

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление

09.03.01 «Информатика и вычислительная техника»

подготовки:

Профиль подготовки:

«Вычислительные машины, комплексы, системы и
сети»

БАКАЛАВРСКАЯ РАБОТА

на тему

**МОБИЛЬНОЕ ПРИЛОЖЕНИЕ ДЛЯ УЧЕТА
УСПЕВАЕМОСТИ ОБУЧАЮЩИХСЯ**

Студент

31.05.2022

(подпись, дата)

Семенов Андрей Сергеевич
(ФИО полностью)

Руководитель

31.05.2022

(подпись, дата)

М.П. Синев

(фамилия, инициалы)

Нормоконтролёр

31.05.2022

(подпись, дата)

А.С. Бычков

(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 02.06.2022 г. № 13)

Заведующий кафедрой

(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой

07.06.2022 № 4

отлично

(протокол заседания ГЭК от

Секретарь ГЭК

(подпись)

Е.И. Гурин
(фамилия, инициалы)

Пенза 2022

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

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«26» 04 2022г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Семенов Андрей Сергеевич гр. 18BV1 факультета вычислительной техники

2. Тема работы Мобильное приложение для учета успеваемости обучающихся

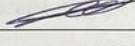
Тема утверждена приказом ПГУ № 357/о от "26" апреля 2022 г.

3. Руководитель бакалаврской работы Синев М.П., к.т.н., доцент кафедры ВТ

4. Задание на работу (назначение разработки, исходные данные и т.п.)
Разработать и отладить мобильное приложение, которое позволяет вести учет успеваемости обучающихся.

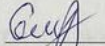
5. Перечень подлежащих разработке вопросов
Требуется произвести анализ актуальности проекта, потребностей целевых пользователей, разрабатываемого приложения, аналогов, современных мобильных операционных систем, инструментов мобильной разработки (фреймворков, сред программирования).

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ требований	9	26.04.2022	
2.	Анализ актуальности проекат	9	27.04.2022	
3.	Анализ потребностей пользователей	18	28.04.2022-29.04.2022	
4.	Анализ аналогов	18	30.04.2022-01.05.2022	
5.	Анализ современных мобильных операционных систем	18	02.05.2022-03.05.2022	
6.	Выбор инструментов разработки	18	04.05.2022-05.05.2022	
7.	Разработка структуры приложения	45	06.05.2022-10.05.2022	
8.	Разработка мобильного приложения для учета успеваемости обучающихся	90	11.05.2022-20.05.2022	
9.	Тестирование работоспособности приложения	18	21.05.2022 - 22.05.2022	
10.	Оформление пояснительной записки	81	23.05.2022-31.05.2022	

Дата выдачи задания "26" апреля 2022г.

Руководитель бакалаврской работы  26.04.2022 Синева М.П.
(подпись, дата)

Задание к исполнению принял студент  26.04.2022 Семенов А.С.
(подпись, дата)

Работу к защите допустить
Декан факультета  6.06.2022 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 101 лист, 12 рисунков, 1 таблицу, 8 источников, 3 приложения.

КРОСПЛАТФОРМЕННАЯ РАЗРАБОТКА, БД, REST, API, DART, FLUTTER, BLOC.

Объектом разработки является мобильное приложение, которое позволяет преподавателям вести учет успеваемости обучающихся.

Цель работы – разработать мобильное приложение для учета успеваемости студентов. Приложение должно обладать следующим функционалом: пользователь приложения должен иметь возможность просматривать список экзаменов, которые ему необходимо провести в текущем семестре, просматривать текущие оценки студентов за экзамен, вести учет успеваемости студентов, а именно выставлять оценки за экзамен или зачет.

В процессе работы проводился анализ актуальности проекта, потребностей целевых пользователей, аналогов, мобильных операционных систем, способов разработки мобильного ПО.

В результате проведенной работы было разработано кроссплатформенное мобильное приложение для ведения учета успеваемости студентов. Приложение было подготовлено для взаимодействия с сервером.

					ПГУ 1.090301.38.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата	Лит.	Лист	Листов
Разраб.		Семенов А.С.				2	101
Провер.		Синев М.П.			ФВТ, каф. ВТ гр. 18ВВ1		
Н. Контр		Бычков А.С.					
Утверд.		Митрохин М.А.					

Мобильное приложение для учета успеваемости обучающихся
Пояснительная записка

Содержание

Перечень сокращений и обозначений.....	4
Введение.....	5
1 Мобильное приложение для учета успеваемости обучающихся.....	7
1.1 Актуальность проекта.....	7
1.2 Анализ потребностей пользователей	8
1.3 Анализ аналогов	9
1.4 Анализ современных операционных систем.....	11
1.5 Постановка задачи.....	13
1.6 Выбор инструментов разработки	14
1.7 Структура приложения	20
1.8 Реализация	24
1.9 Руководство пользователя.....	40
2 Охрана труда.....	45
Заключение	47
Список используемых источников.....	49
Приложение А- UML- диаграммы.....	51
Приложение Б- Листинг программы.....	56
Приложение В- Презентация	92

Перечень сокращений и обозначений

API (Application Programming Interface) – программный интерфейс взаимодействия с программой

BLoC (Buisness Logic Component) – архитектура представления

CPU (Central Processing Unit) – центральный процессор

FPS (Frame Per Second) – частота кадров в секунду

GPU (Graphic Processing Unit) – графический процессор

ID (Identifier) – идентификатор объекта

REST (Representational State Transfer) – передача состояния представления

SQL (Structured Query Language) – язык структурированных запросов

UML (Unified Modeling Language) – унифицированный язык моделирования

URL (Uniform Resource Locator– единообразный локатор ресурса

XML (eXtensible Markup Language) – расширяемый язык разметки

БД – База данных

ОС (операционная система) – комплекс программ, образующих некоторую систему

ПЭВМ (персональная электронно-вычислительная машина) – настольная микро-ЭВМ

ПК- персональный компьютер;

ПО (программное обеспечение) – набор программ

ЭИОС (электронная информационно-образовательная среда) – совокупность информационных, телекоммуникационных технологий, электронных информационных и образовательных ресурсов, обеспечивающих условия освоения обучающимися образовательных программ в полном объеме независимо от их места нахождения.

Введение

Мобильные устройства давно являются неотъемлемой частью жизни современного человека. Современные модели позволяют решать огромный спектр задач, благодаря чему стали заменой множеству устройств ранее необходимых. Это возможно благодаря различному программному обеспечению и легкости его распространения с помощью встроенных магазинов приложений.

Современное устройства уже с завода предоставляет множество базовых функций (начиная с осуществлений звонков, заканчивая воспроизведением различных аудио, видео файлов). Приложения, разрабатываемые сторонними разработчиками направлены на расширение возможностей устройства, в основном на упрощение и повышение мобильности каких-то процессов, выполняемых человеком. Как пример можно привести приложение любого банка, которое позволяет выполнить множество операций со своими финансами, в любой точке мира, а не только в здании банка, как было ранее. Пользователю нужно только современное устройство и доступ к сети интернет.

Целями данного проекта является:

- изучение современных инструментов для создания мобильных приложений;
- изучение компонентов для создания мобильного ПО;
- приложение- мобильный клиент преподавателя, которое позволит оценивать успеваемость студентов с помощью мобильного устройства.

Для достижения поставленных целей необходимо:

- провести анализ актуальности проекта;
- провести анализ потребностей пользователей;
- провести анализ аналогов;
- провести анализ современных мобильных операционных систем;

					ПГУ 1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		5

- провести анализ современных инструментов мобильной разработки;
- продумать структуру приложения;
- разработать кодовую базу приложения;
- протестировать работу созданного приложения.

					ПГУ 1.090301.38.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

1 Мобильное приложение для учета успеваемости обучающихся

1.1 Актуальность проекта

Мобильные устройства плотно интегрировались в жизнь современного человека. Все больше людей предпочитают их полноразмерным компьютерам по той причине, что возможности современных смартфонов и планшетов полностью удовлетворяют их потребности. К тому же данная индустрия постоянно развивается, приближая мощности таких устройств к мощности полноценных ПК. Потому все производители информационных продуктов стремятся адаптировать их под мобильные устройства для того, чтобы продуктом пользовались в любой необходимый момент. Данный проект преследует ту же цель, а именно цель повышения мобильности процесса учета успеваемости студентов.

В связи с ограничениями, которые были установлены из-за пандемии *COVID-19*, подавляющее большинство высших учебных заведений России было вынуждено перейти на дистанционный формат обучения. В соответствии с приказом министерства образования и науки Российской Федерации от 23.08.2017 № 816 "Об утверждении Порядка применения организациями осуществляющими образовательную деятельность, электронного обучения, дистанционных образовательных технологий при реализации образовательных программ", а именно в соответствии с шестым пунктом, прилагаемого порядка, если учебная организация предоставляет полностью дистанционные услуги обучения, она обязана обеспечить функционирование электронной информационно-образовательной среды [1].

Электронная информационно-образовательная среда (ЭИОС) – это совокупность информационных, телекоммуникационных технологий, электронных информационных и образовательных ресурсов, обеспечивающих

					ПГУ 1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		7

условия освоения обучающимися образовательных программ в полном объеме независимо от их места нахождения.

ЭИОС как правило подразумевает наличие доступа к некоторому электронному журналу, где преподаватель может выставить оценку за экзамен или зачет. Данный проект предполагает интеграцию мобильного приложения с API ЭИОС учебного заведения, благодаря чему повысится мобильность процесса учета успеваемости студентов.

