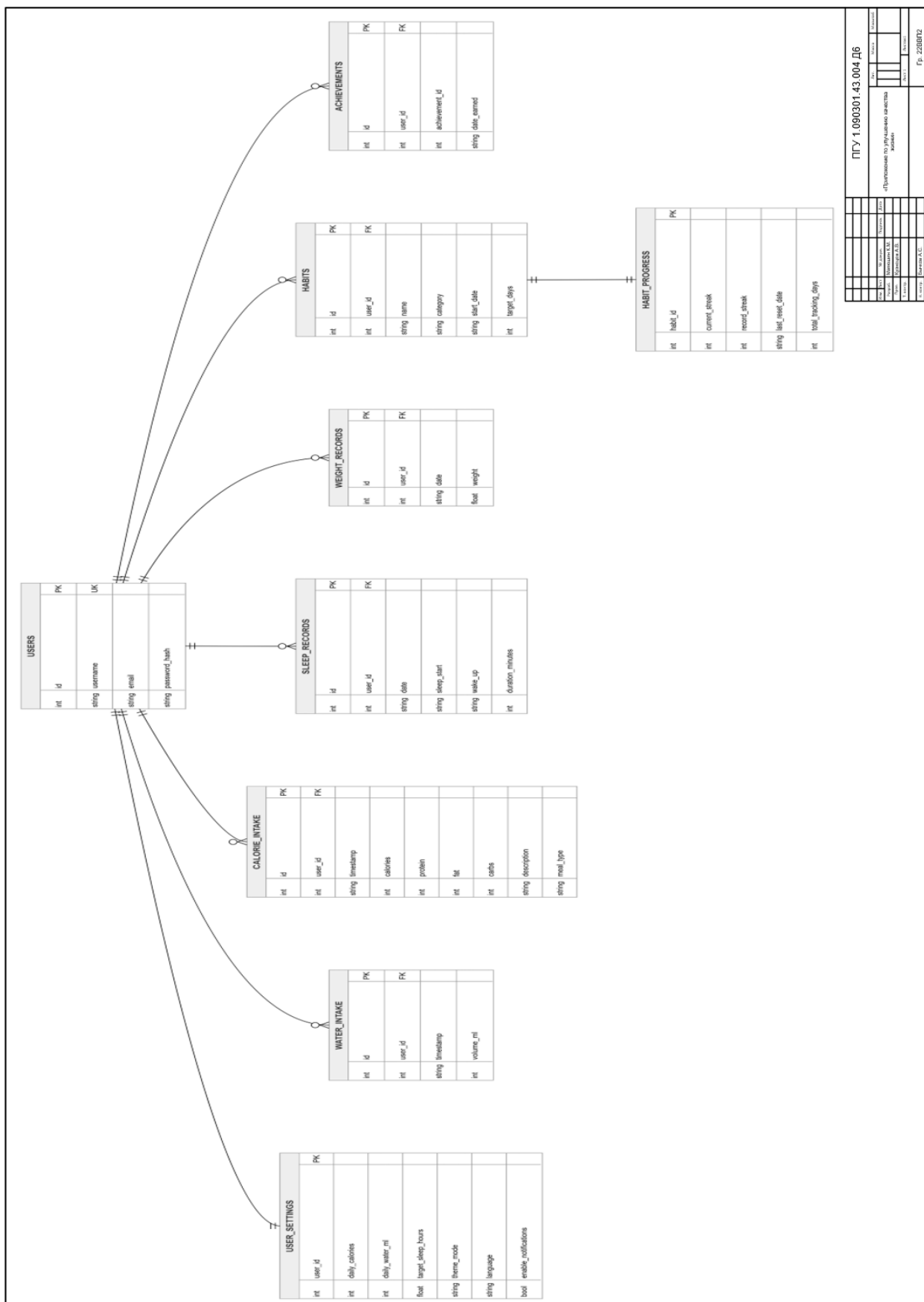


					ПГУ 1.090301.43.002 Д6				
Фамилия	Имя	М. достояние	Подпись	Дата	«Приложение по улучшению качества жизни»				
Подпись	Матвеева К.М.								
Подпись	Кузнецова А.В.								
Подпись									
					Взвешивание	Дозирование	Гр. 22ВВП2		
Подпись	Белкина А.С.								
Подпись	Матвеева Д.А.								

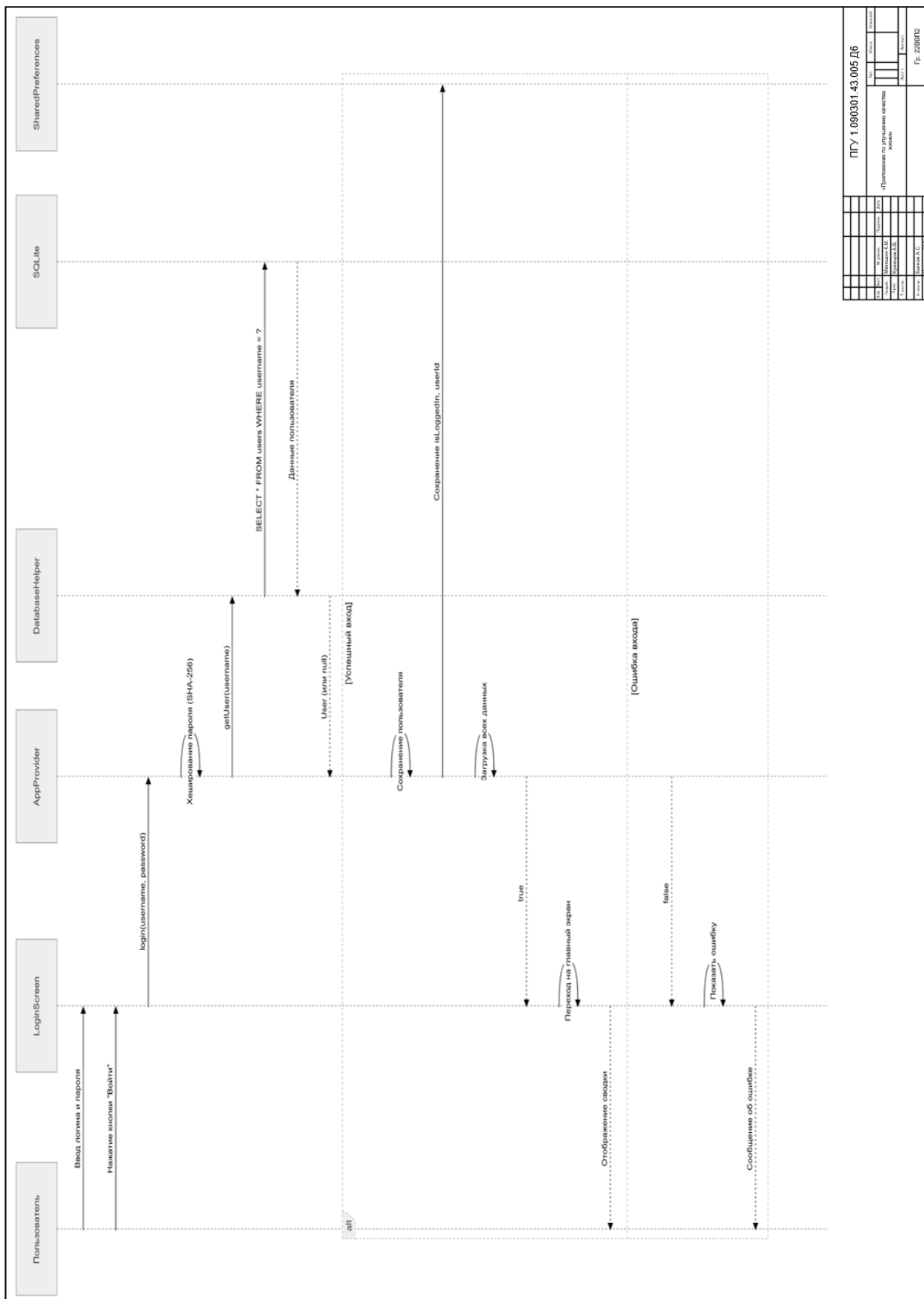
Приложение А.3 – Диаграмма классов



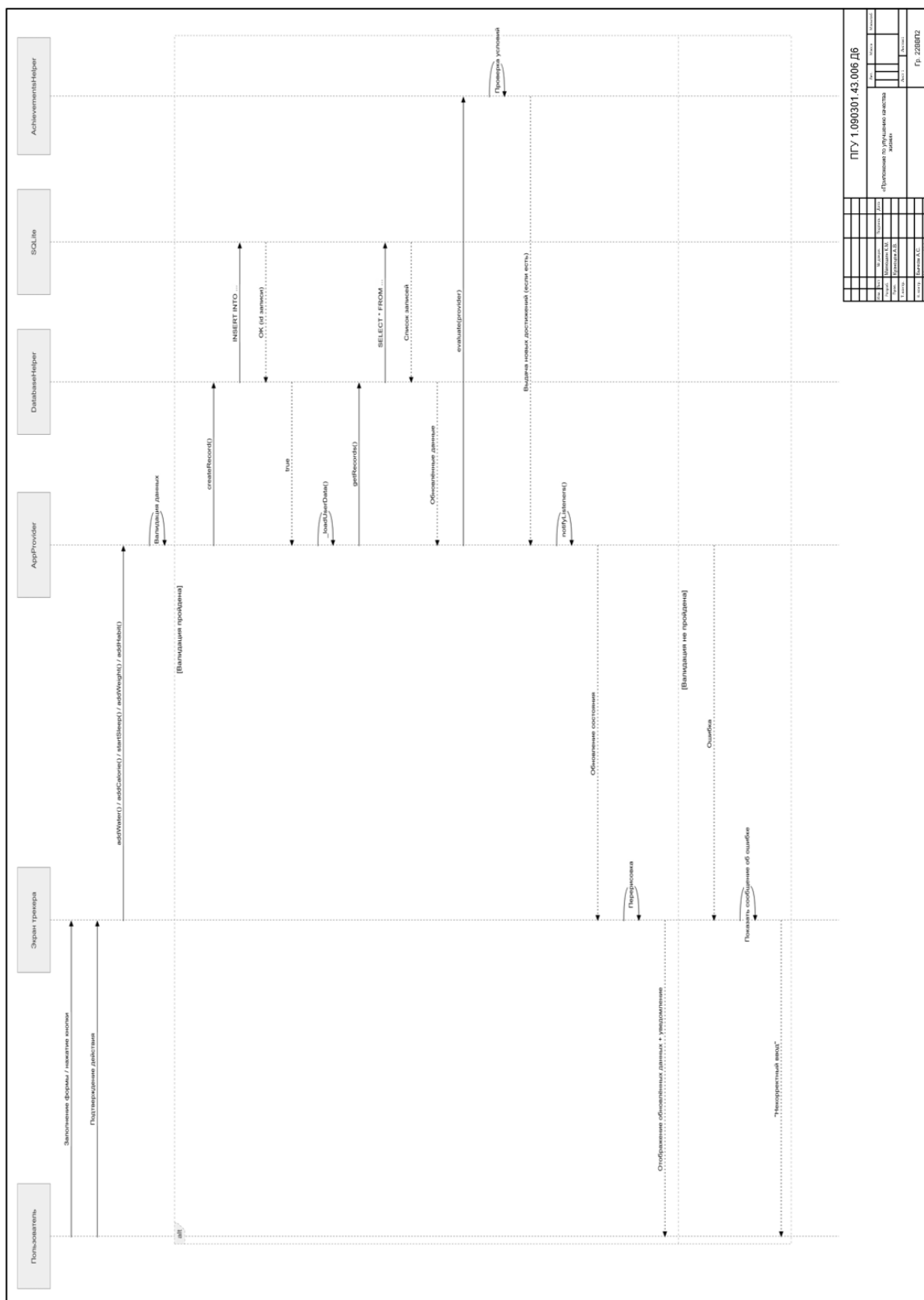
Приложение А.4 – ER-диаграмма базы данных



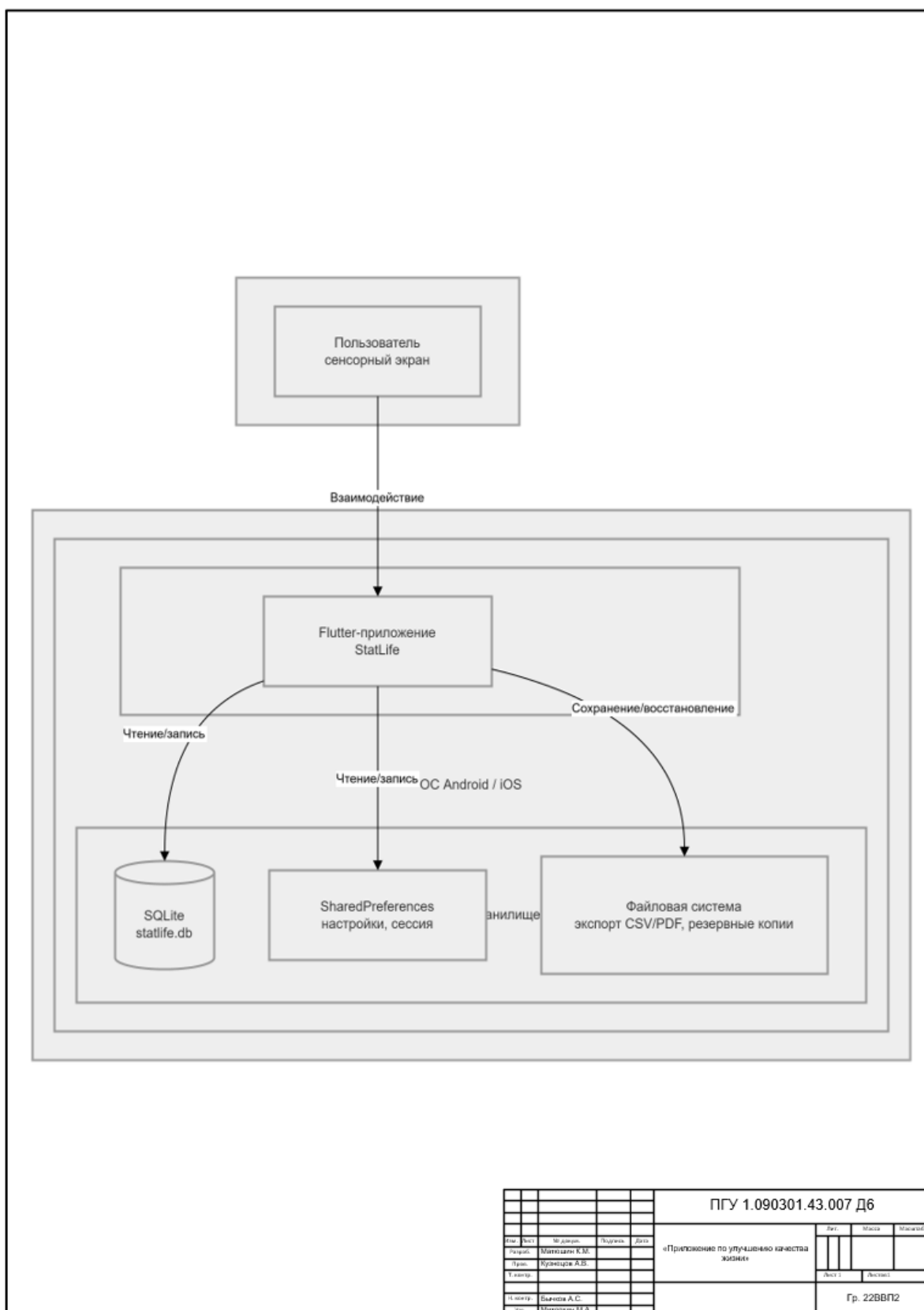
Приложение А.5 – Диаграмма последовательностей «Авторизация»



Приложение А.6 – Обобщённая диаграмма последовательности добавления записи



Приложение А.7 – Диаграмма развертывания приложения



Приложение Б

(обязательное)

«Приложение по улучшению качества жизни»

Листинг программ

					ПГУ1.090301.43.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		71

Файл lib/main.dart

```
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import
'package:flutter_localizations/flutter_localizations.dart';
import 'package:provider/provider.dart';
import 'package:intl/date_symbol_data_local.dart';
import 'providers/app_provider.dart';
import 'screens/login_screen.dart';
import 'screens/home_screen.dart';
import 'screens/sleep_screen.dart';
import 'screens/nutrition_screen.dart';
import 'screens/water_screen.dart';
import 'screens/habits_screen.dart';
import 'screens/achievements_screen.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await
SystemChrome.setPreferredOrientations([DeviceOrientation.portrai
tUp]);
  await initializeDateFormatting('ru_RU', null);

  runApp(
    MultiProvider(
      providers: [
        ChangeNotifierProvider(create: (_) => AppProvider()),
      ],
      child: const StatLifeApp(),
    ),
  );
}

class StatLifeApp extends StatelessWidget {
  const StatLifeApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Consumer<AppProvider>(
      builder: (context, provider, child) {
        return MaterialApp(
          title: 'StatLife',
          theme: ThemeData(
            useMaterial3: true,
            colorScheme: ColorScheme.fromSeed(seedColor:
Colors.teal),
          ),
          darkTheme: ThemeData.dark(useMaterial3: true).copyWith(
            colorScheme: ColorScheme.fromSeed(
              seedColor: Colors.teal,
              brightness: Brightness.dark,
            ),
          ),
        );
      },
    );
  }
}
```

					ПГУ1.090301.43.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		72

```

    ),
    themeMode: provider.themeMode,
    locale: provider.locale ?? const Locale('ru'),
    localizationsDelegates: const [
      GlobalMaterialLocalizations.delegate,
      GlobalWidgetsLocalizations.delegate,
      GlobalCupertinoLocalizations.delegate,
    ],
    supportedLocales: const [
      Locale('ru'),
      Locale('en'),
    ],
    home: Builder(
      builder: (context) {
        if (provider.isInitializing) {
          return const Scaffold(
            body: Center(child:
CircularProgressIndicator()),
          );
        }
        if (provider.currentUser == null) {
          return const LoginScreen();
        }
        return const MainNavigator();
      },
    ),
  );
},
);
}

class MainNavigator extends StatefulWidget {
  const MainNavigator({Key? key}) : super(key: key);

  @override
  State<MainNavigator> createState() => _MainNavigatorState();
}

class _MainNavigatorState extends State<MainNavigator> {
  int _currentIndex = 0;

  @override
  Widget build(BuildContext context) {
    final screens = [
      HomeScreen(onNavigate: (index) => setState(() =>
_currentIndex = index)),
      const SleepScreen(),
      const NutritionScreen(),
      const WaterScreen(),
      const HabitsScreen(),
      const AchievementsScreen(),
    ],
  },
);
}