1.2 Анализ потребностей пользователей

Целевой аудиторией данного проекта являются преподаватели университета. На данный момент для учета успеваемости (в электронном виде) нужен персональный компьютер. Можно использовать мобильное устройство, но графический интерфейс сервиса, используемого университетом, плохо адаптирован под него. Во время проведения зачетной или экзаменационной сессии условие наличия ПК становится проблемой. Не все аудитории оборудованы компьютерами, и не каждый преподаватель имеет желание или возможность носить с собой ноутбук. В связи с этим алгоритм учета успеваемости студента включает в себя ряд ненужных действий. После принятия экзамена, преподавателю приходится сохранять оценку ответа студента на каком-либо носителе (чаще всего на бумажном), затем необходимо хранить этот носитель до того момента, как преподаватель получит доступ к ПК и после дублировать информацию. Данные действия создают риски:

- потери информационного носителя с оценками;
- допущения ошибок при выставлении оценок;
- возможность подделки оценки без ведома преподавателя.

Данный проект должен решить вышеперечисленные проблемы. Большинство преподавателей имеют мобильные устройства под управлением

операционной системы *Android* или *IOS*, которое всегда находится у них под рукой. Нет никаких факторов, мешающих использовать его во время экзамена. Приложение, разрабатываемое в рамках данного проекта, позволит поставить оценку сразу после ответа студента, исключая промежуточные носители информации.

1.3 Анализ аналогов

В рамках работы над проектом были рассмотрены некоторые его аналоги, распространяющиеся в магазинах приложений “*Google Play Market*” и “*App Store*”.

Одним из самых популярный приложений в *Google Play Market* является “Журнал преподавателя” (более 50 тысяч скачиваний). Снимок экрана данного приложения представлен на рисунке 1.

SJournal											
Группа		Дисциплина						Вид занятия			
АПЭМА Автоматизированное проектирование ЭМА		МНЭМА Монтаж и настройка ЭМА		МП Микропроцессорные устройства		НДЭМА Надежность и диагностика ЭМА		ОППЭ Организация, планирование и проведение эксперимента		ТОЭ 3ч Теоретические основы электротехники	
ТРЭМА Тепловые расчеты в ЭМА		Добавить									
АТ-11-1 ТОЭ 3ч - Теоретические основы электротехники ЛБ											
№	Ф.И.О.	ЛБ 1 24.09.2013	ЛБ 2 01.10.2013	Захист ЛБ 1 08.10.2013	Захист ЛБ 2 15.10.2013	ЛБ 3 22.10.2013	ЛБ 4 29.10.2013	ЛБ 5.1 05.11.2013	ЛБ 5.2 12.11.2013	ЛБ 5 19.11.2013	ЛБ 6 26.11.2013
1	Белый Л.Г.			4	4						
2	Богданов С.С.			4	4	н	н	н			н
3	Буц В.В.	н	н	4	4	н	н	н	н	н	н
4	Волошин А.Ю.		н	4	4		н		н	н	н
5	Иванов И.И.		н	4	4				н	н	
6	Крамарев В.С.	н	н	4	4	н	н	н	н	н	н
7	Кулинич А.Ю.			4	4						
8	Петров Д.П.			4	4	6	н		н		
9	Потапов Ю.В.			4	4	н	н	н			н

Рисунок 1- приложение “Журнал преподавателя”

Проанализировав данное приложения были выделены следующие преимущества и недостатки:

Преимуществом является богатый функционал. Приложение предоставляет множество возможностей для работы с журналом: добавление новых дисциплин, групп, видов занятий, настройка системы оценивания, формата отображения занятий, формат отображения имени студента, символа, обозначающего отсутствие студента.

Недостатки:

- данное приложение не работает с сервером. Это значит, что все данные пользователю приходится вводить вручную. Нет возможности загрузить данные о группах, дисциплинах, оценках обучающихся. Так же это говорит о том, что все данные будут храниться на устройстве пользователя и не будет осуществляться их репликация на другие устройства. По сути, это приложение является промежуточным носителем, проблемы которого были указаны выше;

- устаревший, примитивный интерфейс;

- не интуитивно понятный интерфейс. Разработчики приложения прилагают к нему видео-инструкцию, потому что без нее невозможно разобраться как работает приложение. Данный факт сильно портит опыт использования приложения, что приводит к потере интереса конечного потребителя.

На площадке *App Store* крайне мало подобных приложений. Удалось найти лишь приложения, предназначенные для школьных учителей, в которых они могут вести электронный журнал. Одним из самых популярных подобных приложений является “Журнал Дневник.ру”. Снимок экрана данного приложения представлен на рисунке 2.

Проанализировав данное приложение, были выявлены следующие преимущества и недостатки:

Преимущества:

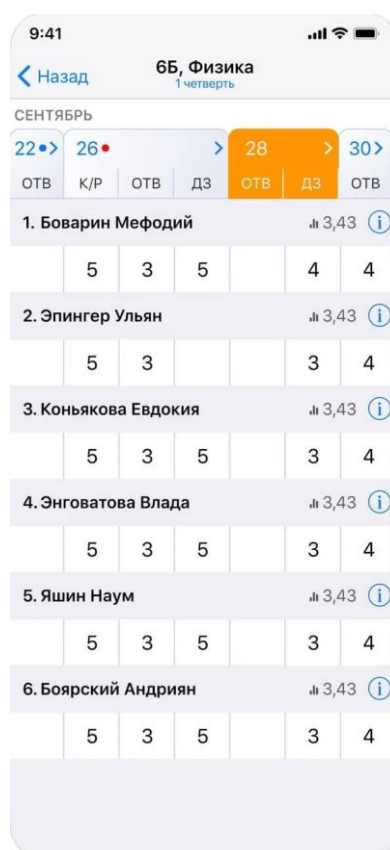


Рисунок 2- Приложение “Журнал Дневник.ру”

- работа с сервером. Приложение хранит минимум данных на устройстве пользователя;

- современный, интуитивно понятный интерфейс.

На основе отзывов пользователей, у этого приложения единственным недостатком является нестабильная работа.

1.4 Анализ современных операционных систем

За долгие годы развития мобильных устройств лишь две операционные системы смогли остаться на плаву и занять лидирующие позиции. Речь идет об *IOS* и *Android*. Согласно статистике, наибольшая доля мирового рынка мобильных операционных систем принадлежит *Android*, она составляет 71.09%. И лишь 28.21% принадлежит *IOS*.

1.4.1 Пожалуй главной причиной, по которой *Android* является самой распространенной мобильной операционной системой в мире, есть тот факт, что он является открытым проектом, что позволяет множеству производителей мобильных устройств дорабатывать и устанавливать *Android* на свои устройства. Также открытость данной операционной системы позволяет сторонним разработчикам создавать различное полезное программное обеспечение для устройств на *Android* [2]. В этих весомых преимуществах кроются не менее весомые недостатки данной ОС: не все производители добросовестно относятся к интеграции *Android* в производимые ими гаджеты, что приводит к снижению производительности, а свобода в создании ПО позволяет злоумышленникам создавать различные вирусные программы, что снижает степень безопасности устройства. Также стоит отметить, что все приложения, которые попадают в официальный магазин приложений *Google Play Market* обязательно проходят сертификацию, но не самую тщательную в сравнении с конкурентами, что также влияет на безопасность устройства.

1.4.2 *IOS*- это вторая по популярности мобильная операционная система в мире. Это звание она получила во многом благодаря своей безопасности и стабильности. Так как *IOS* разрабатывается исключительно под конкретный набор устройств случаи возникновения проблем с производительностью возникают крайне редко и устраняются максимально оперативно. Данная операционная система не является проектом с открытым исходным кодом, потому исключено вмешательство в нее сторонних разработчиков. Все приложения устанавливаются только из официального магазина приложений *App Store*, где они проходят серьезную проверку перед размещением на площадке. Согласно перечисленным ранее факторам *IOS* можно назвать самой безопасной мобильной операционной системой, ведь риск заражения устройства вредоносным ПО стремиться к нулю. В этом преимуществе также

кроется недостаток. В сравнении с *Android* операционная система *IOS* обладает скудным набором полезного стороннего ПО, которое к тому же зачастую стоит немалых денег.

Каждая из приведенных выше мобильных операционных систем обладает своими преимуществами и недостатками, поэтому сложно выделить абсолютного фаворита. Также обе ОС имеют огромную базу преданных пользователей.

1.5 Постановка задачи

Главной целью данного проекта является разработка приложения-мобильного клиента преподавателя, которое позволит оценивать успеваемость студентов с помощью мобильного устройства.

На основе исследования современных мобильных операционных систем можно сделать вывод, что приложение должно работать, как на *IOS*, так и на *Android*. Для достижения поставленной цели можно прибегнуть к способу разработки двух отдельных приложений для каждой ОС, или к способу кроссплатформенной разработки.

Каждый из способов имеет свои преимущества и недостатки. Главным преимуществом разработки приложения под определенную операционную систему является производительность. Ни одно кроссплатформенное решение не может соревноваться в производительности с данным способом разработки. Основным преимуществом кроссплатформенной разработки является минимальная трудозатратность, так как данный способ разработки предполагает создание единой кодовой базы для каждой операционной системы.

Главное преимущество кроссплатформенной разработки является решающим в рамках разработки данного проекта, поэтому именно он был выбран.

Анализ потребностей целевых пользователей разрабатываемого приложения показал, что для их удовлетворения оно должно обладать следующим функционалом:

- пользователь приложения должен иметь возможность просматривать список экзаменов, которые ему необходимо провести в текущем семестре;
- пользователь приложения должен иметь возможность просматривать текущие оценки студентов за экзамен;
- пользователь приложения должен иметь возможность вести учет успеваемости студентов, а именно выставлять оценки за экзамен или зачет.

Наглядно функционал приложения представлен на *UML*- диаграмме прецедентов, которая представлена на рисунке 3.

После исследования аналогов был сделан вывод, что для составления конкуренции уже имеющимся решениям разрабатываемое приложение должно обладать следующими характеристиками:

- стабильная работа;
- современный, интуитивно понятный интерфейс;
- возможность взаимодействия с сервером.

Взаимодействие с сервером предполагает интеграцию с *API* электронно информационно-образовательной среды. На данный момент это не является возможным, потому должен быть подготовлен специальный класс, который имитирует работу с сервером.

1.6 Выбор инструментов разработки

1.6.1 Изначально необходимо выбрать фреймворк для кроссплатформенной мобильной разработки. За последние годы наиболее популярными фреймворками для кроссплатформенной мобильной разработки стали *Flutter* и *React Native*. Их сравнение будет проходить по следующим критериям: производительность, безопасность.

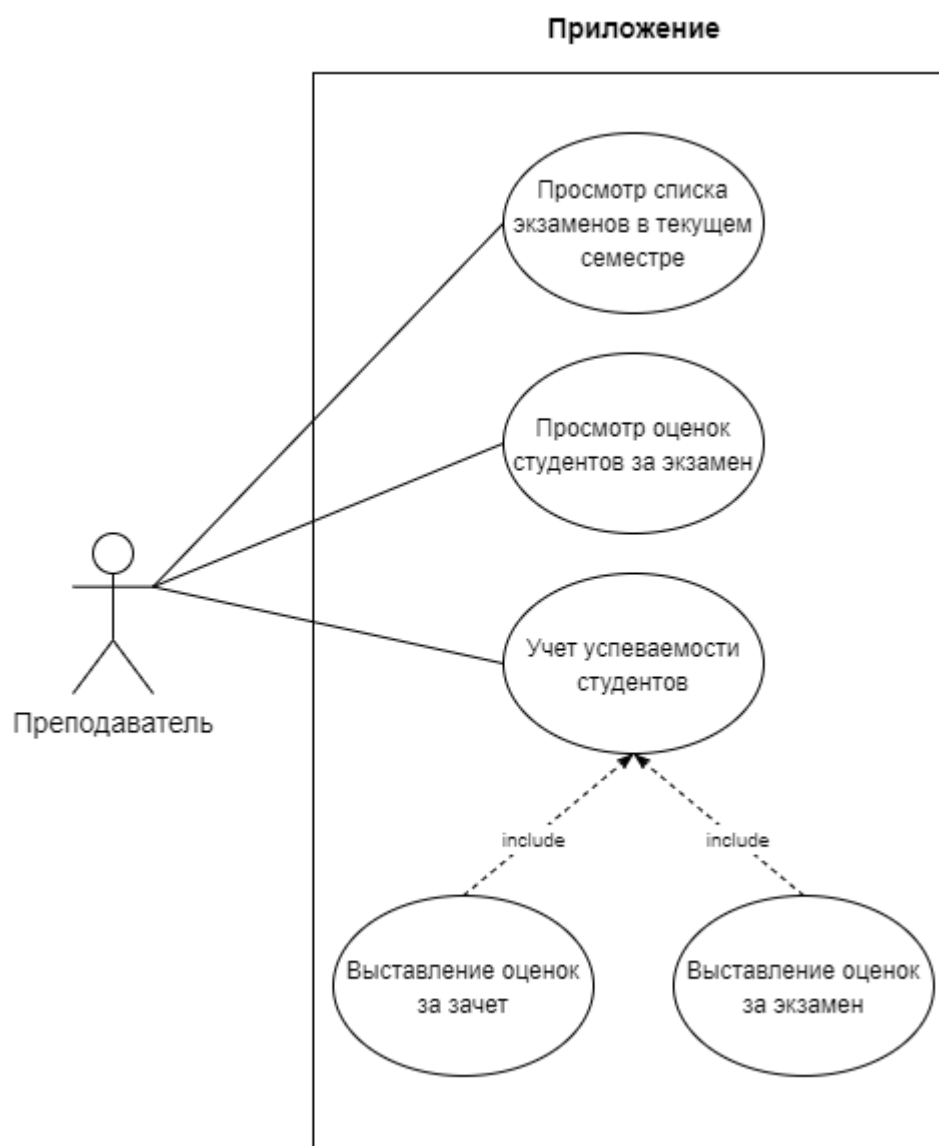


Рисунок 3- UML-диаграмма прецедентов разрабатываемого приложения

Производительность приложений на *Flutter* превосходит производительность приложений на *React Native*. В качестве основного языка программирования *Flutter* использует *Dart*, который в свою очередь компилируется в нативный код платформы, который в равных условиях всегда будет быстрее *JavaScript* от *React Native*. Также стоит отметить производительность рендеринга *Flutter*. *React Native* использует родные виджеты платформы и события передает через *JavaScript*, что сильно

сказывается на производительности. Конкуренты от *Google* используют собственный встроенный 2D движок *Skia*, что позволяет рендерить объекты с частотой до 120 кадров в секунду [3].

Для подтверждения вышесказанного было создано два приложения на *Flutter* и *React Native* с целью протестировать их работу со сложной анимацией. На экране располагается 200 изображений с разной анимацией (вращения, масштабирования и проявления), которая выполняется одновременно. Снимок рабочего экрана с приложением представлен на рисунке 4.



Рисунок 4- Тестовое приложение

В процессе тестирования фиксировались следующие данные: *FPS*, загрузка процессора и количества задействованной оперативной памяти [4]. Результат тестирования представлен в таблице 1.

Таблица 1- Результат тестирования производительности

	<i>FPS</i>	<i>CPU</i>	<i>Memory</i>
<i>React Native</i>	7 к/с	8.5%	424 мб
<i>Flutter</i>	19 к/с	10.28%	168 мб

Согласно этим данным, приложение, на *Flutter*, показало лучшую производительность, за счет большего количества кадров в секунду и меньшего количества затраченной памяти.

Конечное приложение на *React Native* — это *JavaScript* код. Любой может получить к нему доступ в конечной сборке релиза и изменить его логику. Итоговый нативный код приложения, написанного на *Flutter*, не читаем для человека, что делает практически невозможным обратную обработку. Кроме того, единая нативная библиотека со сложной структурой и меняющимися форматами данных, которая формируется после компиляции, не позволяет даже частично восстановить исходный код при реверс-инжиниринге.

Данная работа была написана на *Flutter*. Итоговое приложение, разработанное с помощью *Flutter* безопаснее и производительнее приложения, написанного на *React Native*, что нивелирует преимущества конкурента.

1.6.2 Затем необходимо выбрать среду разработки. Основными критериями при выборе среды разработки были:

- возможность поддержки языка *Dart*;
- наличие плагина для работы с фреймворком *Flutter* [5].

Этим условиям соответствуют следующие три среды разработки:

- *intelliJ IDEA*;
- *sndroid studio*;
- *visual studio code*.

IntelliJ IDEA популярная среда разработки, созданная компанией *JetBrains*, которая поддерживает множество языков программирования. Основным направлением данной среды является программирование на языке *Java*.

Преимущества:

- высокая интеллектуальность среды (дописывание кода, предложения о рефакторинге, навигация по коду);

- поддержка систем контроля версий;
- встроенный отладчик программы.

Недостатки:

- продукт предоставляется по платной подписке;
- высокая требовательность к использованию ресурсов оперативной памяти компьютера.

Среда разработки *Android studio* была создана компанией *Google* на основе ранее упомянутой *IntelliJ IDEA* для разработки мобильных приложений для устройств, управляемых операционной системой *Android*.

Преимущества:

- бесплатный продукт;
- встроенный эмулятор мобильных устройств;
- наличие удобного XML редактора;
- поддержка рендера средствами GPU;
- высокая интеллектуальность среды (дописывание кода, предложения о рефакторинге, навигация по коду);
- поддержка систем контроля версий;
- встроенный отладчик программы.

Недостатком является высокая требовательность к использованию ресурсов оперативной памяти компьютера.

Среда разработки *Visual Studio Code* была создана компанией *Microsoft* на основе ранее упомянутой *IntelliJ IDEA* для разработки мобильных приложений для устройств, управляемых операционной системой *Android*.

Преимущества:

- бесплатный продукт;
- встроенный эмулятор мобильных устройств;
- высокая интеллектуальность среды (дописывание кода, предложения о рефакторинге, навигация по коду);

- поддержка систем контроля версий;
- встроенный отладчик программы.

Недостатки:

- низкая производительность в сравнении с конкурентами
- недостаточно проработана поддержка *Dart* кода (возникают проблемы с “подсветкой” синтаксиса)

На основании приведенных выше данных было принято решение в качестве среды разработки использовать *Android Studio*.

1.6.3 Необходимо выбрать дополнительные библиотеки для разработки. Несмотря на то, что язык программирования *Dart* и фреймворк *Flutter* предоставляют множество возможностей для разработки мобильных приложений, многие сторонние разработчики создают собственные библиотеки для расширения функционала приложения. Для распространения данных пакетов командой Google был создан менеджер пакетов *pub*, который позволяет одной командой добавлять загруженные разработчиками библиотеки в свой проект.