```

```

];

return Scaffold(
  body: screens[_currentIndex],
  bottomNavigationBar: NavigationBar(
    labelBehavior:
NavigationDestinationLabelBehavior.alwaysHide,
    selectedIndex: _currentIndex,
    onDestinationSelected: (index) {
      setState(() {
        _currentIndex = index;
      });
    },
    destinations: const [
      NavigationDestination(
        icon: Icon(Icons.dashboard_outlined),
        selectedIcon: Icon(Icons.dashboard),
        label: 'Сводка',
      ),
      NavigationDestination(
        icon: Icon(Icons.bedtime_outlined),
        selectedIcon: Icon(Icons.bedtime),
        label: 'Сон',
      ),
      NavigationDestination(
        icon: Icon(Icons.restaurant_menu_outlined),
        selectedIcon: Icon(Icons.restaurant_menu),
        label: 'Питание',
      ),
      NavigationDestination(
        icon: Icon(Icons.water_drop_outlined),
        selectedIcon: Icon(Icons.water_drop),
        label: 'Вода',
      ),
      NavigationDestination(
        icon: Icon(Icons.track_changes_outlined),
        selectedIcon: Icon(Icons.track_changes),
        label: 'Привычки',
      ),
      NavigationDestination(
        icon: Icon(Icons.emoji_events_outlined),
        selectedIcon: Icon(Icons.emoji_events),
        label: 'Достижения',
      ),
    ],
  ),
);
}
}

```

Файл lib/providers/app_provider.dart

```

import 'dart:convert';
import 'package:flutter/material.dart';

```

					ПГУ1.090301.43.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		74

```

import 'package:crypto/crypto.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'package:file_picker/file_picker.dart';
import 'package:path_provider/path_provider.dart';
import 'dart:io';
import 'package:path/path.dart' as p;
import '../models/app_models.dart';
import '../helpers/database_helper.dart';
import '../helpers/achievements_helper.dart';
import '../helpers/achievements_definitions.dart';
import '../services/notification_service.dart';

class AppProvider with ChangeNotifier {
  User? _currentUser;
  UserSettings? _settings;
  bool _isInitializing = true;
  DateTime? _activeSleepStart;

  List<SleepRecord> _sleepRecords = [];
  List<CalorieIntake> _calorieIntakes = [];
  List<WaterIntake> _waterIntakes = [];
  List<Habit> _habits = [];
  Map<int, HabitProgress> _habitProgresses = {};
  List<Achievement> _earnedAchievements = [];

  User? get currentUser => _currentUser;
  UserSettings? get settings => _settings;
  bool get isInitializing => _isInitializing;
  DateTime? get activeSleepStart => _activeSleepStart;
  List<SleepRecord> get sleepRecords => _sleepRecords;
  List<CalorieIntake> get calorieIntakes => _calorieIntakes;
  List<WaterIntake> get waterIntakes => _waterIntakes;
  List<Habit> get habits => _habits;
  Map<int, HabitProgress> get habitProgresses => _habitProgresses;
  List<Achievement> get earnedAchievements =>
    _earnedAchievements;

  AppProvider() {
    checkLoginStatus();
  }

  Future<void> checkLoginStatus() async {
    final prefs = await SharedPreferences.getInstance();
    final isLoggedIn = prefs.getBool('isLoggedIn') ?? false;
    final userId = prefs.getInt('userId');

    if (isLoggedIn && userId != null) {
      final user = await
DatabaseHelper.instance.getUserById(userId);
      if (user != null) {
        _currentUser = user;
      }
    }
  }
}

```

```

        final sleepSavedStr =
prefs.getString('activeSleepStart');
        if (sleepSavedStr != null) {
            _activeSleepStart = DateTime.tryParse(sleepSavedStr);
        }
        await _loadUserData();
    }
}
_isInitializing = false;
notifyListeners();
}

String _hashPassword(String password) {
    var bytes = utf8.encode(password);
    var digest = sha256.convert(bytes);
    return digest.toString();
}

Future<bool> login(String username, String password) async {
    final user = await DatabaseHelper.instance.getUser(username);
    if (user != null && user.passwordHash ==
_hashPassword(password)) {
        _currentUser = user;
        final prefs = await SharedPreferences.getInstance();
        await prefs.setBool('isLoggedIn', true);
        await prefs.setInt('userId', user.id!);
        await _loadUserData();
        return true;
    }
    return false;
}

Future<bool> register(String username, String password, String
email) async {
    final existingUser = await
DatabaseHelper.instance.getUser(username);
    if (existingUser != null) return false;

    final newUser = User(
        username: username,
        email: email,
        passwordHash: _hashPassword(password),
    );
    final id = await DatabaseHelper.instance.createUser(newUser);

    final prefs = await SharedPreferences.getInstance();
    await prefs.setBool('isLoggedIn', true);
    await prefs.setInt('userId', id);

    _currentUser = User(id: id, username: username, passwordHash:
_hashPassword(password));
    await _loadUserData();
}

```

```

        return true;
    }

    Future<void> logout() async {
        final prefs = await SharedPreferences.getInstance();
        await prefs.clear();

        _currentUser = null;
        _settings = null;
        _sleepRecords = [];
        _calorieIntakes = [];
        _waterIntakes = [];
        _habits = [];
        _habitProgresses = {};
        _earnedAchievements = [];
        notifyListeners();
    }

    Future<void> _loadUserData() async {
        if (_currentUser == null || _currentUser!.id == null) return;
        final userId = _currentUser!.id!;

        _settings = await
        DatabaseHelper.instance.getUserSettings(userId);
        _sleepRecords = await
        DatabaseHelper.instance.getSleepRecords(userId);
        _calorieIntakes = await
        DatabaseHelper.instance.getCalorieIntakes(userId);
        _waterIntakes = await
        DatabaseHelper.instance.getWaterIntakes(userId);
        _habits = await DatabaseHelper.instance.getHabits(userId);

        _habitProgresses.clear();
        for (var habit in _habits) {
            if (habit.id != null) {
                final progress = await
                DatabaseHelper.instance.getHabitProgress(habit.id!);
                if (progress != null) {
                    _habitProgresses[habit.id!] = progress;
                }
            }
        }
        _recalculateHabitStreaks();
        _earnedAchievements = await
        DatabaseHelper.instance.getAchievements(userId);
        await AchievementsHelper.evaluate(this);
        notifyListeners();
    }

    Future<void> awardAchievement(int achievementId) async {
        if (_currentUser == null) return;

```

```

        if (_earnedAchievements.any((a) => a.achievementId ==
achievementId)) return;

        final achievement = Achievement(
            userId: _currentUser!.id!,
            achievementId: achievementId,
            dateEarned: DateTime.now().toIso8601String(),
        );
        await DatabaseHelper.instance.createAchievement(achievement);
        _earnedAchievements.add(achievement);
        notifyListeners();

        String title = 'Новое достижение!';
        try {
            final def = AchievementsDefinitions.all.firstWhere((d) =>
d.id == achievementId);
            title = def.title;
        } catch (_) {}

        NotificationService.instance.showNotification(
            1000 + achievementId,
            'Поздравляем!',
            'Вы получили достижение: $title'
        );
    }

    Future<void> startSleep() async {
        _activeSleepStart = DateTime.now();
        final prefs = await SharedPreferences.getInstance();
        await prefs.setString('activeSleepStart',
_activeSleepStart!.toIso8601String());
        notifyListeners();
    }

    Future<int> endSleep() async {
        if (_activeSleepStart == null) return 0;

        final wakeUp = DateTime.now();
        final actualSleepStart = _activeSleepStart!.add(const
Duration(minutes: 20));
        int duration = wakeUp.difference(actualSleepStart).inMinutes;
        if (duration < 0) duration = 0;

        await addSleepRecord(
            wakeUp.toIso8601String().substring(0, 10),
            actualSleepStart.toIso8601String(),
            wakeUp.toIso8601String(),
            duration,
        );

        _activeSleepStart = null;
        final prefs = await SharedPreferences.getInstance();

```

```

        await prefs.remove('activeSleepStart');
        notifyListeners();
        return duration;
    }

    Future<void> addSleepRecord(String date, String sleepStart,
String wakeUp, int durationMinutes) async {
        if (_currentUser == null) return;

        final record = SleepRecord(
            userId: _currentUser!.id!,
            date: date,
            sleepStart: sleepStart,
            wakeUp: wakeUp,
            durationMinutes: durationMinutes,
        );
        await DatabaseHelper.instance.createSleepRecord(record);
        await _loadUserData();
    }

    int get todayCalories {
        final now = DateTime.now();
        final today = DateTime(now.year, now.month, now.day);
        return _calorieIntakes
            .where((item)
                DateTime.parse(item.timestamp).isAfter(today.subtract(const
                    Duration(days: 1)))
                &&
                DateTime.parse(item.timestamp).isBefore(today.add(const
                    Duration(days: 1))))
            .fold(0, (sum, item) => sum + item.calories);
    }

    int get todayProtein {
        final todayStr
            DateTime.now().toIso8601String().substring(0, 10);
        return _calorieIntakes
            .where((i) => i.timestamp.startsWith(todayStr))
            .fold(0, (sum, item) => sum + item.protein);
    }

    int get todayFat {
        final todayStr
            DateTime.now().toIso8601String().substring(0, 10);
        return _calorieIntakes
            .where((i) => i.timestamp.startsWith(todayStr))
            .fold(0, (sum, item) => sum + item.fat);
    }

    int get todayCarbs {
        final todayStr
            DateTime.now().toIso8601String().substring(0, 10);

```

=>

=

=

=

```

        return _calorieIntakes
            .where((i) => i.timestamp.startsWith(todayStr))
            .fold(0, (sum, item) => sum + item.carbs);
    }

    Future<void> addCalorie(int calories, String description, String
mealType, {int protein = 0, int fat = 0, int carbs = 0}) async {
    if (_currentUser == null) return;
    if (calories < 0 || calories > 10000) return;

    final intake = CalorieIntake(
        userId: _currentUser!.id!,
        timestamp: DateTime.now().toIso8601String(),
        calories: calories,
        protein: protein,
        fat: fat,
        carbs: carbs,
        description: description,
        mealType: mealType,
    );
    await DatabaseHelper.instance.createCalorieIntake(intake);
    await _loadUserData();
}

Future<void> deleteCalorie(int id) async {
    await DatabaseHelper.instance.deleteCalorieIntake(id);
    await _loadUserData();
}

int get todayWater {
    final                                todayStr                                =
DateTime.now().toIso8601String().substring(0, 10);
    return _waterIntakes
        .where((w) => w.timestamp.startsWith(todayStr))
        .fold(0, (sum, item) => sum + item.volumeMl);
}

Future<void> addWater(int volumeMl) async {
    if (_currentUser == null) return;
    if (volumeMl <= 0 || volumeMl > 5000) return;

    final intake = WaterIntake(
        userId: _currentUser!.id!,
        timestamp: DateTime.now().toIso8601String(),
        volumeMl: volumeMl,
    );
    await DatabaseHelper.instance.createWaterIntake(intake);
    await _loadUserData();
}

Future<void> deleteWater(int id) async {
    await DatabaseHelper.instance.deleteWaterIntake(id);
}

```

```

        await _loadUserData();
    }

    Future<void> updateWaterGoal(int goal) async {
        if (_settings != null) {
            if (goal < 500) goal = 500;
            if (goal > 10000) goal = 10000;
            _settings = _settings!.copyWith(dailyWaterMl: goal);
            await
DatabaseHelper.instance.updateUserSettings(_settings!);
            notifyListeners();
        }
    }

    Future<void> updateCalorieGoal(int goal) async {
        if (_settings != null) {
            if (goal < 500) goal = 500;
            if (goal > 10000) goal = 10000;
            _settings = _settings!.copyWith(dailyCalories: goal);
            await
DatabaseHelper.instance.updateUserSettings(_settings!);
            notifyListeners();
        }
    }

    Future<void> updateSleepGoal(double goal) async {
        if (_settings != null) {
            if (goal <= 0) goal = 1.0;
            if (goal > 100.0) goal = 100.0;
            _settings = _settings!.copyWith(targetSleepHours: goal);
            await
DatabaseHelper.instance.updateUserSettings(_settings!);
            notifyListeners();
        }
    }

    Future<String?> backupDatabase() async {
        try {
            final          path          =          p.join((await
getApplicationDocumentsDirectory()).path, 'statlife.db');
            final dbFile = File(path);

            if (!await dbFile.exists()) return "Database file not
found.";

            String?          selectedDirectory          =          await
FilePicker.platform.getDirectoryPath();
            if (selectedDirectory == null) return "Backup cancelled.";

            final          backupPath          =          p.join(selectedDirectory,
'statlife_backup_${DateTime.now().millisecondsSinceEpoch}.db');
            await dbFile.copy(backupPath);

```

```

        return "Backup successful: $backupPath";
    } catch (e) {
        return "Backup failed: $e";
    }
}

Future<String?> restoreDatabase() async {
    try {
        FilePickerResult? result = await
FilePicker.platform.pickFiles(
        type: FileType.any,
    );

        if (result != null && result.files.single.path != null) {
            String restorePath = result.files.single.path!;

            final path = p.join((await
getApplicationDocumentsDirectory()).path, 'statlife.db');
            final dbFile = File(path);

            await DatabaseHelper.instance.close();

            final restoreFile = File(restorePath);
            await restoreFile.copy(path);

            await logout();

            return "Restore successful. Please log in again.";
        } else {
            return "Restore cancelled.";
        }
    } catch (e) {
        return "Restore failed: $e";
    }
}

ThemeMode get themeMode {
    if (_settings == null) return ThemeMode.system;
    switch (_settings!.themeMode) {
        case 'light': return ThemeMode.light;
        case 'dark': return ThemeMode.dark;
        default: return ThemeMode.system;
    }
}

Locale? get locale {
    if (_settings == null) return null;
    return Locale(_settings!.language);
}

Future<void> updateThemeMode(String mode) async {
    if (_settings != null) {

```

```

        _settings = _settings!.copyWith(themeMode: mode);
        await
        DatabaseHelper.instance.updateUserSettings(_settings!);
        notifyListeners();
    }
}

Future<void> updateLanguage(String lang) async {
    if (_settings != null) {
        _settings = _settings!.copyWith(language: lang);
        await
        DatabaseHelper.instance.updateUserSettings(_settings!);
        notifyListeners();
    }
}

Future<void> updateEnableNotifications(bool enable) async {
    if (_settings != null) {
        _settings = _settings!.copyWith(enableNotifications:
enable);
        await
        DatabaseHelper.instance.updateUserSettings(_settings!);

        if (enable) {
            await NotificationService.instance.requestPermissions();
            await
            NotificationService.instance.scheduleDailyNotification(
                101,
                'Напоминание о воде',
                'Не забудьте выпить стакан воды!',
                const TimeOfDay(hour: 10, minute: 0)
            );
            await
            NotificationService.instance.scheduleDailyNotification(
                102,
                'Напоминание о питании',
                'Запишите ваш обед.',
                const TimeOfDay(hour: 14, minute: 0)
            );
        } else {
            await
            NotificationService.instance.cancelAllNotifications();
        }

        notifyListeners();
    }
}

Future<void> addHabit(String name, String category, DateTime
startDate, {int? targetDays}) async {
    if (_currentUser == null) return;

```

```

final habit = Habit(
    userId: _currentUser!.id!,
    name: name,
    category: category,
    startDate: startDate.toIso8601String(),
    targetDays: targetDays,
);
await DatabaseHelper.instance.createHabit(habit);
await _loadUserData();
}

Future<void> resetHabit(int habitId) async {
    if (!_habitProgresses.containsKey(habitId)) return;
    final prog = _habitProgresses[habitId]!;

    final newProg = HabitProgress(
        habitId: prog.habitId,
        currentStreak: 0,
        recordStreak: prog.recordStreak,
        lastResetDate: DateTime.now().toIso8601String(),
        totalTrackingDays: prog.totalTrackingDays,
    );

    await DatabaseHelper.instance.updateHabitProgress(newProg);
    await _loadUserData();
}

Future<void> deleteHabit(int habitId) async {
    await DatabaseHelper.instance.deleteHabit(habitId);
    _habits.removeWhere((h) => h.id == habitId);
    _habitProgresses.remove(habitId);
    notifyListeners();
}

void _recalculateHabitStreaks() {
    final now = DateTime.now();
    for (var habit in _habits) {
        if (habit.id == null) continue;
        final prog = _habitProgresses[habit.id]!;

        final startDate = DateTime.parse(habit.startDate);
        final resetDateStr = prog.lastResetDate;
        final referenceDate = resetDateStr != null ?
DateTime.parse(resetDateStr) : startDate;

        final refDay = DateTime(referenceDate.year,
referenceDate.month, referenceDate.day);
        final nowDay = DateTime(now.year, now.month, now.day);

        int currentStreak = nowDay.difference(refDay).inDays;
        if (currentStreak < 0) currentStreak = 0;
    }
}

```

```

        int recordStreak = prog.recordStreak;
        if (currentStreak > recordStreak) {
            recordStreak = currentStreak;
        }

        if (currentStreak != prog.currentStreak || recordStreak !=
prog.recordStreak) {
            final newProg = HabitProgress(
                habitId: prog.habitId,
                currentStreak: currentStreak,
                recordStreak: recordStreak,
                lastResetDate: prog.lastResetDate,
                totalTrackingDays: prog.totalTrackingDays,
            );
            DatabaseHelper.instance.updateHabitProgress(newProg);
            _habitProgresses[habit.id!] = newProg;
        }
    }
}

```

Файл lib/helpers/database_helper.dart

```

import 'package:sqflite/sqflite.dart';
import 'package:path/path.dart';
import 'package:path_provider/path_provider.dart';
import '../models/app_models.dart';

class DatabaseHelper {
    static final DatabaseHelper instance =
DatabaseHelper._internal();
    factory DatabaseHelper() => instance;
    static Database? _database;

    DatabaseHelper._internal();

    Future<Database> get database async {
        if (_database != null) return _database!;
        _database = await _initDB('statlife.db');
        return _database!;
    }

    Future<Database> _initDB(String filePath) async {
        final dbPath =
            (await
getApplicationDocumentsDirectory()).path;
        final path = join(dbPath, filePath);

        return await openDatabase(
            path,
            version: 3,
            onConfigure: _onConfigure,
            onCreate: _createDB,
            onUpgrade: _onUpgrade,

```

```

    );
}

Future _onConfigure(Database db) async {
    await db.execute('PRAGMA foreign_keys = ON');
}

Future _createDB(Database db, int version) async {
    const idType = 'INTEGER PRIMARY KEY AUTOINCREMENT';
    const intType = 'INTEGER NOT NULL';
    const textType = 'TEXT NOT NULL';
    const realType = 'REAL NOT NULL';

    await db.execute('''
CREATE TABLE users (
    id $idType,
    username TEXT UNIQUE NOT NULL,
    email TEXT,
    password_hash $textType
)
''');

    await db.execute('''
CREATE TABLE user_settings (
    user_id INTEGER PRIMARY KEY,
    daily_calories INTEGER DEFAULT 2500,
    daily_water_ml INTEGER DEFAULT 2500,
    target_sleep_hours REAL DEFAULT 8.0,
    theme_mode TEXT DEFAULT 'system',
    language TEXT DEFAULT 'ru',
    enable_notifications INTEGER DEFAULT 0,
    FOREIGN KEY (user_id) REFERENCES users (id) ON DELETE CASCADE
)
''');

    await db.execute('''
CREATE TABLE sleep_records (
    id $idType,
    user_id $intType,
    date $textType,
    sleep_start $textType,
    wake_up $textType,
    duration_minutes $intType,
    FOREIGN KEY (user_id) REFERENCES users (id) ON DELETE CASCADE
)
''');

    await db.execute('''
CREATE TABLE calorie_intake (
    id $idType,
    user_id $intType,
    timestamp $textType,