В рамках разработки данного проекта были использованы следующие библиотеки:

- *flutter_native_splash* 2.0.5 - данный пакет позволяет настроить экран при запуске приложения;
- *get_it* 7.2.0- библиотека для внедрения зависимостей;
- *build_runner* 2.1.8- пакет, необходимый для кодогенерации;
- *hive_generator* 1.1.2- пакет для генерации адаптеров для объектов, которые нужно занести в *Hive* БД;
- *flutter_launcher_icons* 0.9.2 - пакет для установки иконки приложения;
- *flutter_bloc* 7.3.1 - данный пакет позволяет просто реализовать архитектурный паттерн *BLoC*. Он берет на себя организацию низкоуровневых

действий (создание потоков, передача данных по этим потокам, реагирование на новые данные в потоке);

- *hive 2.1.0* и *hive_flutter 1.1.0*- пакеты для работы с *NoSQL* базой данных *hive*. База данных написана на чистом *Dart*, отличается высокой скоростью работы с данными и легковесностью. Данные в базе хранятся по принципу ключ-значение.

1.7 Структура приложения

Структурно данное приложение, будет состоять из трех слоев:

- *presentation*;
- *gateway*;
- *domain*.

Presentation слой должен содержать в себе классы, реализующие графический интерфейс приложения.

Данное приложение будет состоять из трех экранов:

- экран со списком экзаменов, которые проводит преподаватель;
- экран с оценками студентов группы по конкретному экзамену;
- экран- модальное окно для выставления оценок студентам.

В документации флаттера написано “всё- виджеты”. Это высказывание говорит о том, что, все, связанное с графическим интерфейсом, состоит из виджетов. Код верстки на *Flutter* представляет собой иерархическую структуру виджетов, которую по-другому называют деревом. Это дерево определяет структуру экрана.

В коде виджет, который создается пользователем — это класс, который наследуется от класса *StatelessWidget* или *StatefulWidget*. *Stateless* виджеты отличаются от *Stateful* виджетов тем, что вторые умеют сохранять и изменять свое состояние. *Stateful* виджет обязательно создает объект класса-состояния, который наследуется от базового класса *State*. Именно этот объект хранит

состояние виджета, а именно он может самостоятельно изменять значение своих полей-признаков.

Таким образом для каждого экрана должен быть создан свой класс, который наследуется от класса *StatelessWidget* или *StatefulWidget*. Также многократно используемые компоненты будут вынесены в отдельный класс-виджет, чтобы избежать дублирования кода.

Еще в данном слое будет находиться *Business Logic Component (BLoC)*, который предназначен для управления логикой представления. *BLoC* работает по принципу реактивного программирования [6].

Реактивное программирование — это парадигма программирования, в основе которой лежат асинхронные потоки данных, которые отвечают за распространение изменений.

BLoC работает на двух потоках:

- поток событий (*event stream*);
- поток состояний (*state stream*).

В процессе работы приложения возникают события, которые передаются в поток событий. Он прослушивается в классе *BLoC*. Событие перехватывается и вызывается некоторый обработчик “*map event to state*”, который выполняет необходимые действия, связанные с событием (обращение к серверу для выполнения *REST* запроса или обращения к локальной базе данных, преобразование данных, которые пришли вместе с событием). Далее формируется объект состояния, который может содержать в себе некоторые данные. Данный объект отправляется в поток состояний, слушателем которого является графический интерфейс. В зависимости от полученного состояния графический интерфейс может изменить свой вид.

Для ускорения процесса разработки в данном проекте будет использована библиотека *flutter_bloc*, которая берет на себя создание вышеуказанных потоков, их закрытие и создание слушателей.

Необходимо будет создать следующие классы:

- *ExamBloc*, который будет наследоваться от базового класса *Bloc*.

Именно в нем будет описана логика представления;

- Абстрактный класс *ExamEvent*. От него будут наследоваться классы, объекты которых будут являться событиями. В рамках данного приложения будут необходимы следующие классы:

- 1) *LoadExamsEvent*. Объект этого класса будет являться событием, сообщаящим о необходимости загрузки списка экзаменов;

- 2) *LoadExamJournalEvent*. Объект этого класса будет являться событием, сообщаящим о необходимости загрузки журнала с оценками по экзамену;

- 3) *MarkCreatedUpdatedEvent*. Объект этого класса будет являться событием, сообщаящим о том, что была выставлена новая оценка или изменена старая.

- Абстрактный класс *ExamState*. От него будут наследоваться классы, объекты которых будут являться состояниями. В рамках данного приложения будут необходимы следующие классы:

- 1) *InitialState*. Объект этого класса будет являться начальным состоянием;

- 2) *ExamsLoadedState*. Объект этого класса будет являться состоянием, которое хранит в себе загруженные экзамены;

- 3) *ExamJournalLoadedState*. Объект этого класса будет являться состоянием, которое хранит в себе загруженный журнал экзамена.

Gateway слой должен содержать классы для работы с внешним источникам данных (локальной базой данных или сервером).

Изначально в рамках данного слоя необходимо реализовать классы-модели, которыми будут оперировать другие классы:

- *Mark*- класс, объект которого является оценкой;

- *Exam*- класс, объект которого является экзаменом;
- *Group*- класс объект которого является группой студентов;
- *Student*- класс объект которого является студентом.

Далее необходимо разработать класс- интерфейс *ExamGateway*. Данный класс должен иметь следующие методы:

- *getGroups*, который возвращает список всех групп;
- *getExams*, который возвращает список всех экзаменов;
- *getMarks*, который возвращает список всех оценок;
- *createUpdateMark*, который создает новую оценку или обновляет старую.

Реализовывать данный интерфейс будет класс *ExamGatewayImpl*.

Так как все перечисленные выше методы подразумевают работу с сервером, а серверная часть не реализуется в рамках данного проекта, необходимо будет реализовать методы, эмулирующие работу с ним:

- *_groupAPICallEmulation*, который создаёт список групп и студентов в них;
- *_examAPICallEmulation*, который создает список экзаменов.

Также из-за отсутствия серверной части оценки студентов будут сохраняться в локальной *NoSQL* базе данных *Hive*. Для взаимодействия с ней должен быть создан класс *HiveDB*, который будет содержать в себе методы:

- *init*, который инициализирует базу данных;
- *getMarks*, который возвращает список оценок из базы данных;
- *createUpdateMark*, который кладет новую оценку в базу данных или обновляет уже существующую.

Условно данные в *HiveDB* хранятся в коробках, каждая из которых имеет собственное название [7]. Необходимо создать две коробки:

- *lastMarkIdBoxName* — это коробка, где будет храниться идентификатор последней добавленной оценки. Его необходимо хранить, потому что при

добавлении новой оценки ей будет присваиваться id, и, чтобы он не совпал с идентификатором старой оценки, последний будет инкрементироваться, и полученное значение будет выступать в качестве id новой оценки;

- *marksBoxName* — это коробка, которая и будет хранить в себе оценки.

Слой *domain* должен содержать классы, которые реализуют бизнес логику приложения. Необходимо разработать класс- интерфейс *ExamService*, который будет содержать в себе методы, взаимодействующие со слоем *gateway*, в целях получения данных из внешних источников. При необходимости данные нужно будет преобразовать к подходящему виду для этого должен быть создан класс- интерфейс *ExamFactory*, методы которого будут получать в качестве параметров некоторые структуры данных и преобразовывать их к виду, который необходим для *presentation* слоя. Данные интерфейсы будут реализовывать классы *ExamServiceImpl* и *ExamFactoryImpl* соответственно.

Методы, которые должен содержать класс *ExamService* представлены ниже:

- *getGroupsExams*- метод, который возвращает экзамены группы;
- *getExamJournal*- метод, который возвращает журнал экзамена;
- *createUpdateMark*- метод, который создает новую оценку или обновляет уже существующую.

UML диаграммы классов, которые необходимо разработать, представлены на рисунках 5 и 6.

1.8 Реализация

Процесс разработки можно поделить на следующие пункты:

- разработка слоя *presentation*:
 - 1) разработка графического интерфейса;
 - 2) разработка логики представления;