```

```

        calories $intType,
        protein INTEGER DEFAULT 0,
        fat INTEGER DEFAULT 0,
        carbs INTEGER DEFAULT 0,
        description $textType,
        meal_type $textType,
        FOREIGN KEY (user_id) REFERENCES users (id) ON DELETE CASCADE
    )
    '');

    await db.execute('''
CREATE TABLE water_intake (
    id $idType,
    user_id $intType,
    timestamp $textType,
    volume_ml $intType,
    FOREIGN KEY (user_id) REFERENCES users (id) ON DELETE CASCADE
)
    '');

    await db.execute('''
CREATE TABLE habits (
    id $idType,
    user_id $intType,
    name $textType,
    category $textType,
    start_date $textType,
    target_days INTEGER,
    FOREIGN KEY (user_id) REFERENCES users (id) ON DELETE CASCADE
)
    '');

    await db.execute('''
CREATE TABLE habit_progress (
    habit_id INTEGER PRIMARY KEY,
    current_streak INTEGER DEFAULT 0,
    record_streak INTEGER DEFAULT 0,
    last_reset_date TEXT,
    total_tracking_days INTEGER DEFAULT 0,
    FOREIGN KEY (habit_id) REFERENCES habits (id) ON DELETE CASCADE
)
    '');

    await db.execute('''
CREATE TABLE weight_records (
    id $idType,
    user_id $intType,
    date $textType,
    weight $realType,
    FOREIGN KEY (user_id) REFERENCES users (id) ON DELETE CASCADE
)
    '');

```

```

        await db.execute('''
CREATE TABLE achievements (
    id $idType,
    user_id $intType,
    achievement_id $intType,
    date_earned $textType,
    FOREIGN KEY (user_id) REFERENCES users (id) ON DELETE CASCADE
)
''');
    }

    Future _onUpgrade(Database db, int oldVersion, int newVersion)
    async {
        if (oldVersion < 3) {
            try {
                await db.execute('ALTER TABLE calorie_intake ADD COLUMN
protein INTEGER DEFAULT 0');
                await db.execute('ALTER TABLE calorie_intake ADD COLUMN
fat INTEGER DEFAULT 0');
                await db.execute('ALTER TABLE calorie_intake ADD COLUMN
carbs INTEGER DEFAULT 0');
            } catch (e) {}
        }
    }

    Future<User?> getUser(String username) async {
        final db = await instance.database;
        final maps = await db.query(
            'users',
            where: 'username = ?',
            whereArgs: [username],
        );
        if (maps.isNotEmpty) {
            return User.fromMap(maps.first);
        }
        return null;
    }

    Future<User?> getUserById(int id) async {
        final db = await instance.database;
        final maps = await db.query(
            'users',
            where: 'id = ?',
            whereArgs: [id],
        );
        if (maps.isNotEmpty) {
            return User.fromMap(maps.first);
        }
        return null;
    }
}

```

```

Future<int> createUser(User user) async {
    final db = await instance.database;
    final id = await db.insert('users', user.toMap());
    await db.insert('user_settings', UserSettings(userId:
id).toMap());
    return id;
}

Future<UserSettings?> getUserSettings(int userId) async {
    final db = await instance.database;
    final maps = await db.query(
        'user_settings',
        where: 'user_id = ?',
        whereArgs: [userId],
    );
    if (maps.isNotEmpty) {
        return UserSettings.fromMap(maps.first);
    }
    return null;
}

Future<int> updateUserSettings(UserSettings settings) async {
    final db = await instance.database;
    return db.update(
        'user_settings',
        settings.toMap(),
        where: 'user_id = ?',
        whereArgs: [settings.userId],
    );
}

Future<int> createSleepRecord(SleepRecord record) async {
    final db = await instance.database;
    return await db.insert('sleep_records', record.toMap());
}

Future<List<SleepRecord>> getSleepRecords(int userId) async {
    final db = await instance.database;
    final result = await db.query(
        'sleep_records',
        where: 'user_id = ?',
        whereArgs: [userId],
        orderBy: 'date DESC',
    );
    return result.map((json) =>
SleepRecord.fromMap(json)).toList();
}

Future<int> createCalorieIntake(CalorieIntake intake) async {
    final db = await instance.database;
    return await db.insert('calorie_intake', intake.toMap());
}

```

```

Future<List<CalorieIntake>> getCalorieIntakes(int userId) async
{
    final db = await instance.database;
    final result = await db.query(
        'calorie_intake',
        where: 'user_id = ?',
        whereArgs: [userId],
        orderBy: 'timestamp DESC',
    );
    return result.map((json) =>
CalorieIntake.fromMap(json)).toList();
}

Future<int> createWaterIntake(WaterIntake intake) async {
    final db = await instance.database;
    return await db.insert('water_intake', intake.toMap());
}

Future<List<WaterIntake>> getWaterIntakes(int userId) async {
    final db = await instance.database;
    final result = await db.query(
        'water_intake',
        where: 'user_id = ?',
        whereArgs: [userId],
        orderBy: 'timestamp DESC',
    );
    return result.map((json) =>
WaterIntake.fromMap(json)).toList();
}

Future<int> deleteWaterIntake(int id) async {
    final db = await instance.database;
    return await db.delete(
        'water_intake',
        where: 'id = ?',
        whereArgs: [id],
    );
}

Future<int> deleteCalorieIntake(int id) async {
    final db = await instance.database;
    return await db.delete(
        'calorie_intake',
        where: 'id = ?',
        whereArgs: [id],
    );
}

Future<int> createHabit(Habit habit) async {
    final db = await instance.database;
    final id = await db.insert('habits', habit.toMap());
}

```

```

        await db.insert('habit_progress', HabitProgress(habitId:
id).toMap());
        return id;
    }

Future<List<Habit>> getHabits(int userId) async {
    final db = await instance.database;
    final result = await db.query(
        'habits',
        where: 'user_id = ?',
        whereArgs: [userId],
    );
    return result.map((json) => Habit.fromMap(json)).toList();
}

Future<HabitProgress?> getHabitProgress(int habitId) async {
    final db = await instance.database;
    final result = await db.query(
        'habit_progress',
        where: 'habit_id = ?',
        whereArgs: [habitId],
    );
    if (result.isNotEmpty) {
        return HabitProgress.fromMap(result.first);
    }
    return null;
}

Future<int> updateHabitProgress(HabitProgress progress) async {
    final db = await instance.database;
    return await db.update(
        'habit_progress',
        progress.toMap(),
        where: 'habit_id = ?',
        whereArgs: [progress.habitId],
    );
}

Future<int> deleteHabit(int habitId) async {
    final db = await instance.database;
    return await db.delete(
        'habits',
        where: 'id = ?',
        whereArgs: [habitId],
    );
}

Future close() async {
    final db = await instance.database;
    db.close();
}

```

```

Future<int> createWeightRecord(WeightRecord record) async {
  final db = await instance.database;
  return await db.insert('weight_records', record.toMap());
}

Future<List<WeightRecord>> getWeightRecords(int userId) async {
  final db = await instance.database;
  final result = await db.query(
    'weight_records',
    where: 'user_id = ?',
    whereArgs: [userId],
    orderBy: 'date DESC',
  );
  return result.map((json) =>
WeightRecord.fromMap(json)).toList();
}

Future<int> createAchievement(Achievement record) async {
  final db = await instance.database;
  return await db.insert('achievements', record.toMap());
}

Future<List<Achievement>> getAchievements(int userId) async {
  final db = await instance.database;
  final result = await db.query(
    'achievements',
    where: 'user_id = ?',
    whereArgs: [userId],
    orderBy: 'date_earned DESC',
  );
  return result.map((json) =>
Achievement.fromMap(json)).toList();
}

Future<List<Map<String, dynamic>>> getAllUsers() async {
  final db = await instance.database;
  return await db.query('users');
}

Future<List<Map<String, dynamic>>> exportTable(String
tableName) async {
  final db = await instance.database;
  return await db.query(tableName);
}
}

```

Файл lib/helpers/achievements_helper.dart

```

import '../providers/app_provider.dart';
import 'database_helper.dart';

class AchievementsHelper {
  static Future<void> evaluate(AppProvider provider) async {

```

```

        final userId = provider.currentUser?.id;
        if (userId == null) return;

        final existing = await
DatabaseHelper.instance.getAchievements(userId);
        final earnedIds = existing.map((e) =>
e.achievementId).toSet();

        Future<void> check(int id, bool condition) async {
            if (condition && !earnedIds.contains(id)) {
                await provider.awardAchievement(id);
                earnedIds.add(id);
            }
        }

        final sleep = provider.sleepRecords;
        final water = provider.waterIntakes;
        final meals = provider.calorieIntakes;
        final habits = provider.habits;
        final habitProgs = provider.habitProgresses;

        // --- Sleep Checks (1-15) ---
        if (sleep.isNotEmpty) {
            await check(1, true);

            bool before6 = sleep.any((s) {
                final time = DateTime.parse(s.wakeUp);
                return time.hour < 6;
            });
            await check(2, before6);

            bool after1 = sleep.any((s) {
                final time = DateTime.parse(s.sleepStart);
                return time.hour >= 1 && time.hour < 5;
            });
            await check(3, after1);

            await check(4, sleep.any((s) => s.durationMinutes >= 475 &&
s.durationMinutes <= 485));
            await check(5, sleep.any((s) => s.durationMinutes > 600));
            await check(6, sleep.any((s) => s.durationMinutes < 300));

            int normalSleepCount = sleep.where((s) => s.durationMinutes
>= 420 && s.durationMinutes <= 510).length;
            await check(7, normalSleepCount >= 7);
            await check(8, normalSleepCount >= 30);
            await check(9, normalSleepCount >= 100);

            await check(11, sleep.where((s)
DateTime.parse(s.wakeUp).hour < 7).length >= 7);
            await check(14, sleep.length >= 30);
            await check(15, sleep.length >= 100);

```

```

}

// --- Water Checks (16-30) ---
if (water.isNotEmpty) {
    await check(16, true);
    await check(17, water.any((w) => w.volumeMl == 250));

    final todayWater = provider.todayWater;
    final waterGoal = provider.settings?.dailyWaterMl ?? 2500;
    await check(18, todayWater >= waterGoal);

    await check(21, todayWater >= 5000);
    await check(22, todayWater > 0 && todayWater < 1000);

    int totalWater = water.fold(0, (sum, item) => sum +
item.volumeMl);
    await check(23, totalWater >= 100000);
    await check(24, totalWater >= 1000000);

    await check(25, water.length >= 50);
    await check(26, water.length >= 500);

    int morningCups = water.where((w) =>
DateTime.parse(w.timestamp).hour < 9 && w.volumeMl >= 250).length;
    await check(27, morningCups >= 7);

    int eveningCups = water.where((w) =>
DateTime.parse(w.timestamp).hour >= 21 && w.volumeMl >=
250).length;
    await check(28, eveningCups >= 7);

    await check(29, (todayWater - waterGoal).abs() <= 50);
    await check(30, todayWater >= 1.5 * waterGoal);
}

// --- Nutrition Checks (31-45) ---
if (meals.isNotEmpty) {
    await check(31, true);
    await check(32, meals.any((m) => m.mealType == 'завтрак'));
    await check(33, meals.any((m) => m.mealType == 'обед'));
    await check(34, meals.any((m) => m.mealType == 'ужин'));

    int snacks = meals.where((m) => m.mealType ==
'перекус').length;
    await check(35, snacks >= 10);

    final todayCal = provider.todayCalories;
    final calGoal = provider.settings?.dailyCalories ?? 2500;
    await check(38, (todayCal - calGoal).abs() <= 100);
    await check(40, (todayCal - calGoal).abs() <= 10);

    await check(41, meals.length >= 100);
}

```

```

        await check(42, meals.length >= 500);

        if (todayCal > 0) {
            await check(43, todayCal < 1000);
            await check(44, todayCal > 4000);
        }
    }

    // --- Habit Checks (46-60) ---
    if (habits.isNotEmpty) {
        await check(46, true);
        await check(53, habitProgs.values.where((p) =>
p.currentStreak > 0).length >= 3);
        await check(54, habitProgs.values.where((p) =>
p.currentStreak > 0).length >= 5);
        await check(55, habitProgs.values.any((p) => p.recordStreak
>= 100));

        for (var h in habits) {
            final p = habitProgs[h.id];
            if (p == null) continue;

            if (h.name.toLowerCase().contains('курен')) await
check(56, p.currentStreak >= 30);
            if (h.name.toLowerCase().contains('алкогол')) await
check(57, p.currentStreak >= 30);
            if (h.name.toLowerCase().contains('соцсет')) await
check(58, p.currentStreak >= 30);

            if (h.targetDays != null && p.currentStreak >=
h.targetDays!) {
                await check(59, true);
            }
        }
        await check(60, habits.length >= 10);
    }

    // --- Combo Checks (61-70) ---
    final todayWater = provider.todayWater;
    final waterGoal = provider.settings?.dailyWaterMl ?? 2500;
    final todayCal = provider.todayCalories;
    final calGoal = provider.settings?.dailyCalories ?? 2500;
    final hasSleepToday = sleep.any((s) => s.date ==
DateTime.now().toIso8601String().substring(0, 10));

    bool healthyDay = todayWater >= waterGoal && (todayCal -
calGoal).abs() <= 200 && hasSleepToday;
    await check(61, healthyDay);
    await check(63, healthyDay);

    // --- Milestones (71-85) ---
    int totalRecords = sleep.length + water.length + meals.length;

```

```

await check(71, totalRecords >= 10);
await check(72, totalRecords >= 100);
await check(73, totalRecords >= 500);
await check(74, totalRecords >= 1000);

if (sleep.isNotEmpty) {
  final first = DateTime.parse(sleep.last.date);
  final diff = DateTime.now().difference(first).inDays;
  await check(75, diff >= 7);
  await check(76, diff >= 30);
  await check(77, diff >= 100);
  await check(78, diff >= 183);
  await check(79, diff >= 365);
}

// --- Special (86-100) ---
if (provider.settings != null) {
  await check(86, provider.settings!.dailyWaterMl != 2500 ||
provider.settings!.dailyCalories != 2500);
  await check(89, provider.settings!.themeMode == 'dark');
  await check(90, provider.settings!.enableNotifications);
}

int collected = earnedIds.length;
await check(94, collected >= 25);
await check(95, collected >= 50);
await check(96, collected >= 75);
if (collected >= 99) await check(97, true);
await check(98, collected >= 1);

if (earnedIds.length >= 99) {
  await check(85, true);
}

}
}

```

Файл **achievements_definitions.dart**

```
import 'package:flutter/material.dart';
```

```

class AchievementDefinition {
  final int id;
  final String category;
  final String title;
  final String description;
  final IconData iconData;
  final Color iconColor;

  const AchievementDefinition({
    required this.id,
    required this.category,
    required this.title,

```

					ПГУ1.090301.43.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		96

```

        required this.description,
        required this.iconData,
        required this.iconColor,
    });
}

class AchievementsDefinitions {
    static const List<AchievementDefinition> all = [
        // Sleep (1-15)
        AchievementDefinition(id: 1, category: 'sleep', title: 'Первый сон', description: 'Добавить первую запись сна.', iconData: Icons.bedtime, iconColor: Colors.indigo),
        AchievementDefinition(id: 2, category: 'sleep', title: 'Ранняя пташка', description: 'Проснуться до 6:00 хотя бы раз.', iconData: Icons.wb_sunny, iconColor: Colors.orange),
        AchievementDefinition(id: 3, category: 'sleep', title: 'Сова', description: 'Лечь спать после 1:00 хотя бы раз.', iconData: Icons.nightlight, iconColor: Colors.indigoAccent),
        AchievementDefinition(id: 4, category: 'sleep', title: 'Золотая середина', description: 'Проспать ровно 8 часов.', iconData: Icons.star, iconColor: Colors.yellow),
        AchievementDefinition(id: 5, category: 'sleep', title: 'Богатырский сон', description: 'Проспать более 10 часов.', iconData: Icons.shield, iconColor: Colors.blueGrey),
        AchievementDefinition(id: 6, category: 'sleep', title: 'Короткий сон', description: 'Проспать менее 5 часов.', iconData: Icons.flash_on, iconColor: Colors.amber),
        AchievementDefinition(id: 7, category: 'sleep', title: '7 дней нормы', description: 'Спать 7-8 часов 7 дней подряд.', iconData: Icons.looks_one, iconColor: Colors.green),
        AchievementDefinition(id: 8, category: 'sleep', title: '30 дней нормы', description: 'Спать 7-8 часов 30 дней подряд.', iconData: Icons.calendar_month, iconColor: Colors.green),
        AchievementDefinition(id: 9, category: 'sleep', title: '100 дней нормы', description: 'Спать 7-8 часов 100 дней подряд.', iconData: Icons.emoji_events, iconColor: Colors.amberAccent),
        AchievementDefinition(id: 10, category: 'sleep', title: 'Идеальный месяц', description: 'Все дни месяца спать 7-8 часов.', iconData: Icons.diamond, iconColor: Colors.cyan),
        AchievementDefinition(id: 11, category: 'sleep', title: 'Ранний подъёмник', description: 'Просыпаться до 7:00 7 дней подряд.', iconData: Icons.alarm, iconColor: Colors.orangeAccent),
        AchievementDefinition(id: 12, category: 'sleep', title: 'Ночной страж', description: 'Не спать более 24 часов.', iconData: Icons.visibility, iconColor: Colors.redAccent),
        AchievementDefinition(id: 13, category: 'sleep', title: 'Без будильника', description: 'Заснуть без установки будильника 5 раз.', iconData: Icons.alarm_off, iconColor: Colors.grey),
        AchievementDefinition(id: 14, category: 'sleep', title: 'Хронометрист', description: 'Заполнить 30 записей сна.', iconData: Icons.timelapse, iconColor: Colors.blue),
    ];
}

```

```

AchievementDefinition(id: 15, category: 'sleep', title:
'Сонный эксперт', description: 'Заполнить 100 записей сна.',
iconData: Icons.hotel, iconColor: Colors.deepPurple),

// Water (16-30)
AchievementDefinition(id: 16, category: 'water', title:
'Первый глоток', description: 'Добавить первую запись воды.',
iconData: Icons.water_drop, iconColor: Colors.lightBlue),
AchievementDefinition(id: 17, category: 'water', title:
'Стакан за стаканом', description: 'Выпить 250 мл за раз.',
iconData: Icons.local_drink, iconColor: Colors.lightBlueAccent),
AchievementDefinition(id: 18, category: 'water', title:
'Дневная норма', description: 'Выполнить норму воды за день.',
iconData: Icons.check_circle, iconColor: Colors.blue),
AchievementDefinition(id: 19, category: 'water', title:
'Неделя гидратации', description: 'Выполнять норму воды 7 дней
подряд.', iconData: Icons.looks_one, iconColor:
Colors.blueAccent),
AchievementDefinition(id: 20, category: 'water', title: 'Месяц
гидратации', description: 'Выполнять норму воды 30 дней подряд.',
iconData: Icons.calendar_month, iconColor: Colors.blueAccent),
AchievementDefinition(id: 21, category: 'water', title:
'Водохлёб', description: 'Выпить 5 литров за один день.',
iconData: Icons.pool, iconColor: Colors.cyan),
AchievementDefinition(id: 22, category: 'water', title:
'Экономный', description: 'Выпить менее 1 литра за день.',
iconData: Icons.eco, iconColor: Colors.green),
AchievementDefinition(id: 23, category: 'water', title: '100
литров', description: 'Суммарно выпить 100 литров воды.',
iconData: Icons.waves, iconColor: Colors.blue),
AchievementDefinition(id: 24, category: 'water', title: '1000
литров', description: 'Суммарно выпить 1000 литров воды.',
iconData: Icons.tsunami, iconColor: Colors.indigo),
AchievementDefinition(id: 25, category: 'water', title: 'Следи
за собой', description: 'Добавить 50 записей воды.', iconData:
Icons.analytics, iconColor: Colors.grey),
AchievementDefinition(id: 26, category: 'water', title:
'Гидроман', description: 'Добавить 500 записей воды.', iconData:
Icons.timeline, iconColor: Colors.deepPurple),
AchievementDefinition(id: 27, category: 'water', title: 'Утро
начинается с воды', description: 'Выпить стакан воды до 9:00 7
раз.', iconData: Icons.wb_twilight, iconColor:
Colors.orangeAccent),
AchievementDefinition(id: 28, category: 'water', title:
'Вечерний глоток', description: 'Выпить стакан воды после 21:00 7
раз.', iconData: Icons.nightlight_round, iconColor:
Colors.indigo),
AchievementDefinition(id: 29, category: 'water', title:
'Идеальный день', description: 'Выпить ровно норму.', iconData:
Icons.done_all, iconColor: Colors.green),