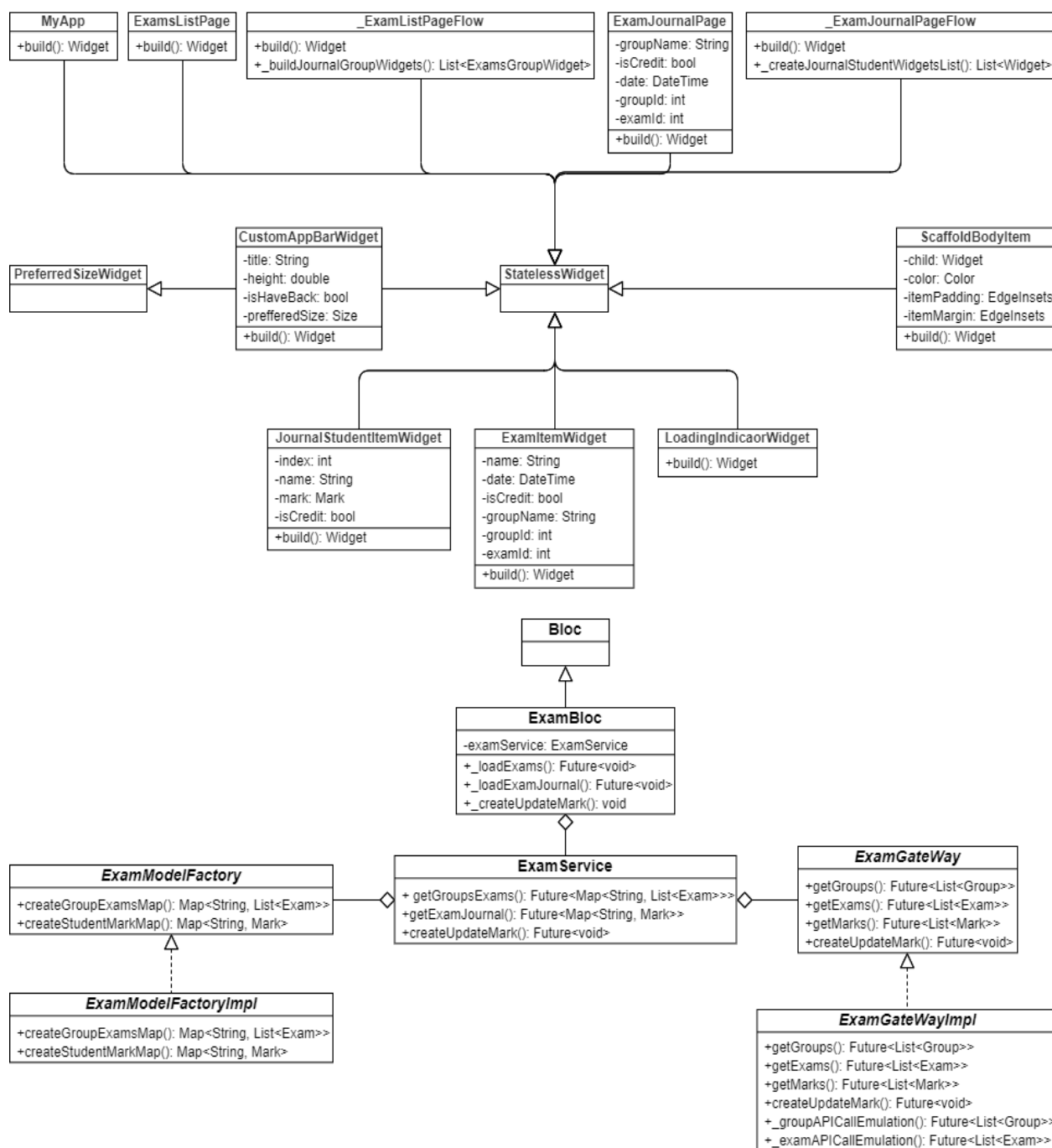


Рисунок 5 — UML- диаграмма классов часть 1

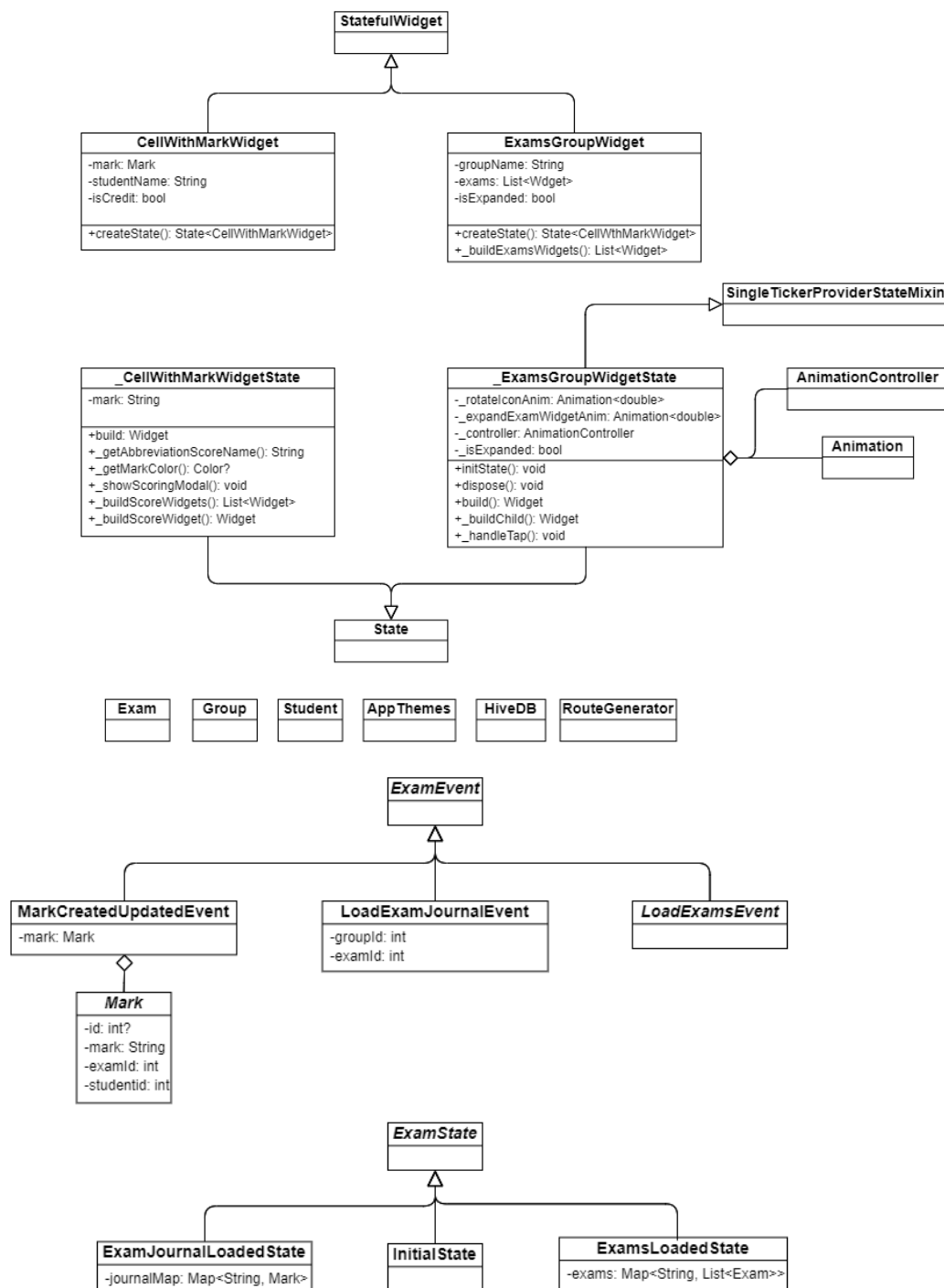


Рисунок 6 — UML- диаграмма классов часть 2

UML- диаграмма пакетов представлена на рисунке 7.

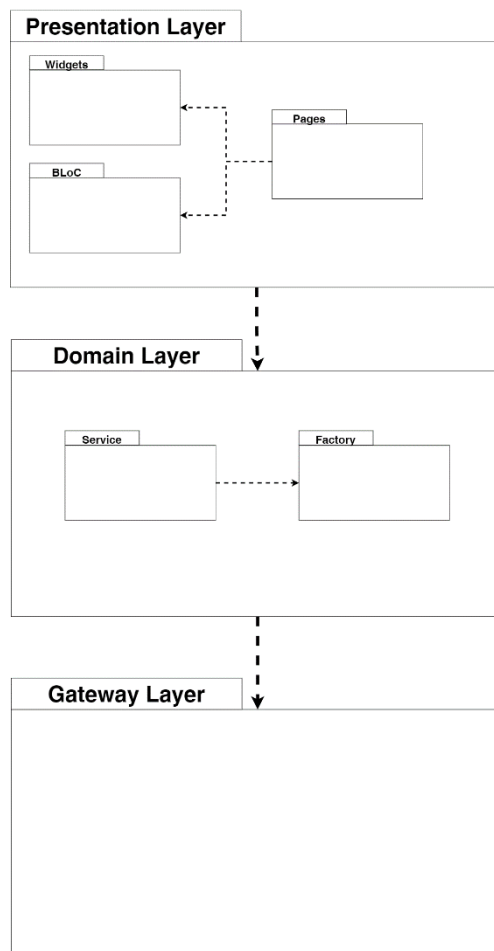


Рисунок 7 — UML- диаграмма пакетов

- разработка слоя *domain*:

- 1) разработка классов- моделей;
- 2) разработка классов- сервисов, реализующих бизнес- логику приложения;
- 3) разработка классов для преобразования данных;

- разработка слоя *gateway*:

- 1) разработка классов, необходимых для эмуляции работы с сервером;
- 2) разработка классов;
- 3) разработка классов-моделей.

1.8.1 Изначально был разработан слой *presentation*. Виджеты *Flutter* можно поделить на три категории:

- *Proxy widgets*- эти виджеты не участвуют в формировании пользовательского интерфейса. Они нужны для хранения информации, которая может понадобиться их поддереву виджетов. К таким виджетам, например, относится *InheritedWidget*;

- *Component widgets*- эти виджеты имеют непосредственное отношение к формированию пользовательского интерфейса и способа взаимодействия с ним. По сути их и видит конечный пользователь. К этой категории виджетов относятся *Text*, *Container*, *GestureDetectore* и так далее;

- *Render widgets* — это виджеты, которые отвечают за структуризацию экрана. Они в конечном итоге определяют расположение, размер *Component widgets*. К этой категории виджетов относятся *Row*, *Column*, *Padding* и так далее.

Каждый экран приложения разрабатывался по следующему алгоритму:

- формировалась структура экрана с помощью *render widgets*;
- опираясь на структуру, экран заполнялся *component widgets*;
- при необходимости передачи данных в виджеты, в дерево не добавлялся *proxy widgets*. Вместо этого использовался сервис локатор.