```

```

    AchievementDefinition(id: 30, category: 'water', title:
'Sверхнорма', description: 'Выпить 150% от нормы за день.',
iconData: Icons.arrow_upward, iconColor: Colors.purple),

    // Nutrition (31-45)
    AchievementDefinition(id: 31, category: 'nutrition', title:
'Первый приём', description: 'Добавить первый приём пищи.',
iconData: Icons.restaurant, iconColor: Colors.orange),
    AchievementDefinition(id: 32, category: 'nutrition', title:
'Завтрак героя', description: 'Записать завтрак.', iconData:
Icons.egg, iconColor: Colors.orangeAccent),
    AchievementDefinition(id: 33, category: 'nutrition', title:
'Обед чемпиона', description: 'Записать обед.', iconData:
Icons.lunch_dining, iconColor: Colors.yellow),
    AchievementDefinition(id: 34, category: 'nutrition', title:
'Ужин философа', description: 'Записать ужин.', iconData:
Icons.dinner_dining, iconColor: Colors.deepOrange),
    AchievementDefinition(id: 35, category: 'nutrition', title:
'Любитель перекусов', description: 'Записать 10 перекусов.',
iconData: Icons.fastfood, iconColor: Colors.red),
    AchievementDefinition(id: 36, category: 'nutrition', title:
'Легко', description: 'Уложиться в норму калорий 7 дней подряд.',
iconData: Icons.fitness_center, iconColor: Colors.green),
    AchievementDefinition(id: 37, category: 'nutrition', title:
'Сбалансированно', description: 'Уложиться в норму 30 дней
подряд.', iconData: Icons.scale, iconColor: Colors.green),
    AchievementDefinition(id: 38, category: 'nutrition', title:
'Золотая середина', description: 'День в жёлтой зоне.', iconData:
Icons.trending_flat, iconColor: Colors.yellow),
    AchievementDefinition(id: 39, category: 'nutrition', title:
'Экономия', description: 'Неделя в зелёной зоне.', iconData:
Icons.trending_down, iconColor: Colors.green),
    AchievementDefinition(id: 40, category: 'nutrition', title:
'Честный день', description: 'Точное попадание в норму.',
iconData: Icons.gps_fixed, iconColor: Colors.red),
    AchievementDefinition(id: 41, category: 'nutrition', title:
'Фуд-блогер', description: 'Добавить 100 приёмов пищи.', iconData:
Icons.camera_alt, iconColor: Colors.pink),
    AchievementDefinition(id: 42, category: 'nutrition', title:
'Гастроном', description: 'Добавить 500 приёмов пищи.', iconData:
Icons.star_border, iconColor: Colors.amber),
    AchievementDefinition(id: 43, category: 'nutrition', title:
'Разгрузка', description: 'День с калориями менее 1000.',
iconData: Icons.air, iconColor: Colors.lightBlue),
    AchievementDefinition(id: 44, category: 'nutrition', title:
'Пирушка', description: 'День с калориями более 4000.', iconData:
Icons.celebration, iconColor: Colors.redAccent),
    AchievementDefinition(id: 45, category: 'nutrition', title:
'Контроль', description: '7 дней подряд все приёмы описаны.',
iconData: Icons.edit_note, iconColor: Colors.blueGrey),

    // Habits (46-60)

```

```

    AchievementDefinition(id: 46, category: 'habit', title:
'Первый шаг', description: 'Добавить первую привычку.', iconData:
Icons.directions_walk, iconColor: Colors.green),
    AchievementDefinition(id: 47, category: 'habit', title:
'Бросающий', description: 'Добавить привычку с датой отказа.',
iconData: Icons.block, iconColor: Colors.red),
    AchievementDefinition(id: 48, category: 'habit', title:
'Неделя чистоты', description: '7 дней без срыва у любой
привычки.', iconData: Icons.cleaning_services, iconColor:
Colors.teal),
    AchievementDefinition(id: 49, category: 'habit', title: 'Месяц
силы воли', description: '30 дней без срыва у любой привычки.',
iconData: Icons.shield, iconColor: Colors.blue),
    AchievementDefinition(id: 50, category: 'habit', title: 'Год
свободы', description: '365 дней без срыва.', iconData:
Icons.flight_takeoff, iconColor: Colors.purple),
    AchievementDefinition(id: 51, category: 'habit', title: 'Срыв
- не конец', description: 'Сбросить серию.', iconData:
Icons.refresh, iconColor: Colors.orange),
    AchievementDefinition(id: 52, category: 'habit', title:
'Феникс', description: 'Восстановить серию после срыва.',
iconData: Icons.local_fire_department, iconColor:
Colors.deepOrange),
    AchievementDefinition(id: 53, category: 'habit', title:
'Многопрофильный', description: 'Иметь 3 активные привычки.',
iconData: Icons.group_work, iconColor: Colors.blueGrey),
    AchievementDefinition(id: 54, category: 'habit', title:
'Борец', description: 'Иметь 5 активных привычек.', iconData:
Icons.sports_martial_arts, iconColor: Colors.indigo),
    AchievementDefinition(id: 55, category: 'habit', title:
'Серийный рекордсмен', description: 'Установить рекорд серии > 100
дней.', iconData: Icons.military_tech, iconColor: Colors.amber),
    AchievementDefinition(id: 56, category: 'habit', title:
'Никотиновый ноль', description: 'Привычка "Курение" 30 дней.',
iconData: Icons.smoke_free, iconColor: Colors.grey),
    AchievementDefinition(id: 57, category: 'habit', title:
'Трезвость', description: 'Привычка "Алкоголь" 30 дней.',
iconData: Icons.water_drop_outlined, iconColor: Colors.blue),
    AchievementDefinition(id: 58, category: 'habit', title:
'Цифровой детокс', description: 'Привычка "Соцсети" 30 дней.',
iconData: Icons.phonelink_erase, iconColor: Colors.redAccent),
    AchievementDefinition(id: 59, category: 'habit', title:
'Чистый лист', description: 'Удалить привычку после достижения
цели.', iconData: Icons.delete_outline, iconColor: Colors.brown),
    AchievementDefinition(id: 60, category: 'habit', title:
'Опытный', description: 'Иметь 10 разных привычек.', iconData:
Icons.school, iconColor: Colors.deepPurple),

    // Combos (61-70)
    AchievementDefinition(id: 61, category: 'combo', title:
'Здоровый день', description: 'Выполнить норму воды, калорий и
сна.', iconData: Icons.favorite, iconColor: Colors.red),

```

```

    AchievementDefinition(id: 62, category: 'combo', title:
'Здоровая неделя', description: 'Выполнить 3 цели 7 дней подряд.',
iconData: Icons.favorite_border, iconColor: Colors.pink),
    AchievementDefinition(id: 63, category: 'combo', title:
'Тройной контроль', description: 'Вода + Калории (норма) + Сон.',
iconData: Icons.games, iconColor: Colors.purple),
    AchievementDefinition(id: 64, category: 'combo', title: 'Утро
бодрое', description: 'Встать до 7:00, вода до 9:00, завтрак.',
iconData: Icons.wb_sunny, iconColor: Colors.orange),
    AchievementDefinition(id: 65, category: 'combo', title: 'Вечер
спокойный', description: 'Уснуть до 23:00, вода после 21:00,
ужин.', iconData: Icons.nightlight, iconColor: Colors.indigo),
    AchievementDefinition(id: 66, category: 'combo', title:
'Полный контроль', description: 'Все цели выполнены 30 дней
подряд.', iconData: Icons.verified, iconColor: Colors.blue),
    AchievementDefinition(id: 67, category: 'combo', title:
'Супер-комбо', description: 'Все трекары + ни одного срыва за
день.', iconData: Icons.stars, iconColor: Colors.amber),
    AchievementDefinition(id: 68, category: 'combo', title:
'Гранд-комбо', description: 'Супер-комбо 7 дней подряд.',
iconData: Icons.emoji_events, iconColor: Colors.amberAccent),
    AchievementDefinition(id: 69, category: 'combo', title:
'Легендарное комбо', description: 'Супер-комбо 30 дней подряд.',
iconData: Icons.workspace_premium, iconColor: Colors.deepOrange),
    AchievementDefinition(id: 70, category: 'combo', title:
'Идеальный профиль', description: 'Активность во всех трекарах за
месяц.', iconData: Icons.person_pin, iconColor: Colors.teal),

    // Milestones (71-85)
    AchievementDefinition(id: 71, category: 'milestone', title:
'Новичок', description: 'Заполнить 10 любых записей.', iconData:
Icons.child_care, iconColor: Colors.green),
    AchievementDefinition(id: 72, category: 'milestone', title:
'Пользователь', description: '100 записей.', iconData:
Icons.person, iconColor: Colors.blue),
    AchievementDefinition(id: 73, category: 'milestone', title:
'Эксперт', description: '500 записей.', iconData: Icons.school,
iconColor: Colors.purple),
    AchievementDefinition(id: 74, category: 'milestone', title:
'Гуру', description: '1000 записей.', iconData:
Icons.self_improvement, iconColor: Colors.deepPurple),
    AchievementDefinition(id: 75, category: 'milestone', title: '7
дней в приложении', description: 'Использовать 7 дней подряд.',
iconData: Icons.looks_one, iconColor: Colors.orange),
    AchievementDefinition(id: 76, category: 'milestone', title:
'30 дней в приложении', description: 'Использовать 30 дней
подряд.', iconData: Icons.event, iconColor: Colors.red),
    AchievementDefinition(id: 77, category: 'milestone', title:
'100 дней в приложении', description: 'Использовать 100 дней
подряд.', iconData: Icons.calendar_month, iconColor:
Colors.redAccent),

```

```

    AchievementDefinition(id: 78, category: 'milestone', title:
'Полгода', description: 'Использовать приложение 183 дня.',
iconData: Icons.timelapse, iconColor: Colors.grey),
    AchievementDefinition(id: 79, category: 'milestone', title:
'Год', description: 'Использовать приложение 365 дней.', iconData:
Icons.grade, iconColor: Colors.amber),
    AchievementDefinition(id: 80, category: 'milestone', title:
'Активный пользователь', description: 'Запись в каждом треке за
неделю.', iconData: Icons.directions_run, iconColor:
Colors.teal),
    AchievementDefinition(id: 81, category: 'milestone', title:
'Трудоголик', description: '20 записей за один день.', iconData:
Icons.work, iconColor: Colors.brown),
    AchievementDefinition(id: 82, category: 'milestone', title:
'Терпеливый', description: 'Заполнять данные 3 месяца.', iconData:
Icons.hourglass_full, iconColor: Colors.blueGrey),
    AchievementDefinition(id: 83, category: 'milestone', title:
'Ветеран', description: 'Заполнять данные 6 месяцев.', iconData:
Icons.military_tech, iconColor: Colors.indigo),
    AchievementDefinition(id: 84, category: 'milestone', title:
'Легенда', description: 'Заполнять данные 12 месяцев.', iconData:
Icons.account_balance, iconColor: Colors.black),
    AchievementDefinition(id: 85, category: 'milestone', title:
'Полная коллекция', description: 'Получить все достижения.',
iconData: Icons.all_inclusive, iconColor: Colors.purple),

    // Special (86-100)
    AchievementDefinition(id: 86, category: 'special', title:
'Настройщик', description: 'Изменить норму воды или калорий.',
iconData: Icons.settings, iconColor: Colors.grey),
    AchievementDefinition(id: 87, category: 'special', title:
'Персонализатор', description: 'Изменить оба лимита.', iconData:
Icons.tune, iconColor: Colors.blueGrey),
    AchievementDefinition(id: 88, category: 'special', title:
'Хранитель данных', description: 'Сделать экспорт/бэкап.',
iconData: Icons.save, iconColor: Colors.green),
    AchievementDefinition(id: 89, category: 'special', title:
'Тёмный рыцарь', description: 'Включить тёмную тему.', iconData:
Icons.dark_mode, iconColor: Colors.black),
    AchievementDefinition(id: 90, category: 'special', title:
'Напоминатель', description: 'Установить напоминания.', iconData:
Icons.notifications_active, iconColor: Colors.orange),
    AchievementDefinition(id: 91, category: 'special', title:
'Испытатель', description: 'Запустить приложение 50 раз.',
iconData: Icons.science, iconColor: Colors.purple),
    AchievementDefinition(id: 92, category: 'special', title:
'Помощник', description: 'Поделиться приложением с другом.',
iconData: Icons.share, iconColor: Colors.blue),
    AchievementDefinition(id: 93, category: 'special', title:
'Синхронист', description: 'Восстановить сессию после
перезапуска.', iconData: Icons.sync, iconColor: Colors.green),

```

```

    AchievementDefinition(id: 94, category: 'special', title:
'Коллекционер', description: 'Получить 25 достижений.', iconData:
Icons.collections, iconColor: Colors.teal),
    AchievementDefinition(id: 95, category: 'special', title:
'Коллекционер-любитель', description: 'Получить 50 достижений.',
iconData: Icons.collections_bookmark, iconColor: Colors.indigo),
    AchievementDefinition(id: 96, category: 'special', title:
'Коллекционер-профи', description: 'Получить 75 достижений.',
iconData: Icons.library_books, iconColor: Colors.purple),
    AchievementDefinition(id: 97, category: 'special', title:
'Коллекционер-мастер', description: 'Получить 100 достижений.',
iconData: Icons.diamond, iconColor: Colors.cyan),
    AchievementDefinition(id: 98, category: 'special', title:
'Открывашка', description: 'Получить первое достижение.',
iconData: Icons.key, iconColor: Colors.amber),
    AchievementDefinition(id: 99, category: 'special', title:
'Секретное', description: 'Спать ровно 7 часов 7 раз подряд.',
iconData: Icons.visibility_off, iconColor: Colors.black26),
    AchievementDefinition(id: 100, category: 'special', title:
'Секретное №2', description: 'Выполнить норму воды 2500 мл 10
раз.', iconData: Icons.water, iconColor: Colors.blueAccent),
  ];
}

```

Файл notification_service.dart

```

import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../providers/app_provider.dart';

class LoginScreen extends StatefulWidget {
  const LoginScreen({Key? key}) : super(key: key);

  @override
  State<LoginScreen> createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {
  final _usernameController = TextEditingController();
  final _passwordController = TextEditingController();
  final _emailController = TextEditingController();
  bool _isLogin = true;

  Future<void> _submit() async {
    final username = _usernameController.text.trim();
    final password = _passwordController.text.trim();
    final email = _emailController.text.trim();

    if (username.isEmpty || password.isEmpty) return;

    if (!_isLogin) {
      if (username.length < 3 || username.length > 20 ||
!RegExp(r'^[a-zA-Z0-9]+$').hasMatch(username)) {

```

```

        ScaffoldMessenger.of(context).showSnackBar(const
SnackBar(content: Text('Имя пользователя должно быть от 3 до 20
символов (буквы и цифры).')));
        return;
    }
    if (password.length < 6) {
        ScaffoldMessenger.of(context).showSnackBar(const
SnackBar(content: Text('Пароль должен быть не менее 6
символов.')));
        return;
    }
    if (email.isEmpty || !email.contains('@')) {
        ScaffoldMessenger.of(context).showSnackBar(const
SnackBar(content: Text('Укажите корректный email.')));
        return;
    }
}

final provider = Provider.of<AppProvider>(context, listen:
false);
bool success;

if (_isLogin) {
    success = await provider.login(username, password);
} else {
    success = await provider.register(username, password,
email);
}

if (!success && mounted) {
    ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(_isLogin ? 'Ошибка входа. Неверный
логин или пароль.' : 'Пользователь уже существует.')),
    );
}

void _showForgotPasswordDialog() {
    final emailCtrl = TextEditingController();
    showDialog(
        context: context,
        builder: (ctx) => AlertDialog(
            title: const Text('Восстановление пароля'),
            content: Column(
                mainAxisAlignment: MainAxisAlignment.min,
                children: [
                    const Text('Введите ваш email для сброса пароля:'),
                    const SizedBox(height: 8),
                    TextField(controller: emailCtrl, decoration: const
InputDecoration(labelText: 'Email')),
                ],
            ),
        ),
    );
}

```

```

        actions: [
            TextButton(onPressed: () => Navigator.pop(ctx), child:
const Text('Отмена')),
            ElevatedButton(
                onPressed: () {
                    if (emailCtrl.text.isNotEmpty) {
                        Navigator.pop(ctx);
                        ScaffoldMessenger.of(context).showSnackBar(
                            const SnackBar(content: Text('Инструкция по
восстановлению отправлена на email.'))
                        );
                    }
                },
                child: const Text('Отправить'),
            ),
        ],
    ),
);
}

```

```

@override
Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(title: const Text('StatLife')),
        body: Center(
            child: SingleChildScrollView(
                padding: const EdgeInsets.all(24.0),
                child: Column(
                    mainAxisAlignment: MainAxisAlignment.center,
                    children: [
                        Text(
                            _isLogin ? 'Вход в систему' : 'Регистрация',
                            style:
Theme.of(context).textTheme.headlineMedium,
                        ),
                        const SizedBox(height: 32),
                        TextField(
                            controller: _usernameController,
                            decoration: const InputDecoration(
                                labelText: 'Имя пользователя',
                                border: OutlineInputBorder(),
                            ),
                        ),
                        const SizedBox(height: 16),
                        if (!_isLogin) ...[
                            TextField(
                                controller: _emailController,
                                decoration: const InputDecoration(
                                    labelText: 'Email',
                                    border: OutlineInputBorder(),
                                ),
                                keyboardType: TextInputType.emailAddress,

```

```

    ),
    const SizedBox(height: 16),
  ],
  TextField(
    controller: _passwordController,
    decoration: const InputDecoration(
      labelText: 'Пароль',
      border: OutlineInputBorder(),
    ),
    obscureText: true,
  ),
  const SizedBox(height: 24),
  if (_isLogin)
    Align(
      alignment: Alignment.centerRight,
      child: TextButton(
        onPressed: _showForgotPasswordDialog,
        child: const Text('Забыли пароль?'),
      ),
    ),
  const SizedBox(height: 8),
  ElevatedButton(
    onPressed: _submit,
    style: ElevatedButton.styleFrom(
      minimumSize: const Size(double.infinity, 50),
    ),
    child: Text(_isLogin ? 'Войти' :
'Зарегистрироваться'),
  ),
  const SizedBox(height: 16),
  TextButton(
    onPressed: () {
      setState(() {
        _isLogin = !_isLogin;
      });
    },
    child: Text(_isLogin ? 'Нет аккаунта?
Зарегистрироваться' : 'Уже есть аккаунт? Войти'),
  ),
),
],
),
),
),
);
}
}

```

Файл export_service.dart

```

import 'dart:io';
import 'package:file_picker/file_picker.dart';
import 'package:path/path.dart' as p;
import 'package:pdf/widgets.dart' as pw;

```

					ПГУ1.090301.43.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		106

```

import 'package:csv/csv.dart';
import '../helpers/database_helper.dart';

class ExportService {
  static Future<String?> exportToCSV() async {
    try {
      final tables = ['water_intake', 'calorie_intake',
'sleep_records', 'habits', 'habit_progress', 'achievements'];

      String? directory = await
FilePicker.platform.getDirectoryPath();
      if (directory == null) return "Экспорт отменен.";

      for (var table in tables) {
        final data = await
DatabaseHelper.instance.exportTable(table);
        if (data.isEmpty) continue;

        List<List<dynamic>> rows = [];
        rows.add(data.first.keys.toList());

        for (var map in data) {
          rows.add(map.values.toList());
        }

        String csvData = const
ListToCsvConverter().convert(rows);
        final file = File(p.join(directory,
'statlife_${table}.csv'));
        await file.writeAsString(csvData);
      }
      return "Экспорт CSV выполнен в $directory";
    } catch (e) {
      return "Ошибка экспорта CSV: $e";
    }
  }

  static Future<String?> exportToPDF(int userId) async {
    try {
      final pdf = pw.Document();

      final db = DatabaseHelper.instance;
      final user = await db.getUserById(userId);
      if (user == null) return "Пользователь не найден";

      final water = await db.getWaterIntakes(userId);
      final cals = await db.getCalorieIntakes(userId);

      pdf.addPage(
        pw.Page(
          build: (pw.Context context) => pw.Column(
            crossAxisAlignment: pw.CrossAxisAlignment.start,

```

```

        children: [
          pw.Text('StatLife      Data      Report',      style:
pw.TextStyle(fontSize: 24, fontWeight: pw.FontWeight.bold)),
          pw.SizedBox(height: 20),
          pw.Text('User: ${user.username}'),
          pw.SizedBox(height: 20),
          pw.Text('Water  Records:  ${water.length}',  style:
const pw.TextStyle(fontSize: 18)),
          pw.SizedBox(height: 20),
          pw.Text('Calorie Records:  ${cals.length}',  style:
const pw.TextStyle(fontSize: 18)),
          pw.SizedBox(height: 40),
          pw.Text('This      report      was      generated
automatically.'),
        ]
      )
    )
  );

  String?      directory      =      await
FilePicker.platform.getDirectoryPath();
  if (directory == null) return "Экспорт отменен.";

  final      file      =      File(p.join(directory,
'statlife_report.pdf'));
  await file.writeAsBytes(await pdf.save());

  return "Экспорт PDF выполнен в $directory";
} catch (e) {
  return "Ошибка экспорта PDF: $e";
}
}
}

```

Файл login_screen.dart

```

import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../providers/app_provider.dart';

class LoginScreen extends StatefulWidget {
  const LoginScreen({Key? key}) : super(key: key);

  @override
  State<LoginScreen> createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {
  final _usernameController = TextEditingController();
  final _passwordController = TextEditingController();
  final _emailController = TextEditingController();
  bool _isLogin = true;

```

```

Future<void> _submit() async {
  final username = _usernameController.text.trim();
  final password = _passwordController.text.trim();
  final email = _emailController.text.trim();

  if (username.isEmpty || password.isEmpty) return;

  if (!_isLoggedIn) {
    if (username.length < 3 || username.length > 20 ||
!RegExp(r'^[a-zA-Z0-9]+$').hasMatch(username)) {
      ScaffoldMessenger.of(context).showSnackBar(const
SnackBar(content: Text('Имя пользователя должно быть от 3 до 20
символов (буквы и цифры).')));
      return;
    }
    if (password.length < 6) {
      ScaffoldMessenger.of(context).showSnackBar(const
SnackBar(content: Text('Пароль должен быть не менее 6
символов.')));
      return;
    }
    if (email.isEmpty || !email.contains('@')) {
      ScaffoldMessenger.of(context).showSnackBar(const
SnackBar(content: Text('Укажите корректный email.')));
      return;
    }
  }

  final provider = Provider.of<AuthProvider>(context, listen:
false);
  bool success;

  if (_isLoggedIn) {
    success = await provider.login(username, password);
  } else {
    success = await provider.register(username, password,
email);
  }

  if (!success && mounted) {
    ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(_isLoggedIn ? 'Ошибка входа. Неверный
логин или пароль.' : 'Пользователь уже существует.')),
    );
  }
}

void _showForgotPasswordDialog() {
  final emailCtrl = TextEditingController();
  showDialog(
    context: context,
    builder: (ctx) => AlertDialog(

```

```

        title: const Text('Восстановление пароля'),
        content: Column(
          mainAxisAlignment: MainAxisAlignment.min,
          children: [
            const Text('Введите ваш email для сброса пароля:'),
            const SizedBox(height: 8),
            TextField(controller: emailCtrl, decoration: const
InputDecoration(labelText: 'Email')),
          ],
        ),
        actions: [
          TextButton(onPressed: () => Navigator.pop(ctx), child:
const Text('Отмена')),
          ElevatedButton(
            onPressed: () {
              if (emailCtrl.text.isNotEmpty) {
                Navigator.pop(ctx);
                ScaffoldMessenger.of(context).showSnackBar(
                  const SnackBar(content: Text('Инструкция по
восстановлению отправлена на email.'))
                );
              }
            },
            child: const Text('Отправить'),
          ),
        ],
      ),
    );
  }

```

```

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: const Text('StatLife')),
    body: Center(
      child: SingleChildScrollView(
        padding: const EdgeInsets.all(24.0),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text(
              _isLogin ? 'Вход в систему' : 'Регистрация',
              style:
Theme.of(context).textTheme.headlineMedium,
            ),
            const SizedBox(height: 32),
            TextField(
              controller: _usernameController,
              decoration: const InputDecoration(
                labelText: 'Имя пользователя',
                border: OutlineInputBorder(),
              ),
            ),

```

```

),
const SizedBox(height: 16),
if (!_isLogin) ...[
  TextField(
    controller: _emailController,
    decoration: const InputDecoration(
      labelText: 'Email',
      border: OutlineInputBorder(),
    ),
    keyboardType: TextInputType.emailAddress,
  ),
  const SizedBox(height: 16),
],
TextField(
  controller: _passwordController,
  decoration: const InputDecoration(
    labelText: 'Пароль',
    border: OutlineInputBorder(),
  ),
  obscureText: true,
),
const SizedBox(height: 24),
if (_isLogin)
  Align(
    alignment: Alignment.centerRight,
    child: TextButton(
      onPressed: _showForgotPasswordDialog,
      child: const Text('Забыли пароль?'),
    ),
  ),
const SizedBox(height: 8),
ElevatedButton(
  onPressed: _submit,
  style: ElevatedButton.styleFrom(
    minimumSize: const Size(double.infinity, 50),
  ),
  child: Text(_isLogin ? 'Войти' :
'Зарегистрироваться'),
),
const SizedBox(height: 16),
TextButton(
  onPressed: () {
    setState(() {
      _isLogin = !_isLogin;
    });
  },
  child: Text(_isLogin ? 'Нет аккаунта?
Зарегистрироваться' : 'Уже есть аккаунт? Войти'),
),
],
),
),
),

```

```

    ),
  );
}
}

```

Файл home_screen.dart

```

import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../providers/app_provider.dart';
import 'settings_screen.dart';
import 'charts_screen.dart';
import 'weight_screen.dart';

class HomeScreen extends StatelessWidget {
  final void Function(int) onNavigate;
  const HomeScreen({Key? key, required this.onNavigate}) :
    super(key: key);

  @override
  Widget build(BuildContext context) {
    final provider = Provider.of<AppProvider>(context);
    final user = provider.currentUser;

    if (user == null) return const Center(child:
CircularProgressIndicator());

    return Scaffold(
      appBar: AppBar(
        title: const Text('Сводка'),
        actions: [
          IconButton(
            icon: const Icon(Icons.bar_chart),
            onPressed: () {
              Navigator.push(
                context,
                MaterialPageRoute(builder: (context) => const
ChartsScreen()),
              );
            },
          ),
          IconButton(
            icon: const Icon(Icons.settings),
            onPressed: () {
              Navigator.push(
                context,
                MaterialPageRoute(builder: (context) => const
SettingsScreen()),
              );
            },
          ),
          IconButton(
            icon: const Icon(Icons.logout),

```

```

        onPressed: () => provider.logout(),
      ),
    ],
  ),
  body: Padding(
    padding: const EdgeInsets.all(16.0),
    child: GridView.count(
      crossAxisCount: 2,
      crossAxisSpacing: 16,
      mainAxisSpacing: 16,
      children: [
        _buildSummaryCard(
          context,
          'Сон',
          Icons.bedtime,
          Colors.indigo,
          '${provider.sleepRecords.isNotEmpty
(provider.sleepRecords.first.durationMinutes
60).toStringAsFixed(1) : 0} ч',
          () => onNavigate(1),
        ),
        _buildSummaryCard(
          context,
          'Питание',
          Icons.restaurant,
          Colors.orange,
          '${provider.todayCalories} ккал',
          () => onNavigate(2),
        ),
        _buildSummaryCard(
          context,
          'Вода',
          Icons.water_drop,
          Colors.blue,
          '${provider.todayWater} мл',
          () => onNavigate(3),
        ),
        _buildSummaryCard(
          context,
          'Привычки',
          Icons.track_changes,
          Colors.green,
          'Отслеживается: ${provider.habits.length}',
          () => onNavigate(4),
        ),
        _buildSummaryCard(
          context,
          'Вес',
          Icons.monitor_weight,
          Colors.teal,
          'Журнал',
          () => Navigator.push(

```

?

/

```

        context,
        MaterialPageRoute(builder: (context) => const
WeightScreen()),
    ),
  ),
  _buildSummaryCard(
    context,
    'Достижения',
    Icons.emoji_events,
    Colors.amber,
    '${provider.earnedAchievements.length} / 100',
    () => onNavigate(5),
  ),
],
),
),
);
}

```

```

Widget _buildSummaryCard(BuildContext context, String title,
IconData icon, Color color, String value, VoidCallback? onTap) {
  return Card(
    elevation: 4,
    shape: RoundedRectangleBorder(borderRadius:
BorderRadius.circular(16)),
    child: InkWell(
      onTap: onTap,
      borderRadius: BorderRadius.circular(16),
      child: Container(
        padding: const EdgeInsets.all(16),
        decoration: BoxDecoration(
          borderRadius: BorderRadius.circular(16),
          gradient: LinearGradient(
            colors: [color.withOpacity(0.7), color],
            begin: Alignment.topLeft,
            end: Alignment.bottomRight,
          ),
        ),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Icon(icon, size: 48, color: Colors.white),
          const SizedBox(height: 8),
          Text(
            title,
            style: const TextStyle(
              color: Colors.white,
              fontSize: 18,
              fontWeight: FontWeight.bold,
            ),
          ),
          const SizedBox(height: 8),

```

```

        Text(
          value,
          style: const TextStyle(
            color: Colors.white,
            fontSize: 16,
          ),
        ),
      ],
    ),
  ),
);
}
}

```

Файл water_screen.dart

```

import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:table_calendar/table_calendar.dart';
import '../providers/app_provider.dart';

class WaterScreen extends StatelessWidget {
  const WaterScreen({Key? key}) : super(key: key);

  void _addCustomWaterDialog(BuildContext context) {
    final volCtrl = TextEditingController();
    showDialog(
      context: context,
      builder: (ctx) => AlertDialog(
        title: const Text('Добавить воду'),
        content: TextField(
          controller: volCtrl,
          keyboardType: TextInputType.number,
          decoration: const InputDecoration(labelText: 'Объём
(мл)'),
        ),
        actions: [
          TextButton(onPressed: () => Navigator.pop(ctx), child:
const Text('Отмена')),
          ElevatedButton(
            onPressed: () {
              final vol = int.tryParse(volCtrl.text);
              if (vol != null && vol > 0) {
                if (vol > 5000) {

ScaffoldMessenger.of(context).showSnackBar(const
SnackBar(content: Text('Объём за раз не должен превышать 5000
мл')));

                return;
              }
              Provider.of<AppProvider>(context, listen:
false).addWater(vol);
              Navigator.pop(ctx);

```

```

        }
    },
    child: const Text('Добавить'),
)
],
),
);
}

void _editGoalDialog(BuildContext context) {
    final volCtrl = TextEditingController(
        text: Provider.of<AppProvider>(context, listen:
false).settings?.dailyWaterMl.toString() ?? '2500'
    );
    showDialog(
        context: context,
        builder: (ctx) => AlertDialog(
            title: const Text('Изменить норму'),
            content: TextField(
                controller: volCtrl,
                keyboardType: TextInputType.number,
                decoration: const InputDecoration(labelText: 'Норма
(мл)'),
            ),
            actions: [
                TextButton(onPressed: () => Navigator.pop(ctx), child:
const Text('Отмена')),
                ElevatedButton(
                    onPressed: () {
                        final vol = int.tryParse(volCtrl.text);
                        if (vol != null && vol > 0) {
                            Provider.of<AppProvider>(context, listen:
false).updateWaterGoal(vol);
                            Navigator.pop(ctx);
                        }
                    },
                    child: const Text('Сохранить'),
                )
            ],
        ),
    );
}

@override
Widget build(BuildContext context) {
    final provider = Provider.of<AppProvider>(context);
    final totalWater = provider.todayWater;
    final targetWater = provider.settings?.dailyWaterMl ?? 2500;

    final progress = (totalWater / targetWater).clamp(0.0, 1.0);
    final color = totalWater >= targetWater ? Colors.green :
Colors.blue;

```

```

        final todayIntakes = provider.waterIntakes.where((w) {
            final isoDate =
DateTime.now().toIso8601String().substring(0, 10);
            return w.timestamp.startsWith(isoDate);
        }).toList();

return DefaultTabController(
    length: 2,
    child: Scaffold(
        appBar: AppBar(
            title: const Text('Трекер воды'),
            actions: [
                IconButton(
                    icon: const Icon(Icons.settings),
                    onPressed: () => _editGoalDialog(context),
                ),
            ],
            bottom: const TabBar(
                tabs: [
                    Tab(text: 'За день'),
                    Tab(text: 'Календарь'),
                ],
            ),
        ),
        body: TabBarView(
            children: [
                Padding(
                    padding: const EdgeInsets.all(16.0),
                    child: Column(
                        children: [
                            Stack(
                                alignment: Alignment.center,
                                children: [
                                    SizedBox(
                                        width: 200,
                                        height: 200,
                                        child: CircularProgressIndicator(
                                            value: progress,
                                            strokeWidth: 20,
                                            backgroundColor: Colors.grey[300],
                                            color: color,
                                        ),
                                    ),
                                ],
                            ),
                            Column(
                                children: [
                                    const Icon(Icons.water_drop, size: 40,
color: Colors.blue),
                                    Text(
                                        '$totalWater мл',
                                        style: const TextStyle(fontSize: 24,
fontWeight: FontWeight.bold),

```

```

        ),
        Text('/ $targetWater мл', style: const
TextStyle(fontSize: 16)),
    ],
),
],
),
const SizedBox(height: 32),
Row(
  mainAxisAlignment:
MainAxisAlignment.spaceEvenly,
  children: [
    ElevatedButton.icon(
      icon: const Icon(Icons.add),
      label: const Text('+ 250 мл'),
      onPressed: () => provider.addWater(250),
    ),
    OutlinedButton.icon(
      icon: const Icon(Icons.edit),
      label: const Text('Свой объём'),
      onPressed: () =>
_addCustomWaterDialog(context),
    ),
  ],
),
const SizedBox(height: 32),
const Align(
  alignment: Alignment.centerLeft,
  child: Text('Выпито сегодня:', style:
TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),
),
const SizedBox(height: 16),
Expanded(
  child: ListView.builder(
    itemCount: todayIntakes.length,
    itemBuilder: (context, index) {
      final intake = todayIntakes[index];
      final timeObj =
DateTime.parse(intake.timestamp);
      final timeStr =
'${timeObj.hour.toString().padLeft(2, '0')}:${timeObj.minute.toSt
ring().padLeft(2, '0')}';
      return Dismissible(
        key: Key(intake.id.toString()),
        direction: DismissDirection.endToStart,
        background: Container(
          color: Colors.red,
          alignment: Alignment.centerRight,
          padding: const
EdgeInsets.symmetric(horizontal: 20),
          child: const Icon(Icons.delete,
color: Colors.white),

```



```

        calendarFormat: CalendarFormat.month,
        headerStyle: const HeaderStyle(formatButtonVisible:
false),
        calendarBuilders: CalendarBuilders(
          defaultBuilder: (context, day, focusedDay) {
            final normalizedDay = DateTime.utc(day.year,
day.month, day.day);
            if (events.containsKey(normalizedDay)) {
              final total = events[normalizedDay]!;
              final color = total >= targetWater ? Colors.green :
Colors.red;
              return Container(
                margin: const EdgeInsets.all(4.0),
                decoration: BoxDecoration(
                  color: color,
                  shape: BoxShape.circle,
                ),
                child: Center(
                  child: Text('${day.day}', style: const
TextStyle(color: Colors.white)),
                ),
              );
            }
            return null;
          },
        ),
      ),
    );
  }
}

```

Файл nutrition_screen.dart

```

import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:table_calendar/table_calendar.dart';
import '../providers/app_provider.dart';

class NutritionScreen extends StatefulWidget {
  const NutritionScreen({Key? key}) : super(key: key);

  @override
  State<NutritionScreen> createState() =>
  _NutritionScreenState();
}

class _NutritionScreenState extends State<NutritionScreen> {
  void _addMealDialog(BuildContext context) {
    final caloriesCtrl = TextEditingController();
    final proteinCtrl = TextEditingController(text: '0');
    final fatCtrl = TextEditingController(text: '0');
    final carbsCtrl = TextEditingController(text: '0');
    final descCtrl = TextEditingController();
  }
}