Согласно спроектированной диаграмме классов, было разработано десять классов- *Stateless* виджетов, каждый из них унаследован от класса *Stateless Widget*. Четыре из них представляют собой страницы приложения:

- *ExamListPage* и *_ExamListPageFlow*- реализуют страницу со списком экзаменов;

- *ExamJournalPage* и *_ExamJournalPageFlow*- реализуют страницу с журналом экзамена.

Оставшиеся шесть классов- виджетов представляют отдельные элементы:

- *MyApp*- виджет, с которого приложение начинает работу;
- *CustomAppBarWidget*- виджет, который является верхней панелью приложения;
- *ScaffoldBodyItem*- это виджет оболочка, в который обрачиваются все виджеты, которые находятся в основной зоне приложения;
- *ExamItemWidget*- это виджет, который является элементов в списке экзаменов;
- *JournalStudentItemWidget*- это виджет, который является строкой журнала. Он используется в классе *_ExamJournalPageFlow*;
- *LoadingIndicatorWidget*- это виджет, который является индикатором загрузки.

Обычно *Stateless* виджет принимает свои признаки, например цвет, через конструктора класса. Поля класса виджета- являются атрибутами его состояния. *Stateless* виджет не может самостоятельно их изменить. Единственная функция, которую обязательно должен содержать *Stateless* виджет- это функция *build*, в котором определяется компоновка виджета. Для примера ниже представлен код *Stateless* виджета *ScaffoldBodyItem*:

```
class ScaffoldBodyItem extends StatelessWidget {
  final Widget child;
  final Color color;
  final EdgeInsets itemPadding;
  final EdgeInsets itemMargin;

  const ScaffoldBodyItem({
    Key? key,
    required this.child,
    this.color = Colors.white,
    this.itemPadding = const EdgeInsets.symmetric(vertical:
20, horizontal: 16),
```

```

        this.itemMargin = const EdgeInsets.symmetric(vertical: 0,
horizontal: 0),
    }) : super(key: key);

@override
Widget build(BuildContext context) {
    return Container(
        margin: itemMargin,
        child: ClipRRect(
            borderRadius: const
BorderRadius.all(Radius.circular(12.0)),
            child: Container(
                width: double.infinity,
                padding: itemPadding,
                decoration: BoxDecoration(
                    color: color,
                ),
                child: child,
            ),
        ),
    );
}
}

```

Согласно разработанной диаграмме классов, было разработано два класса- *Stateful* виджета, каждый из них унаследован от класса *StatefulWidget*. Эти виджеты являются отдельными элементами, используемыми на странице. Их реализация в виде *Stateful* виджета обусловлена тем, что они должны часто изменять свой внешний вид, то есть свое состояние. Вызов функции *setState* для этих виджетов вызывает только их перерисовку, что положительно влияет на производительность приложения:

- *CellWithMarkWidget*- это виджет, являющийся клеткой в журнале, куда ставится оценка. Он используется в виджете *JournalStudentItemWidget*;

- *ExamsGroupWidget*- это виджет, который является списком экзаменов одной конкретной группы.

Перечисленные выше виджеты создают объекты- состояния следующих классов:

- *_CellWithMarkWidgetState*;
- *_ExamsGroupWidgetState*.

Stateful виджет также может получать с помощью конструктора данные, которые могут являться некоторыми признаками виджета, но эти признаки также не могут измениться по воле виджета. Как было сказано ранее в отличие от *Stateless* виджета, *Stateful* виджет создает объект-состояние (это происходит с помощью функции *createState*), где хранятся признаки, которые виджет может самостоятельно изменять. Это происходит с помощью метода *setState*. Класс, который наследуется от класса *State* обязательно должен иметь метод *build*, в котором определяется компоновка виджета. Для примера код *Stateful* виджета и его *State*-объекта представлен ниже:

```
class CellWithMarkWidget extends StatefulWidget {
  Mark mark;
  final String studentName;
  final bool isCredit;

  CellWithMarkWidget(
    {required this.mark,
    required this.studentName,
    required this.isCredit,
    Key? key})
    : super(key: key);

  @override
  State<CellWithMarkWidget> createState() =>
    _CellWithMarkWidgetState();
}
```

```

class _CellWithMarkWidgetState extends
State<CellWithMarkWidget> {
    late String mark;

    _CellWithMarkWidgetState() {
        mark = widget.mark.mark;
    }
    @override
    Widget build(BuildContext context) {
        return InkWell(
            onTap: () => _showScoringModal(context),
            child: Container(
                height: 40,
                width: 40,
                decoration: BoxDecoration(
                    border: Border.all(width: 0.8, color:
AppColors.grayTextColor),
                    borderRadius: BorderRadius.circular(4)),
                child: Center(
                    child: Text(_getAbbreviationScoreName(),
                        style: TextStyle(fontSize: 18, color:
_getMarkColor()))),),),);}

```

В рамках разработки логики представления, согласно спроектированной диаграмме классов, был разработан класс *ExamBloc*, он наследуется от класса *Bloc*. В конструкторе данного класса с помощью ключевого слова *on* определяется как *BLoC* реагирует на то или иное событие:

```

ExamBloc({required this.examService}) :
super(InitialState()) {
    on<LoadExamsEvent>((event, emit) => _loadExams(event,
emit));
}

```

В данном случае, если в *BLoC* поступает событие *LoadExamsEvent*, вызывается функция *_loadExams*, которая в качестве параметров принимает объекты *ExamEvent* и *Emitter*. Последний нужен для того, чтобы “выбрасывать” объект состояния в поток состояний, который прослушивается виджетом. Функция асинхронна, потому не тормозит основной поток программы и приложение работает без визуальных притормаживаний:

```
Future<void> _loadExamJournal(
    LoadExamJournalEvent event, Emitter<ExamState> emit)
async {
    final journalMap = await examService.getExamJournal(
        groupId: event.groupId, examId: event.examId);
    emit(ExamJournalLoadedState(journalMap: journalMap));
}
```

Согласно спроектированной диаграмме классов, был разработан абстрактный класс *ExamEvent*:

```
abstract class ExamEvent {}
```

Так же были разработаны классы, объекты которых являются событиями:

- *MarkCreatedUplatedEvent*. Данный класс содержит поле с типом данных *Mark*. Класс *Mark*- является оценкой;
- *LoadExamJournalEvent*. Данный класс содержит поля типа *int*: *groupId* (идентификатор группы), *examId* (идентификатор экзамена);
- *LoadExamsEvent*.

Пример кода класса- события *LoadStudentsEvent* представлен ниже:

```
class LoadStudentsEvent extends ExamEvent {
    final int groupId;
    LoadStudentsEvent({required this.groupId});}
```

Согласно спроектированной диаграмме классов, был разработан абстрактный класс *ExamState*:

```
abstract class ExamState {}
```

Так же были разработаны классы, объекты которых являются состояниями:

- *InitialState*;
- *ExamsLoadedState*. Данный класс содержит поле, которое является словарем. В этом словаре ключом является название группы (объект типа *String*), а в качестве значения выступает список экзаменов (объект типа *List<Exam>*);
- *ExamJournalLoadedState*. Данный класс содержит поле, которое является словарем. В этом словаре ключом является имя студента (объект типа *String*), а в качестве значения выступает оценка (объект типа *Mark*).

Пример кода класса- состояния *ExamsLoadedState* представлен ниже:

```
class ExamsLoadedState extends ExamState {  
    final Map<String, List<Exam>> exams;  
    ExamsLoadedState({required this.exams});  
}
```

1.8.2 Затем был разработан слой *gateway*. Согласно спроектированной диаграмме классов, были разработаны классы- модели:

- *Mark*- класс, который содержит поля:
 - 1) *int id*- идентификатор оценки;
 - 2) *String mark*- оценка;
 - 3) *int examid*- идентификатор экзамена;
 - 4) *int studentId*- идентификатор студента, которому оценка принадлежит;
- *Exam*- класс, который содержит поля:
 - 1) *int id* - идентификатор экзамена;
 - 2) *bool isCredit*- поле, которое показывает является ли поле зачетом;
 - 3) *DateTime date*- дата экзамена;
 - 4) *int groupId*- идентификатор группы, в которой проходит экзамен;

- *Group*- класс объект которого является группой студентов:

1) *int id*- идентификатор группы;

2) *String name*- название группы;

3) *List<Student> students*- список студентов в группе;

- *Student*- класс объект которого является студентом:

1) *int id*- идентификатор студента;

2) *String name*- имя студента.

Также был разработан класс- интерфейс *ExamGateway* для взаимодействия с внешними источниками данных. Данный интерфейс был реализован классом *ExamGatewayImpl*, конкретно были реализованы следующие методы:

- *getGroups*, типом возвращаемого значения которого является *Future<List<Group>>*;

- *getExams*, типом возвращаемого значения которого является *Future<List<Exam>>*;

- *etMarks*, типом возвращаемого значения которого является *Future<List<Mark>>*;

- *createUpdateMark*, типом возвращаемого значения которого является *Future<List<Group>>*.

У всех методов, которые перечислены выше, типом возвращаемого значения является объект *Future*, так как методы асинхронны. Это сделано для того, чтобы не тормозить основной поток выполнения программы [8]. Для примера ниже приведен код метода *getGroups*:

```
Future<List<Group>> getGroups() async {  
    return await _groupAPICallEmulation();  
}
```

Также были созданы методы, эмулирующие работу с сервером:

- *groupAPICallEmulation*, типом возвращаемого значения которого является *Future<List<Group>>*. Данный метод с помощью вложенных циклов создает четыре группы и для каждой из них создает двадцать студентов;

- *examAPICallEmulation*, типом возвращаемого значения которого является *Future<List<Exam>>*. Данный метод для каждой группы создает два экзамена (один из них зачет).