```

```

String mealType = 'Завтрак';

showDialog(
    context: context,
    builder: (ctx) => AlertDialog(
        title: const Text('Добавить приём пищи'),
        content: Column(
            mainAxisAlignment: MainAxisAlignment.min,
            children: [
                DropdownButtonFormField<String>(
                    value: mealType,
                    items:      ['Завтрак',      'Обед',      'Ужин',
'Перекус'].map((e)      =>      DropdownMenuItem(value:      e,      child:
Text(e))).toList(),
                    onChanged: (val) => mealType = val ?? 'Завтрак',
                    decoration:      const      InputDecoration(labelText:
'Тип'),
                ),
                TextField(
                    controller: caloriesCtrl,
                    keyboardType: TextInputType.number,
                    decoration:      const      InputDecoration(labelText:
'Калории (ккал)'),
                ),
                Row(
                    children: [
                        Expanded(child:      TextField(controller:
proteinCtrl,      keyboardType:      TextInputType.number,      decoration:
const      InputDecoration(labelText: 'Белки (г)')),
                        const SizedBox(width: 8),
                        Expanded(child:      TextField(controller:      fatCtrl,
keyboardType:      TextInputType.number,      decoration:      const
InputDecoration(labelText: 'Жиры (г)')),
                        const SizedBox(width: 8),
                        Expanded(child:      TextField(controller:      carbsCtrl,
keyboardType:      TextInputType.number,      decoration:      const
InputDecoration(labelText: 'Углеводы (г)')),
                    ],
                ),
                TextField(
                    controller: descCtrl,
                    decoration:      const      InputDecoration(labelText:
'Описание'),
                ),
            ],
        ),
        actions: [
            TextButton(onPressed: () => Navigator.pop(ctx), child:
const Text('Отмена')),
            ElevatedButton(
                onPressed: () {
                    final cals = int.tryParse(caloriesCtrl.text);

```

```

        final prot = int.tryParse(proteinCtrl.text) ?? 0;
        final f = int.tryParse(fatCtrl.text) ?? 0;
        final carb = int.tryParse(carbsCtrl.text) ?? 0;
        if (cals != null) {
            if (cals < 0 || cals > 10000) {

ScaffoldMessenger.of(context).showSnackBar(const
SnackBar(content: Text('Калории должны быть от 0 до 10000')));
            return;
        }
        Provider.of<AppProvider>(context,          listen:
false).addCalorie(
            cals,
            descCtrl.text,
            mealType,
            protein: prot,
            fat: f,
            carbs: carb,
        );
        Navigator.pop(ctx);
    }
    },
    child: const Text('Добавить'),
)
    ],
),
);
}

void _editGoalDialog(BuildContext context) {
    final calsCtrl = TextEditingController(
        text:          Provider.of<AppProvider>(context,          listen:
false).settings?.dailyCalories.toString() ?? '2500'
    );
    showDialog(
        context: context,
        builder: (ctx) => AlertDialog(
            title: const Text('Изменить норму'),
            content: TextField(
                controller: calsCtrl,
                keyboardType: TextInputType.number,
                decoration: const InputDecoration(labelText: 'Норма
(ккал)'),
            ),
            actions: [
                TextButton(onPressed: () => Navigator.pop(ctx), child:
const Text('Отмена')),
                ElevatedButton(
                    onPressed: () {
                        final cals = int.tryParse(calsCtrl.text);
                        if (cals != null && cals > 0) {

```

```

        Provider.of<AppProvider>(context, listen:
false).updateCalorieGoal(cals);
        Navigator.pop(ctx);
    }
},
child: const Text('Сохранить'),
)
],
),
);
}

@override
Widget build(BuildContext context) {
    final provider = Provider.of<AppProvider>(context);
    final totalCals = provider.todayCalories;
    final targetCals = provider.settings?.dailyCalories ?? 2500;

    Color progressColor = Colors.green;
    if (totalCals > targetCals + 100) {
        progressColor = Colors.red;
    } else if (totalCals > targetCals - 100 && totalCals <=
targetCals + 100) {
        progressColor = Colors.amber;
    }

    final todayIntakes = provider.calorieIntakes.where((i) {
        final isoDate =
DateTime.now().toIso8601String().substring(0, 10);
        return i.timestamp.startsWith(isoDate);
    }).toList();

    return DefaultTabController(
        length: 2,
        child: Scaffold(
            appBar: AppBar(
                title: const Text('Трекер питания'),
                actions: [
                    IconButton(
                        icon: const Icon(Icons.settings),
                        onPressed: () => _editGoalDialog(context),
                    ),
                ],
            bottom: const TabBar(
                tabs: [
                    Tab(text: 'За день'),
                    Tab(text: 'Календарь'),
                ],
            ),
            floatingActionButton: FloatingActionButton(
                onPressed: () => _addMealDialog(context),
            ),
        ),
    );
}

```

```

        child: const Icon(Icons.add),
      ),
      body: TabBarView(
        children: [
          Padding(
            padding: const EdgeInsets.all(16.0),
            child: Column(
              children: [
                Container(
                  padding: const EdgeInsets.all(16.0),
                  decoration: BoxDecoration(
                    color: progressColor.withOpacity(0.2),
                    borderRadius: BorderRadius.circular(16),
                    border: Border.all(color: progressColor,
width: 2),
                  ),
                  child: Row(
                    mainAxisAlignment:
MainAxisAlignment.spaceBetween,
                    children: [
                      Column(
                        crossAxisAlignment:
CrossAxisAlignment.start,
                        children: [
                          const Text('Калории сегодня', style:
TextStyle(fontSize: 16)),
                          Text(
                            '$totalCals / $targetCals ккал',
                            style: TextStyle(fontSize: 24,
fontWeight: FontWeight.bold, color: progressColor),
                          ),
                        ],
                      ),
                      Icon(Icons.monitor_weight, size: 48,
color: progressColor),
                    ],
                  ),
                ),
                const SizedBox(height: 16),
                Row(
                  mainAxisAlignment:
MainAxisAlignment.spaceAround,
                  children: [
                    buildMacrosIndicator('Белки',
provider.todayProtein, 150, Colors.red),
                    buildMacrosIndicator('Жиры',
provider.todayFat, 80, Colors.orange),
                    buildMacrosIndicator('Углеводы',
provider.todayCarbs, 300, Colors.blue),
                  ],
                ),
                const SizedBox(height: 24),

```

```
const Align(
    alignment: Alignment.centerLeft,
    child: Text('Приёмы пищи за сегодня:', style:
TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),
),
const SizedBox(height: 16),
Expanded(
    child: ListView.builder(
        itemCount: todayIntakes.length,
        itemBuilder: (context, index) {
            final intake = todayIntakes[index];
            final timeObj =
DateTime.parse(intake.timestamp);
            final timeStr =
'${timeObj.hour.toString().padLeft(2, '0')}:${timeObj.minute.toSt
ring().padLeft(2, '0')}';
            return Dismissible(
                key: Key(intake.id.toString()),
                direction: DismissDirection.endToStart,
                background: Container(
                    color: Colors.red,
                    alignment: Alignment.centerRight,
                    padding: const
EdgeInsets.symmetric(horizontal: 20),
                    child: const Icon(Icons.delete,
color: Colors.white),
                ),
                onDismissed: (_) {
                    if (intake.id != null) {
provider.deleteCalorie(intake.id!);
                    }
                },
                child: Card(
                    child: ListTile(
                        leading: const CircleAvatar(child:
Icon(Icons.fastfood)),
                        title: Text('${intake.mealType} -
${intake.calories} ккал'),
                        subtitle:
Text(intake.description.isNotEmpty ? intake.description : 'Без
описания'),
                        trailing: Text(timeStr),
                    ),
                ),
            );
        },
    ),
),
],
),
),
```

```

        _buildCalendarTab(context, provider),
    ],
),
),
);
}

Widget _buildCalendarTab(BuildContext context, AppProvider
provider) {
    final targetCals = provider.settings?.dailyCalories ?? 2500;

    final events = <DateTime, int>{};
    for (var c in provider.calorieIntakes) {
        final date = DateTime.parse(c.timestamp);
        final normalizedDate = DateTime.utc(date.year, date.month,
date.day);
        events[normalizedDate] = (events[normalizedDate] ?? 0) +
c.calories;
    }

    return Padding(
        padding: const EdgeInsets.all(16.0),
        child: TableCalendar(
            firstDay: DateTime.utc(2020, 1, 1),
            lastDay: DateTime.utc(2030, 12, 31),
            focusedDay: DateTime.now(),
            calendarFormat: CalendarFormat.month,
            headerStyle: const HeaderStyle(formatButtonVisible:
false),
            calendarBuilders: CalendarBuilders(
                defaultBuilder: (context, day, focusedDay) {
                    final normalizedDay = DateTime.utc(day.year,
day.month, day.day);
                    if (events.containsKey(normalizedDay)) {
                        final total = events[normalizedDay]!;
                        Color color = Colors.green;
                        if (total > targetCals + 100) {
                            color = Colors.red;
                        } else if (total > targetCals - 100 && total <=
targetCals + 100) {
                            color = Colors.amber;
                        }
                    }
                    return Container(
                        margin: const EdgeInsets.all(4.0),
                        decoration: BoxDecoration(
                            color: color,
                            shape: BoxShape.circle,
                        ),
                        child: Center(
                            child: Text('${day.day}', style: const
TextStyle(color: Colors.white)),
                        ),

```

```

        );
    }
    return null;
  },
),
),
);
}

Widget _buildMacrosIndicator(String label, int value, int
target, Color color) {
  return Column(
    children: [
      Text(label, style: const TextStyle(fontSize: 12)),
      const SizedBox(height: 4),
      Stack(
        alignment: Alignment.center,
        children: [
          SizedBox(
            width: 50,
            height: 50,
            child: CircularProgressIndicator(
              value: (value / target).clamp(0.0, 1.0),
              strokeWidth: 6,
              backgroundColor: color.withOpacity(0.1),
              color: color,
            ),
          ),
          Text('$value', style: const TextStyle(fontSize: 12,
fontWeight: FontWeight.bold)),
        ],
      ),
    ],
  );
}
}

```

Файл sleep_screen.dart

```

import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:table_calendar/table_calendar.dart';
import '../providers/app_provider.dart';
import '../services/notification_service.dart';

class SleepScreen extends StatefulWidget {
  const SleepScreen({Key? key}) : super(key: key);

  @override
  State<SleepScreen> createState() => _SleepScreenState();
}

class _SleepScreenState extends State<SleepScreen> {

```

```

TimeOfDay? _alarmTime;

@override
Widget build(BuildContext context) {
  final provider = Provider.of<AppProvider>(context);
  final settings = provider.settings;
  final records = provider.sleepRecords;

  final events = <DateTime, SleepRecord>{};
  for (var r in records) {
    final date = DateTime.parse(r.date);
    final normalizedDate = DateTime.utc(date.year, date.month,
date.day);
    events[normalizedDate] = r;
  }

  return Scaffold(
    appBar: AppBar(title: const Text('Трекер сна')),
    body: SingleChildScrollView(
      child: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.stretch,
          children: [
            Card(
              child: Padding(
                padding: const EdgeInsets.all(16.0),
                child: Column(
                  children: [
                    Row(
                      mainAxisAlignment:
MainAxisAlignment.spaceBetween,
                      children: [
                        const Text('Будильник (Цель):', style:
TextStyle(fontSize: 18)),
                        TextButton(
                          onPressed: () async {
                            final time = await showTimePicker(
                              context: context,
                              initialTime: _alarmTime ?? const
TimeOfDay(hour: 7, minute: 0),
                            );
                            if (time != null) {
                              setState(() => _alarmTime =
time);
                              await
NotificationService.instance.scheduleDailyNotification(
                                200,
                                'Будильник',
                                'Пора просыпаться!',
                                time
                              );

```

```

        if (mounted) {
ScaffoldMessenger.of(context).showSnackBar(
                SnackBar(content:
Text('Будильник установлен на ${time.format(context)}')),
                );
        }
    },
    child: Text(
        _alarmTime == null ? 'Установить' :
        _alarmTime!.format(context),
        style: const TextStyle(fontSize:
18),
    ),
),
],
),
const SizedBox(height: 16),
Row(
    mainAxisAlignment:
MainAxisAlignment.spaceEvenly,
    children: [
        ElevatedButton.icon(
            icon: const Icon(Icons.bed),
            label: const Text('Начать сон'),
            style: ElevatedButton.styleFrom(
                backgroundColor:
provider.activeSleepStart != null ? Colors.grey : Colors.indigo,
                foregroundColor: Colors.white,
            ),
            onPressed: provider.activeSleepStart
!= null ? null : () async {
                await provider.startSleep();
                if (mounted) {
ScaffoldMessenger.of(context).showSnackBar(
                    const SnackBar(content:
Text('Спокойной ночи! Время засыпания зафиксировано.')),
                );
            },
        ),
        ElevatedButton.icon(
            icon: const Icon(Icons.wb_sunny),
            label: const Text('Проснулся'),
            style: ElevatedButton.styleFrom(
                backgroundColor: Colors.orange,
                foregroundColor: Colors.white,
            ),
            onPressed: provider.activeSleepStart
== null ? null : () async {

```

```

        int duration = await
provider.endSleep();

        if (mounted) {
            final hours = duration / 60.0;
            final hoursStr =
hours.toStringAsFixed(1);

            final isGoodSleep = hours >=
(settings?.targetSleepHours ?? 7.0);

ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(
        content: Text('Вы спали
$hoursStr часов', style: const TextStyle(color: Colors.white)),
        backgroundColor: isGoodSleep
? Colors.green : Colors.red,
    ),
);
}
},
),
],
),
),
),
),
const SizedBox(height: 24),
const Text('График сна (последние 30 дней)', style:
TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),
const SizedBox(height: 16),
TableCalendar(
    firstDay: DateTime.utc(2020, 1, 1),
    lastDay: DateTime.utc(2030, 12, 31),
    focusedDay: DateTime.now(),
    calendarFormat: CalendarFormat.month,
    headerStyle: const
HeaderStyle(formatButtonVisible: false),
    calendarBuilders: CalendarBuilders(
        defaultBuilder: (context, day, focusedDay) {
            final normalizedDay = DateTime.utc(day.year,
day.month, day.day);

            if (events.containsKey(normalizedDay)) {
                final record = events[normalizedDay]!;
                final hours = record.durationMinutes / 60.0;
                final color = (hours >=
(settings?.targetSleepHours ?? 7.0)) ? Colors.green : Colors.red;
                return Container(
                    margin: const EdgeInsets.all(4.0),
                    decoration: BoxDecoration(
                        color: color,
                        shape: BoxShape.circle,
                    ),

```

```

        child: Center(
          child: Text(
            '${day.day}',
            style: const TextStyle(color:
Colors.white),
          ),
        ),
      ),
    );
  }
  return null;
},
),
],
),
),
),
);
}
}

```

Файл weight_screen.dart

```

import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:intl/intl.dart';
import 'package:fl_chart/fl_chart.dart';
import '../providers/app_provider.dart';
import '../models/app_models.dart';
import '../helpers/database_helper.dart';

class WeightScreen extends StatefulWidget {
  const WeightScreen({Key? key}) : super(key: key);

  @override
  State<WeightScreen> createState() => _WeightScreenState();
}

class _WeightScreenState extends State<WeightScreen> {
  List<WeightRecord> _records = [];
  bool _isLoading = true;

  @override
  void initState() {
    super.initState();
    _loadData();
  }

  Future<void> _loadData() async {
    final provider = Provider.of<AppProvider>(context, listen:
false);
    if (provider.currentUser != null) {

```

```

        final data = await
DatabaseHelper.instance.getWeightRecords(provider.currentUser!.id!);

        setState(() {
            _records = data;
            _isLoading = false;
        });
    }
}

void _addRecord() {
    final ctrl = TextEditingController();
    DateTime selectedDate = DateTime.now();

    showDialog(
        context: context,
        builder: (ctx) => StatefulBuilder(
            builder: (ctx, setStateSB) => AlertDialog(
                title: const Text('Записать вес'),
                content: Column(
                    mainAxisAlignment: MainAxisAlignment.min,
                    children: [
                        TextField(
                            controller: ctrl,
                            keyboardType: TextInputType.numberWithOptions(decimal: true),
                            decoration: const InputDecoration(labelText: 'Вес (кг)'),
                        ),
                        const SizedBox(height: 16),
                        Row(
                            mainAxisAlignment: MainAxisAlignment.spaceBetween,
                            children: [
                                const Text('Дата:'),
                                TextButton(
                                    onPressed: () async {
                                        final d = await showDatePicker(
                                            context: context,
                                            initialDate: selectedDate,
                                            firstDate: DateTime(2000),
                                            lastDate: DateTime.now(),
                                        );
                                        if (d != null) {
                                            setStateSB(() => selectedDate = d);
                                        }
                                    },
                                    child: Text(DateFormat('dd.MM.yyyy').format(selectedDate)),
                                ),
                            ],
                        ),
                    ],
                ),
            ),
    );
}

```

```

        ],
    ),
    actions: [
        TextButton(onPressed: () => Navigator.pop(ctx),
child: const Text('Отмена')),
        ElevatedButton(
            onPressed: () async {
                if (ctrl.text.isNotEmpty) {
                    double? w =
double.tryParse(ctrl.text.replaceAll(',', '.'));
                    if (w != null) {
                        if (w < 20 || w > 300) {

ScaffoldMessenger.of(context).showSnackBar(const
SnackBar(content: Text('Вес должен быть от 20 до 300 кг')));
                        return;
                    }
                    final provider =
Provider.of<AppProvider>(context, listen: false);
                    final rec = WeightRecord(
                        userId: provider.currentUser!.id!,
                        date: selectedDate.toIso8601String(),
                        weight: w,
                    );
                    await
DatabaseHelper.instance.createWeightRecord(rec);
                    Navigator.pop(ctx);
                    _loadData();
                }
            },
            child: const Text('Сохранить'),
        )
    ],
),
);
}

List<FlSpot> _getSpots() {
    if (_records.isEmpty) return [];
    final sorted = List<WeightRecord>.from(_records);
    sorted.sort((a, b) =>
DateTime.parse(a.date).compareTo(DateTime.parse(b.date)));

    List<FlSpot> spots = [];
    final firstDate = DateTime.parse(sorted.first.date);
    for (var r in sorted) {
        final d = DateTime.parse(r.date);
        final days = d.difference(firstDate).inDays.toDouble();
        spots.add(FlSpot(days, r.weight));
    }
}

```

```

    return spots;
}

@override
Widget build(BuildContext context) {
    if (_isLoading) return const Scaffold(body: Center(child:
CircularProgressIndicator()));

    return Scaffold(
        appBar: AppBar(title: const Text('Трекер веса')),
        floatingActionButton: FloatingActionButton(
            onPressed: _addRecord,
            child: const Icon(Icons.add),
        ),
        body: _records.isEmpty
            ? const Center(child: Text('Нет записей веса'))
            : Column(
                children: [
                    const SizedBox(height: 16),
                    const Text('График изменения веса', style:
TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),
                    const SizedBox(height: 16),
                    SizedBox(
                        height: 250,
                        width: double.infinity,
                        child: Padding(
                            padding: const
EdgeInsets.symmetric(horizontal: 16.0),
                            child: LineChart(
                                LineChartData(
                                    minY: 0,
                                    lineBarsData: [
                                        LineChartBarData(
                                            spots: _getSpots(),
                                            isCurved: true,
                                            color: Colors.green,
                                            barWidth: 3,
                                            dotData: const FlDotData(show: true),
                                            belowBarData: BarAreaData(show: true,
color: Colors.green.withOpacity(0.3)),
                                        )
                                    ],
                                    titlesData: const FlTitlesData(
                                        rightTitles: AxisTitles(sideTitles:
SideTitles(showTitles: false)),
                                        topTitles: AxisTitles(sideTitles:
SideTitles(showTitles: false)),
                                        bottomTitles: AxisTitles(sideTitles:
SideTitles(showTitles: false)),
                                    ),
                                ),
                            ),
                        ),
                    ),
                ],
            ),
    );
}

```

```

    ),
  ),
  const Divider(height: 32),
  Expanded(
    child: ListView.builder(
      itemCount: _records.length,
      itemBuilder: (context, i) {
        final r = _records[i];
        return ListTile(
          leading: const Icon(Icons.monitor_weight,
color: Colors.green),
          title:
Text('${r.weight.toStringAsFixed(1)} кг', style: const
TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),
          subtitle: Text(DateFormat('dd MMMM
yyyy').format(DateTime.parse(r.date))),
        );
      },
    ),
  ),
],
),
);
}
}

```

Файл habits_screen.dart

```

import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:intl/intl.dart';
import '../providers/app_provider.dart';
import '../models/app_models.dart';

class HabitsScreen extends StatefulWidget {
  const HabitsScreen({Key? key}) : super(key: key);

  @override
  State<HabitsScreen> createState() => _HabitsScreenState();
}

class _HabitsScreenState extends State<HabitsScreen> {
  void _addHabitDialog(BuildContext context) {
    final nameCtrl = TextEditingController();
    final targetDaysCtrl = TextEditingController();
    String category = 'Курение';
    DateTime selectedDate = DateTime.now();

    showDialog(
      context: context,
      builder: (ctx) => StatefulBuilder(
        builder: (context, setStateSB) {
          return AlertDialog(

```

```

        title: const Text('Добавить привычку'),
        content: Column(
          mainAxisAlignment: MainAxisAlignment.min,
          children: [
            TextField(
              controller: nameCtrl,
              decoration: const InputDecoration(labelText:
'Название привычки'),
            ),
            DropdownButtonFormField<String>(
              value: category,
              items: ['Курение', 'Алкоголь', 'Сладкое',
'Sоцсети', 'Другое']
                .map((e) => DropdownMenuItem(value: e,
child: Text(e)))
                .toList(),
              onChanged: (val) => setStateSB(() => category =
val ?? 'Другое'),
              decoration: const InputDecoration(labelText:
'Категория'),
            ),
            const SizedBox(height: 16),
            TextField(
              controller: targetDaysCtrl,
              keyboardType: TextInputType.number,
              decoration: const InputDecoration(labelText:
'Цель (дней, опционально)'),
            ),
            const SizedBox(height: 16),
            Row(
              mainAxisAlignment:
MainAxisAlignment.spaceBetween,
              children: [
                const Text('Дата отказа:'),
                TextButton(
                  child:
Text(DateFormat('dd.MM.yyyy').format(selectedDate)),
                  onPressed: () async {
                    final date = await showDatePicker(
                      context: context,
                      initialDate: selectedDate,
                      firstDate: DateTime(2000),
                      lastDate: DateTime.now(),
                    );
                    if (date != null) {
                      setStateSB(() => selectedDate = date);
                    }
                  },
                ),
              ],
            ),
          ],
        ),
      ],
    ),
  ],

```

```

        ),
        actions: [
            TextButton(onPressed: () => Navigator.pop(ctx),
child: const Text('Отмена')),
            ElevatedButton(
                onPressed: () {
                    if (nameCtrl.text.isNotEmpty) {
                        int? targetDays =
int.tryParse(targetDaysCtrl.text);
                        Provider.of<AppProvider>(context, listen:
false)
                            .addHabit(nameCtrl.text, category,
selectedDate, targetDays: targetDays);
                        Navigator.pop(ctx);
                    }
                },
child: const Text('Создать'),
            ),
        ],
    );
}
),
);
}

```

```

@override
Widget build(BuildContext context) {
    final provider = Provider.of<AppProvider>(context);
    final habits = provider.habits;

    return Scaffold(
        appBar: AppBar(title: const Text('Трекер вредных
привычек')),
        floatingActionButton: FloatingActionButton(
            onPressed: () => _addHabitDialog(context),
            child: const Icon(Icons.add),
        ),
        body: habits.isEmpty
            ? const Center(child: Text('Нет отслеживаемых
привычек'))
            : ListView.builder(
                padding: const EdgeInsets.all(16),
                itemCount: habits.length,
                itemBuilder: (context, index) {
                    final habit = habits[index];
                    final prog = provider.habitProgresses[habit.id];

                    if (prog == null) return const SizedBox.shrink();

                    IconData icon;
                    switch (habit.category) {

```

```

        case 'Курение': icon = Icons.smoking_rooms;
break;

        case 'Алкоголь': icon = Icons.local_bar; break;
        case 'Сладкое': icon = Icons.cake; break;
        case 'Соцсети': icon = Icons.phone_android;
break;

        default: icon = Icons.block;
    }

    return Card(
        elevation: 2,
        margin: const EdgeInsets.only(bottom: 16),
        shape: RoundedRectangleBorder(borderRadius:
BorderRadius.circular(16)),
        child: Padding(
            padding: const EdgeInsets.all(16.0),
            child: Column(
                children: [
                    Row(
                        children: [
                            CircleAvatar(radius: 24, child:
Icon(icon, size: 28)),
                            const SizedBox(width: 16),
                            Expanded(
                                child: Column(
                                    crossAxisAlignment:
CrossAxisAlignment.start,
                                    children: [
                                        Text(habit.name, style: const
TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),
                                        Text(habit.category, style:
TextStyle(color: Colors.grey[600])),
                                    ],
                                ),
                            ),
                            IconButton(
                                icon: const Icon(Icons.delete,
color: Colors.red),
                                onPressed: () {
                                    showDialog(
                                        context: context,
                                        builder: (ctx) => AlertDialog(
                                            title: const Text('Удалить
привычку?'),
                                            content: const Text('Все
данные и прогресс будут удалены. Продолжить?'),
                                            actions: [
                                                TextButton(onPressed: () =>
Navigator.pop(ctx), child: const Text('Отмена')),
                                                ElevatedButton(

```

```

style:
ElevatedButton.styleFrom(backgroundColor: Colors.red,
foregroundColor: Colors.white),
onPressed: () {
provider.deleteHabit(habit.id!);
Navigator.pop(ctx);
},
child: const
Text('Удалить'),
)
],
),
);
},
),
],
),
const SizedBox(height: 16),
Row(
mainAxisAlignment:
MainAxisAlignment.spaceAround,
children: [
buildStatBadge('Текущая\нсерия',
'${prog.currentStreak} дн.', Colors.orange),
buildStatBadge('Рекорд',
'${prog.recordStreak} дн.', Colors.green),
],
),
if (habit.targetDays != null &&
habit.targetDays! > 0) ...[
const SizedBox(height: 16),
Column(
crossAxisAlignment:
CrossAxisAlignment.start,
children: [
Row(
mainAxisAlignment:
MainAxisAlignment.spaceBetween,
children: [
const Text('Прогресс цели:'),
Text('${prog.currentStreak} /
${habit.targetDays} дней', style: const TextStyle(fontWeight:
FontWeight.bold)),
],
),
const SizedBox(height: 8),
LinearProgressIndicator(
value: (prog.currentStreak /
habit.targetDays!).clamp(0.0, 1.0),
backgroundColor:
Colors.grey.shade300,

```

```

        color: Colors.green,
        minHeight: 8,
        borderRadius:
BorderRadius.circular(4),
    ),
  ],
)
],
const SizedBox(height: 16),
SizedBox(
  width: double.infinity,
  child: OutlinedButton(
    style: OutlinedButton.styleFrom(
      foregroundColor: Colors.red,
      side: const BorderSide(color:
Colors.red),
    ),
    onPressed: () {
      showDialog(
        context: context,
        builder: (ctx) => AlertDialog(
          title: const Text('Срыв?'),
          content: const Text('Вы
уверены, что хотите сбросить текущую серию? Рекорд будет
сохранен.'),
          actions: [
            TextButton(onPressed: () =>
Navigator.pop(ctx), child: const Text('Отмена')),
            ElevatedButton(
              style:
ElevatedButton.styleFrom(background-color: Colors.red,
foregroundColor: Colors.white),
              onPressed: () {
provider.resetHabit(habit.id!);
Navigator.pop(ctx);
ScaffoldMessenger.of(context).showSnackBar(
const SnackBar(content:
Text('Серия сброшена. Не сдавайтесь!')),
);
},
child: const
Text('Сбросить'),
),
),
),
),
child: const Text('С Б Р О С И Т Ь'),
),
),

```

```

    ],
    ),
  ),
);
},
),
);
}

Widget _buildStatBadge(String title, String value, Color color)
{
  return Column(
    children: [
      Text(title,      textAlign:      TextAlign.center,      style:
TextStyle(fontSize: 12, color: Colors.grey[600])),
      const SizedBox(height: 4),
      Container(
        padding: const  EdgeInsets.symmetric(horizontal: 16,
vertical: 8),
        decoration: BoxDecoration(
          color: color.withOpacity(0.2),
          borderRadius: BorderRadius.circular(12),
        ),
        child: Text(
          value,
          style:      TextStyle(fontSize:      18,      fontWeight:
FontWeight.bold, color: color),
        ),
      ),
    ],
  );
}
}

```

Файл achievements_screen.dart

```

import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../providers/app_provider.dart';
import '../helpers/achievements_definitions.dart';
import '../widgets/achievement_icon.dart';

class AchievementsScreen extends StatelessWidget {
  const AchievementsScreen({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    final provider = Provider.of<AppProvider>(context);
    final earnedIds = provider.earnedAchievements.map((e) =>
e.achievementId).toSet();
    final allDefs = AchievementsDefinitions.all;

    return Scaffold(

```

					ПГУ1.090301.43.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		141

```

appBar: AppBar(
  title: const Text('Достижения'),
  centerTitle: true,
),
body: CustomScrollView(
  slivers: [
    SliverToBoxAdapter(
      child: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Text(
          'Собрано:           ${earnedIds.length}           /
${allDefs.length}',
          style: const TextStyle(fontSize: 20, fontWeight:
FontWeight.bold),
          textAlign: TextAlign.center,
        ),
      ),
    ),
    SliverPadding(
      padding:      const      EdgeInsets.symmetric(horizontal:
16.0),
      sliver: SliverGrid(
        gridDelegate:                                const
SliverGridDelegateWithFixedCrossAxisCount(
          crossAxisCount: 3,
          crossAxisSpacing: 16,
          mainAxisSpacing: 16,
          childAspectRatio: 0.75,
        ),
        delegate: SliverChildBuilderDelegate(
          (context, index) {
            final def = allDefs[index];
            final isEarned = earnedIds.contains(def.id);
            return GestureDetector(
              onTap: () {
                showDialog(
                  context: context,
                  builder: (_) => AlertDialog(
                    title: Row(
                      children: [
                        AchievementIcon(
                          iconData: def.iconData,
                          iconColor: def.iconColor,
                          isEarned: isEarned,
                          size: 30,
                        ),
                        const SizedBox(width: 8),
                        Expanded(child:      Text(def.title,
style: const TextStyle(fontSize: 18))),
                      ],
                    ),
                    content: Text(def.description),

```

```

        actions: [
          TextButton(
            onPressed: () =>
Navigator.pop(context),
            child: const Text('OK'),
          )
        ],
      ),
    );
  },
  child: Column(
    mainAxisAlignment:
MainAxisAlignment.center,
    children: [
      AchievementIcon(
        iconData: def.iconData,
        iconColor: def.iconColor,
        isEarned: isEarned,
        size: 60,
      ),
      const SizedBox(height: 8),
      Text(
        def.title,
        textAlign: TextAlign.center,
        style: TextStyle(
          fontSize: 12,
          fontWeight: isEarned ?
FontWeight.bold : FontWeight.normal,
          color: isEarned ? null : Colors.grey,
        ),
        maxLines: 2,
        overflow: TextOverflow.ellipsis,
      ),
    ],
  ),
),
);
},
childCount: allDefs.length,
),
),
),
const SliverPadding(padding: EdgeInsets.only(bottom:
32)),
],
),
);
}
}

```

Файл charts_screen.dart

```

import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:fl_chart/fl_chart.dart';

```

```

import '../providers/app_provider.dart';
import '../models/app_models.dart';

class ChartsScreen extends StatelessWidget {
  const ChartsScreen({Key? key}) : super(key: key);

  List<FlSpot> _getWaterSpots(List<WaterIntake> intakes) {
    final now = DateTime.now();
    Map<int, int> daysToVol = {for (int i = 1; i <= 7; i++) i: 0};
    for (var w in intakes) {
      final date = DateTime.parse(w.timestamp);
      final diff = now.difference(date).inDays;
      if (diff <= 7 && diff >= 0) {
        daysToVol[7 - diff] = (daysToVol[7 - diff] ?? 0) +
w.volumeMl;
      }
    }
    List<FlSpot> spots = [];
    for (int i = 1; i <= 7; i++) {
      spots.add(FlSpot(i.toDouble(), (daysToVol[i] ??
0).toDouble()));
    }
    return spots;
  }

  List<FlSpot> _getCalorieSpots(List<CalorieIntake> intakes) {
    final now = DateTime.now();
    Map<int, int> daysToCal = {for (int i = 1; i <= 7; i++) i: 0};
    for (var c in intakes) {
      final date = DateTime.parse(c.timestamp);
      final diff = now.difference(date).inDays;
      if (diff <= 7 && diff >= 0) {
        daysToCal[7 - diff] = (daysToCal[7 - diff] ?? 0) +
c.calories;
      }
    }
    List<FlSpot> spots = [];
    for (int i = 1; i <= 7; i++) {
      spots.add(FlSpot(i.toDouble(), (daysToCal[i] ??
0).toDouble()));
    }
    return spots;
  }

  List<PieChartSectionData>
_getMealPieSections(List<CalorieIntake> intakes) {
    if (intakes.isEmpty) return [];
    Map<String, int> counts = {};
    for (var c in intakes) {
      counts[c.mealType] = (counts[c.mealType] ?? 0) + 1;
    }
  }

```

```

final colors = {
    'Завтрак': Colors.orangeAccent,
    'Обед': Colors.yellow,
    'Ужин': Colors.deepOrange,
    'Перекус': Colors.red,
};

return counts.entries.map((e) {
    return PieChartSectionData(
        value: e.value.toDouble(),
        title: '${e.key}\n(${e.value}}',
        color: colors[e.key] ?? Colors.grey,
        radius: 60,
        titleStyle: const TextStyle(fontSize: 12, fontWeight:
FontWeight.bold, color: Colors.black),
    );
}).toList();
}

@override
Widget build(BuildContext context) {
    final provider = Provider.of<AppProvider>(context);

    return Scaffold(
        appBar: AppBar(title: const Text('Графики (7 дней)'),
        body: ListView(
            padding: const EdgeInsets.all(16),
            children: [
                const Text('Потребление воды', style:
TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),
                const SizedBox(height: 16),
                SizedBox(
                    height: 200,
                    child: LineChart(
                        LineChartData(
                            minY: 0,
                            titlesData: const FlTitlesData(
                                rightTitles: AxisTitles(sideTitles:
SideTitles(showTitles: false)),
                                topTitles: AxisTitles(sideTitles:
SideTitles(showTitles: false)),
                            ),
                            lineBarsData: [
                                LineChartBarData(
                                    spots: _getWaterSpots(provider.waterIntakes),
                                    isCurved: true,
                                    color: Colors.blue,
                                    barWidth: 3,
                                    belowBarData: BarAreaData(show: true, color:
Colors.blue.withOpacity(0.3)),
                                )
                            ],
                        )
                    ],
                )
            ],
        ),
    );
}

```

```

        ),
    ),
),

const Divider(height: 48),

const Text('Калории', style: TextStyle(fontSize: 18,
fontWeight: FontWeight.bold)),
const SizedBox(height: 16),
SizedBox(
    height: 200,
    child: LineChart(
        LineChartData(
            minY: 0,
            titlesData: const FlTitlesData(
                rightTitles: AxisTitles(sideTitles:
SideTitles(showTitles: false)),
                topTitles: AxisTitles(sideTitles:
SideTitles(showTitles: false)),
            ),
            lineBarsData: [
                LineChartBarData(
                    spots:
_getCalorieSpots(provider.calorieIntakes),
                    isCurved: true,
                    color: Colors.orange,
                    barWidth: 3,
                    belowBarData: BarAreaData(show: true, color:
Colors.orange.withOpacity(0.3)),
                )
            ],
        ),
    ),
),

const Divider(height: 48),

const Text('Распределение приёмов пищи', style:
TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),
const SizedBox(height: 16),
SizedBox(
    height: 200,
    child: provider.calorieIntakes.isEmpty
        ? const Center(child: Text('Нет данных'))
        : PieChart(
            PieChartData(
                sectionsSpace: 2,
                centerSpaceRadius: 40,
                sections:
_getMealPieSections(provider.calorieIntakes),
            ),
        ),
),

```

```

    ),
  ],
),
);
}
}

```

Файл settings_screen.dart

```

import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../providers/app_provider.dart';
import '../services/export_service.dart';

class SettingsScreen extends StatefulWidget {
  const SettingsScreen({Key? key}) : super(key: key);

  @override
  State<SettingsScreen> createState() => _SettingsScreenState();
}

class _SettingsScreenState extends State<SettingsScreen> {
  late TextEditingController _waterController;
  late TextEditingController _caloriesController;
  late TextEditingController _sleepController;

  @override
  void initState() {
    super.initState();
    final settings = Provider.of<AppProvider>(context, listen:
false).settings;
    _waterController = TextEditingController(text:
settings?.dailyWaterMl.toString() ?? '2500');
    _caloriesController = TextEditingController(text:
settings?.dailyCalories.toString() ?? '2500');
    _sleepController = TextEditingController(text:
settings?.targetSleepHours.toString() ?? '8.0');
  }

  @override
  void dispose() {
    _waterController.dispose();
    _caloriesController.dispose();
    _sleepController.dispose();
    super.dispose();
  }

  void _saveGoals(AppProvider provider) {
    if (_waterController.text.isNotEmpty) {
      int w = int.tryParse(_waterController.text) ?? 2500;
      if (w < 500) w = 500;
      if (w > 10000) w = 10000;
      _waterController.text = w.toString();
      provider.updateWaterGoal(w);
    }
  }
}

```

```

    }
    if (_caloriesController.text.isNotEmpty) {
        int c = int.tryParse(_caloriesController.text) ?? 2500;
        if (c < 500) c = 500;
        if (c > 10000) c = 10000;
        _caloriesController.text = c.toString();
        provider.updateCalorieGoal(c);
    }
    if (_sleepController.text.isNotEmpty) {
        double s = double.tryParse(_sleepController.text) ?? 8.0;
        if (s <= 0) s = 1.0;
        if (s > 100.0) s = 100.0;
        _sleepController.text = s.toString();
        provider.updateSleepGoal(s);
    }
    ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Цели сохранены (значения скорректированы в допустимых пределах)'),
    );
}

@override
Widget build(BuildContext context) {
    final provider = Provider.of<AppProvider>(context);
    final settings = provider.settings;

    if (settings == null) {
        return Scaffold(
            appBar: AppBar(title: const Text('Настройки')),
            body: const Center(child: Text('Загрузка настроек...')),
        );
    }

    return Scaffold(
        appBar: AppBar(
            title: const Text('Настройки'),
        ),
        body: ListView(
            padding: const EdgeInsets.all(16.0),
            children: [
                const Text('Цели', style: TextStyle(fontSize: 20,
fontWeight: FontWeight.bold)),
                const SizedBox(height: 16),
                TextField(
                    controller: _waterController,
                    keyboardType: TextInputType.number,
                    decoration: const InputDecoration(
                        labelText: 'Норма воды (мл)',
                        border: OutlineInputBorder(),
                        prefixIcon: Icon(Icons.water_drop),
                    ),
                ),
            ],
        ),
    );
}

```

```

const SizedBox(height: 16),
TextField(
  controller: _caloriesController,
  keyboardType: TextInputType.number,
  decoration: const InputDecoration(
    labelText: 'Норма калорий (ккал)',
    border: OutlineInputBorder(),
    prefixIcon: Icon(Icons.local_dining),
  ),
),
const SizedBox(height: 16),
TextField(
  controller: _sleepController,
  keyboardType: TextInputType.number,
  decoration: const InputDecoration(
    labelText: 'Цель сна (часы)',
    border: OutlineInputBorder(),
    prefixIcon: Icon(Icons.bedtime),
  ),
),
const SizedBox(height: 16),
ElevatedButton(
  onPressed: () => _saveGoals(provider),
  child: const Text('Сохранить цели'),
),
const Divider(height: 32),

const Text('Внешний вид', style: TextStyle(fontSize: 20,
fontWeight: FontWeight.bold)),
ListTile(
  title: const Text('Тема оформления'),
  trailing: DropdownButton<String>(
    value: settings.themeMode,
    items: const [
      DropdownMenuItem(value: 'system', child:
Text('Системная')),
      DropdownMenuItem(value: 'light', child:
Text('Светлая')),
      DropdownMenuItem(value: 'dark', child:
Text('Тёмная')),
    ],
    onChanged: (val) {
      if (val != null) provider.updateThemeMode(val);
    },
  ),
),
ListTile(
  title: const Text('Язык'),
  trailing: DropdownButton<String>(
    value: settings.language,
    items: const [

```

```

        DropdownMenuItem(value: 'ru', child:
Text('Русский')),
        DropdownMenuItem(value: 'en', child:
Text('English')),
    ],
    onChanged: (val) {
        if (val != null) provider.updateLanguage(val);
    },
),
),

const Divider(height: 32),

const Text('Уведомления', style: TextStyle(fontSize:
20, fontWeight: FontWeight.bold)),
SwitchListTile(
    title: const Text('Включить напоминания'),
    subtitle: const Text('Напоминать о воде, еде и сне'),
    value: settings.enableNotifications,
    onChanged: (val) {
        provider.updateEnableNotifications(val);
    },
),

const Divider(height: 32),

const Text('Данные', style: TextStyle(fontSize: 20,
fontWeight: FontWeight.bold)),
ListTile(
    leading: const Icon(Icons.backup),
    title: const Text('Создать резервную копию'),
    subtitle: const Text('Сохранить базу данных'),
    onTap: () async {
        String? res = await provider.backupDatabase();
        if (res != null && mounted) {

ScaffoldMessenger.of(context).showSnackBar(SnackBar(content:
Text(res)));
        }
    },
),
ListTile(
    leading: const Icon(Icons.restore),
    title: const Text('Восстановить из копии'),
    subtitle: const Text('Заменить текущую БД на
сохранённую'),
    onTap: () async {
        String? res = await provider.restoreDatabase();
        if (res != null && mounted) {

ScaffoldMessenger.of(context).showSnackBar(SnackBar(content:
Text(res)));
        }
    },
),

```

```

        },
      ),
    ListTile(
      leading: const Icon(Icons.file_download),
      title: const Text('Экспорт в CSV'),
      subtitle: const Text('Сохранить данные всех
трекеров'),
      onTap: () async {
        String? res = await ExportService.exportToCSV();
        if (res != null && mounted) {

ScaffoldMessenger.of(context).showSnackBar(SnackBar(content:
Text(res)));
        }
      },
    ),
    ListTile(
      leading: const Icon(Icons.picture_as_pdf),
      title: const Text('Отчёт в PDF'),
      subtitle: const Text('Сгенерировать сводный отчёт'),
      onTap: () async {
        if (provider.currentUser == null) return;
        String? res = await
ExportService.exportToPDF(provider.currentUser!.id!);
        if (res != null && mounted) {

ScaffoldMessenger.of(context).showSnackBar(SnackBar(content:
Text(res)));
        }
      },
    ),
  ],
),
);
}
}

```

Файл achievement_icon.dart

```

import 'package:flutter/material.dart';

class AchievementIcon extends StatelessWidget {
  final IconData iconData;
  final Color iconColor;
  final bool isEarned;
  final double size;

  const AchievementIcon({
    Key? key,
    required this.iconData,
    required this.iconColor,
    this.isEarned = false,

```

```

        this.size = 50.0,
    }) : super(key: key);

@override
Widget build(BuildContext context) {
    if (!isEarned) {
        return CustomPaint(
            size: Size(size, size),
            painter: HexagonPainter(color: Colors.grey.shade300),
            child: SizedBox(
                width: size,
                height: size,
                child: Icon(
                    Icons.lock_outline,
                    color: Colors.grey.shade600,
                    size: size * 0.5,
                ),
            ),
        );
    }

    return CustomPaint(
        size: Size(size, size),
        painter: HexagonPainter(color: iconColor.withOpacity(0.2)),
        child: SizedBox(
            width: size,
            height: size,
            child: Icon(
                iconData,
                color: iconColor,
                size: size * 0.6,
            ),
        ),
    );
}

}

class HexagonPainter extends CustomPainter {
    final Color color;

    HexagonPainter({required this.color});

    @override
    void paint(Canvas canvas, Size size) {
        Paint paint = Paint()..color = color..style =
        PaintingStyle.fill;
        Path path = Path();

        final side = size.width / 2;
        path.moveTo(side * 0.3, 0);
        path.lineTo(size.width - side * 0.3, 0);
        path.lineTo(size.width, side * 0.3);
    }
}

```

```

        path.lineTo(size.width, size.height - side * 0.3);
        path.lineTo(size.width - side * 0.3, size.height);
        path.lineTo(side * 0.3, size.height);
        path.lineTo(0, size.height - side * 0.3);
        path.lineTo(0, side * 0.3);
        path.close();

        canvas.drawPath(path, paint);
    }

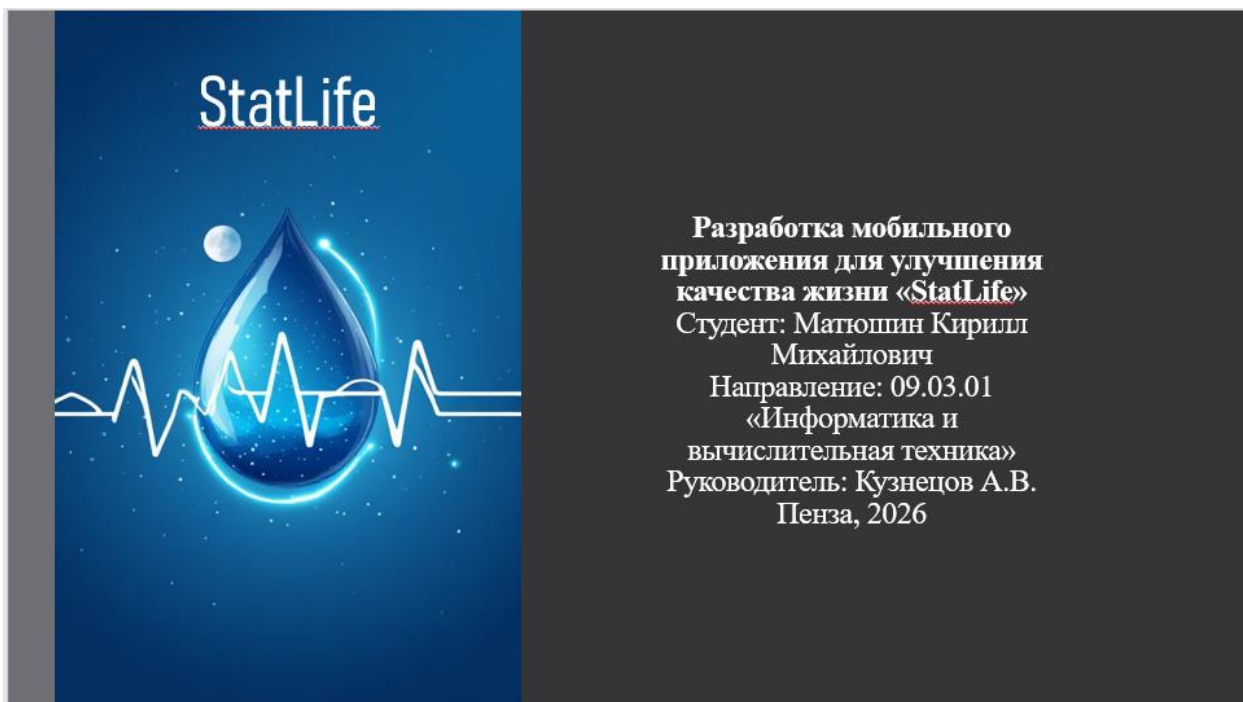
    @override
    bool shouldRepaint(covariant CustomPainter oldDelegate) {
        return false;
    }
}

```

Приложение В

«Веб-приложение фитнес-центра»

Презентация

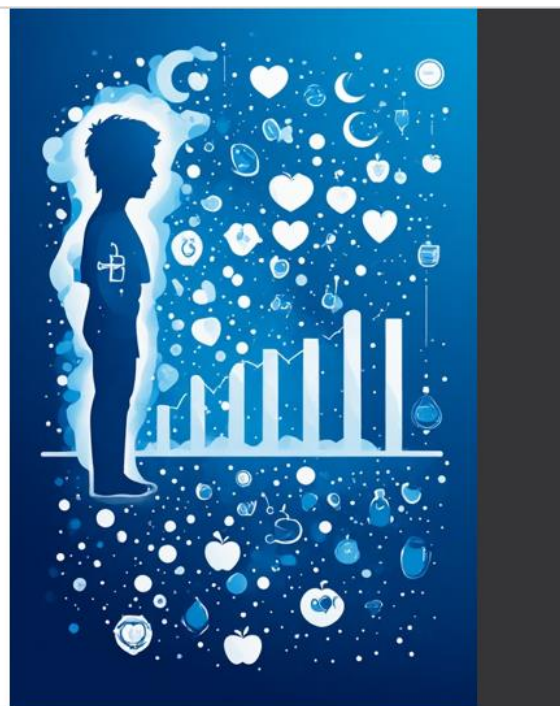


> По данным ВОЗ, 71% смертей в мире приходится на неинфекционные заболевания, связанные с образом жизни

> До 80% случаев инфарктов и инсультов можно предотвратить за счёт коррекции питания, водного баланса, сна и отказа от вредных привычек

> Рынок mHealth-приложений в 2025 году превысил 75 млрд долларов

> Существующие аналоги узкоспециализированы (только вода, только питание или только сон) и требуют платной подписки

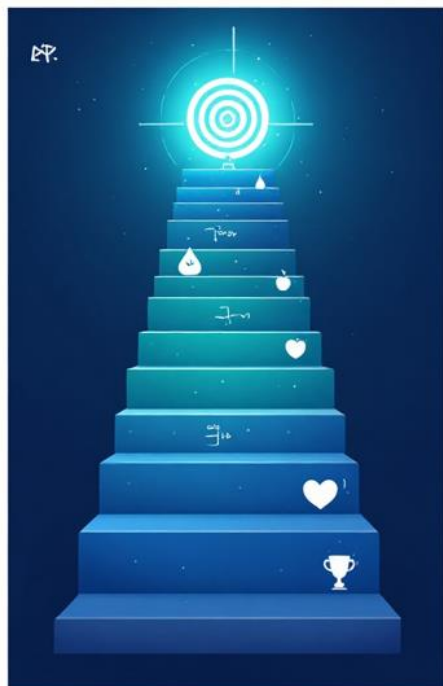


Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.43.001 ПЗ

Лист

154



Цель:
разработка кроссплатформенного мобильного приложения для комплексного контроля качества жизни.

Задачи:
Анализ аналогов и требований

Выбор технологий (Flutter, Dart, SQLite, Provider)

Реализация трекеров воды, питания, сна, веса, привычек

Реализация системы достижений (100 наград)

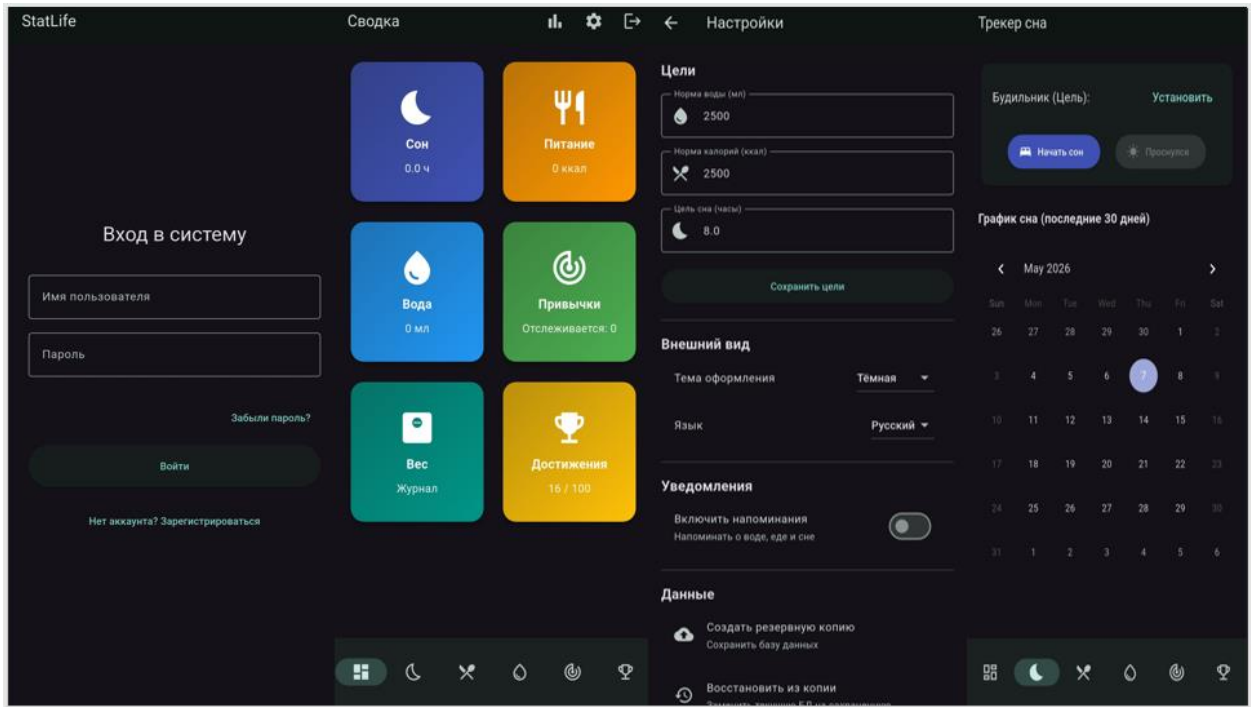
Реализация визуализации данных (графикки, календари)

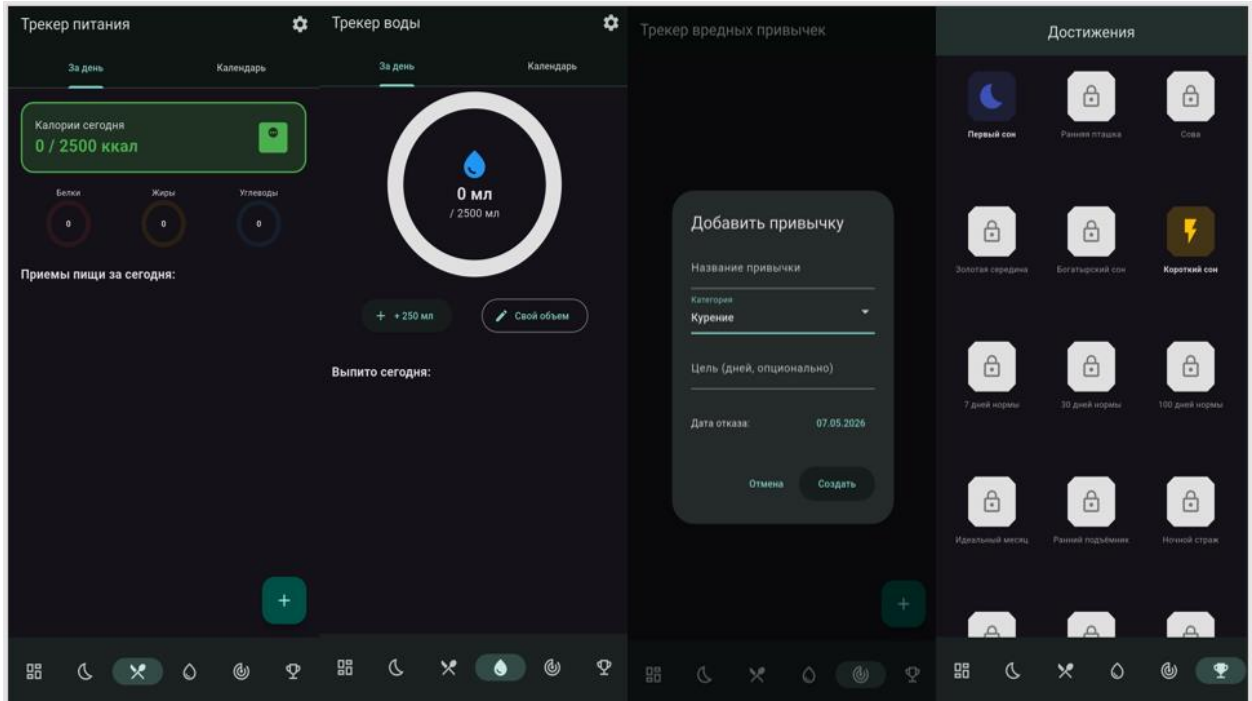
Реализация экспорта в CSV/PDF и резервного копирования

MyFitnessPal	Нет <u>трекера</u> воды и сна, нет привычек, платная подписка
<u>WaterMinder</u>	Нет питания, сна, привычек, нет игрофикации
Sleep Cycle	Нет воды, питания, привычек, платная подписка



Тип архитектуры: локальная (без сервера), MVVM + <u>Provider</u>		
Компонент	Технология	Что делает
View	Flutter виджеты	Отображает интерфейс, передаёт действия пользователя
<u>ViewModel</u>	<u>AppProvider</u> + <u>ChangeNotifier</u>	Хранит состояние, бизнес-логику, уведомляет UI об изменениях
Model	<u>DatabaseHelper</u> + <u>SQLite</u>	Хранит данные, выполняет CRUD-операции
Настройки	<u>SharedPreferences</u>	Хранит статус авторизации, тему, язык, активную сессию сна
Уведомления	<u>flutter local notifications</u>	Напоминания о воде, еде, будильник
Экспорт	pdf + csv + <u>file_picker</u>	Выгрузка данных в CSV и PDF, резервное копирование





Разработано:

Кроссплатформенное мобильное приложение «StatLife»

6 трекеров (вода, питание, сон, вес, привычки, достижения)

100 уникальных достижений

Визуализация данных (графики, календари, диаграммы)

Экспорт в CSV и PDF

Резервное копирование и восстановление БД



Практическая значимость:

- Комплексный контроль здоровья в одном приложении
- Полностью бесплатно и автономно (без интернета)
- Мотивация через геймификацию

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.43.001 ПЗ

Лист

157