Перечисленные выше методы также являются асинхронными, потому тоже имеют тип возвращаемого значения *Future*. Для примера ниже приведен код метода *_groupAPICallEmulation*:

```
Future<List<Group>> _groupAPICallEmulation() async {
  var studentId = 0;
  List<Group> groups = [];
  return await Future.delayed(const Duration(seconds: 2), ()
  {
    for (int i = 0; i < 4; i++) {
      List<Student> students = [];
      for (int j = 0; j < 20; j++) {
        var student = Student(id: studentId, name: 'Студент
$studentId');
        studentId++;
        students.add(student);
      }
      var group = Group(id: i, name: 'Группа $i', students:
students);
      groups.add(group);
    }
    return groups;
  });
}
```

Для взаимодействия с локальной *NoSQL* базой данных *Hive* был создан класс *HiveDB*, который содержит в себе методы:

- *init*, типом возвращаемого значения которого является *Future<void>*.

Для инициализации базы данных в процессе исполнения данного метода выбирается путь, по которому база данных будет сохранена, а также регистрируются адаптеры. Адаптеры нужны для того, чтобы сохранять объекты класса в БД, ведь по умолчанию в *Hive DB* можно хранить только данные примитивного типа, типа *DateTime*, *BigInt*, *Uint8List*, их списки (*List*), словари (*Map*). Ниже приведен код данного метода:

```
static Future<void> init() async {  
    final dir = await getApplicationDocumentsDirectory();  
    Hive.init(dir.path);  
    Hive.registerAdapter(MarkAdapter());  
};
```

- *getMarks*, типом возвращаемого значения которого является *Future<List<Mark>>*. Данный метод взаимодействует с коробкой *marksBoxName*, доставая из нее все имеющиеся оценки. После завершения работы с коробкой она сжимается и закрывается, с помощью методов *compress* и *close* соответственно;

- *createUpdateMark*, типом возвращаемого значения которого является *Future<void>*. Данный метод взаимодействует с коробками *lastMarkIdBoxName*, доставая из нее идентификатор последней добавленной оценки. Также данный метод взаимодействует с коробкой *marksBoxName*, которая и хранит в себе оценки. Для обновления данных в БД используется метод *put*, который в качестве параметров принимает ключ и значение. Для демонстрации работы с базой данных ниже приведен код данного метода:

```
static Future<void> createUpdateMark(int? markId, Mark mark)  
async {  
    final lastMarkIdBox = await  
    Hive.openBox<dynamic>(lastMarkIdBoxName);  
    final marksBox = await Hive.openBox<dynamic>(marksBoxName);  
    if (markId == null) {
```

```

var lastMarkId = lastMarkIdBox.get('lastMarkId');
if (lastMarkId == null) {
    lastMarkId = 0;
} else {
    lastMarkId++;
}
lastMarkIdBox.put('lastMarkId', lastMarkId);
mark.id = lastMarkId;
markId = lastMarkId;
}
marksBox.put('$markId', mark);
lastMarkIdBox.compact();
lastMarkIdBox.close();
marksBox.compact();
marksBox.close();
}

```

Перечисленные выше методы также являются асинхронными, потому тоже имеют тип возвращаемого значения *Future*.

Далее был разработан слой *domain*. Согласно спроектированной диаграмме классов, был разработан класс-интерфейс *ExamService* и реализующий его класс *ExamServiceImpl*. Конкретно были реализованы следующие методы:

- *getGroupsExams*- метод который возвращает *Future<Map<String, List<Exam>>>* (объект *Future*, который содержит в себе словарь, где ключом является название группы а значением список ее экзаменов);
- *getExamJournal*- метод, который возвращает *Future<Map<String, Mark>>* (объект *Future*, который содержит в себе словарь, где ключом является имя студента, а значением его оценка);
- *createUpdateMark*- метод, который возвращает *Future<void>*.

Также был создан класс *ExamModelsFactory*, реализующий его класс *ExamModelsFactoryImpl*. Конкретно были реализованы следующие методы:

- *createGroupExamsMap*, который используется в методе *getGroupsExams* класса *ExamService* для создания из списка всех групп и из списка всех экзаменов словаря, где ключом является название группы а значением список ее экзаменов (тип возвращаемого значения *Future<Map<String, List<Exam>>>*);

- *createStudentMarkMap*, который используется в методе *getExamJournal* класса *ExamService* для создания из списка всех групп и оценок словаря, где ключом является имя студента, а значением его оценка (тип возвращаемого значения *Future<Map<String, Mark>>*).

Для примера приведен код функции *getGroupsExams*, которая обращается к слою *gateway*, чтобы получить списки групп и экзаменов:

```
Future<Map<String, List<Exam>>> getGroupsExams() async {
    final groups = await examGateWay.getGroups();
    final exams = await examGateWay.getExams();
    return examModelsFactory.createGroupExamsMap(groups,
exams);
}
```

В конце вызывается метод *createGroupExamsMap*, который в качестве параметров принимает эти списки и возвращает *Map*, где в качестве ключа используется имя группы, а в качестве значения список ее экзаменов:

```
Map<String, List<Exam>> createGroupExamsMap(
    List<Group> groups, List<Exam> exams) {
    Map<String, List<Exam>> groupExamMap = {};
    for (var group in groups) {
        List<Exam> groupExams = [];
        for (var exam in exams) {
            if (exam.groupId == group.id) groupExams.add(exam);
        }
        groupExamMap.addAll({group.name: groupExams});
    }
    return groupExamMap;}
```

1.9 Руководство пользователя

После запуска пользователь попадает на страницу со списком групп и их экзаменов. Страница представлена на рисунке 8:

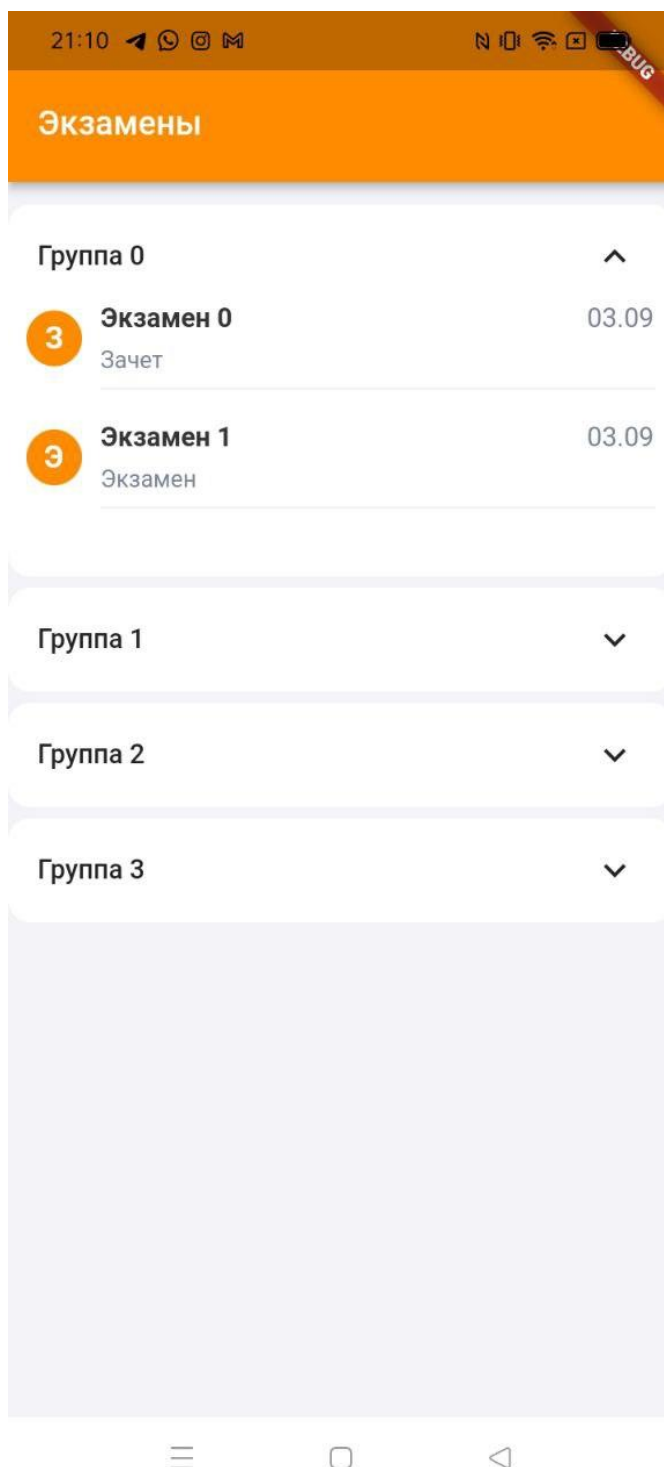


Рисунок 8- Страница со списком групп и их экзаменов

После тапа на нужный экзамен пользователь попадает на страницу с оценками по данному экзамену. Страница представлена на рисунке 9:

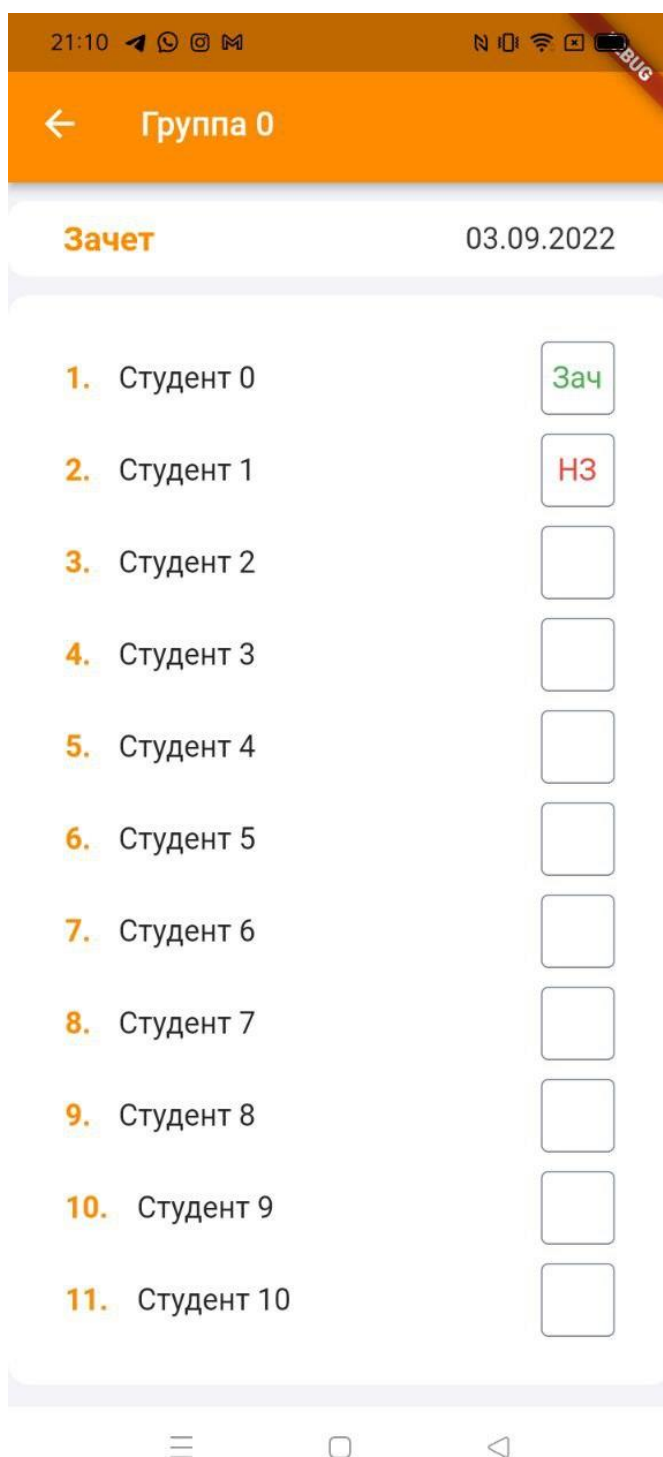


Рисунок 9- Страница с оценками по данному экзамену

После тапа на клетку, где должна стоять оценка, появляется модальное окно для ее выбора. Модальное окно представлено на рисунке 10:

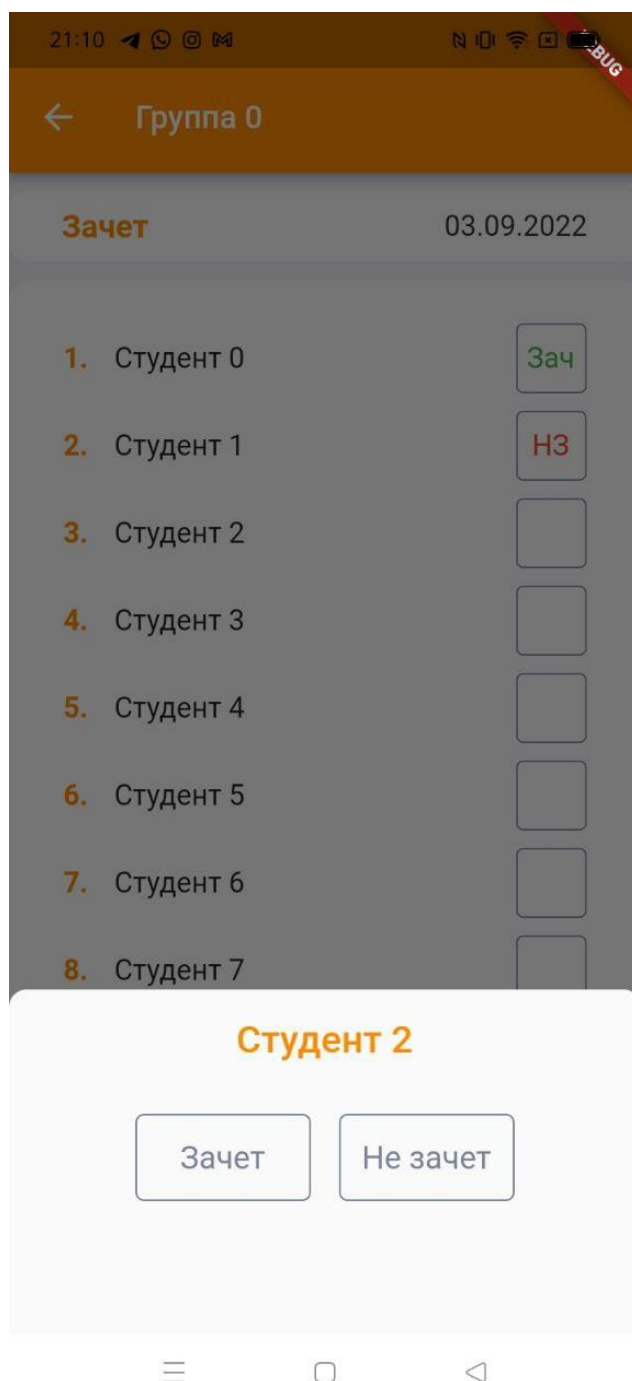


Рисунок 10- Модальное окно для выбора оценки

Если оценка уже стояла, она выделяется в модальном окне. Модальное окно с выделенной оценкой представлено на рисунке 11.

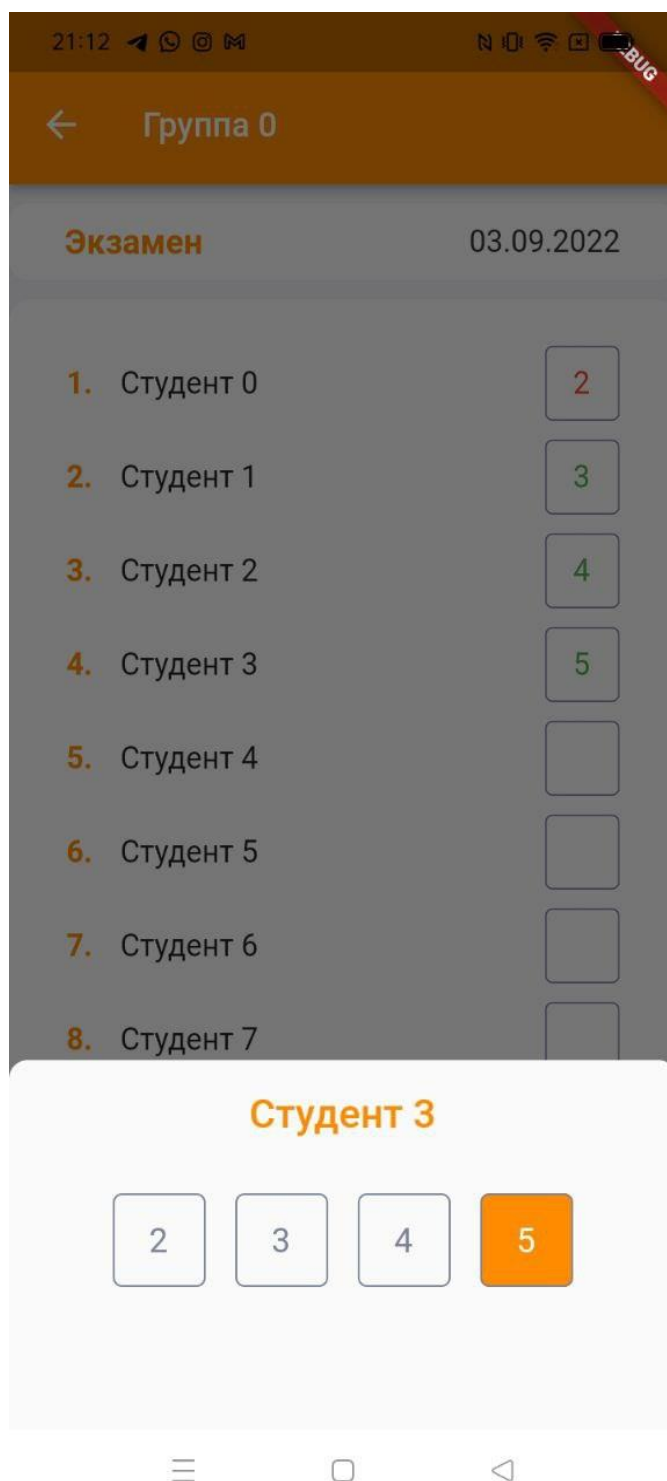


Рисунок 11- Модальное с выбранной оценкой

Наглядно алгоритм использования приложения (выставления оценки за экзамен или зачет) представлен на *UML*- диаграмме деятельности, которая представлена на рисунке 12.

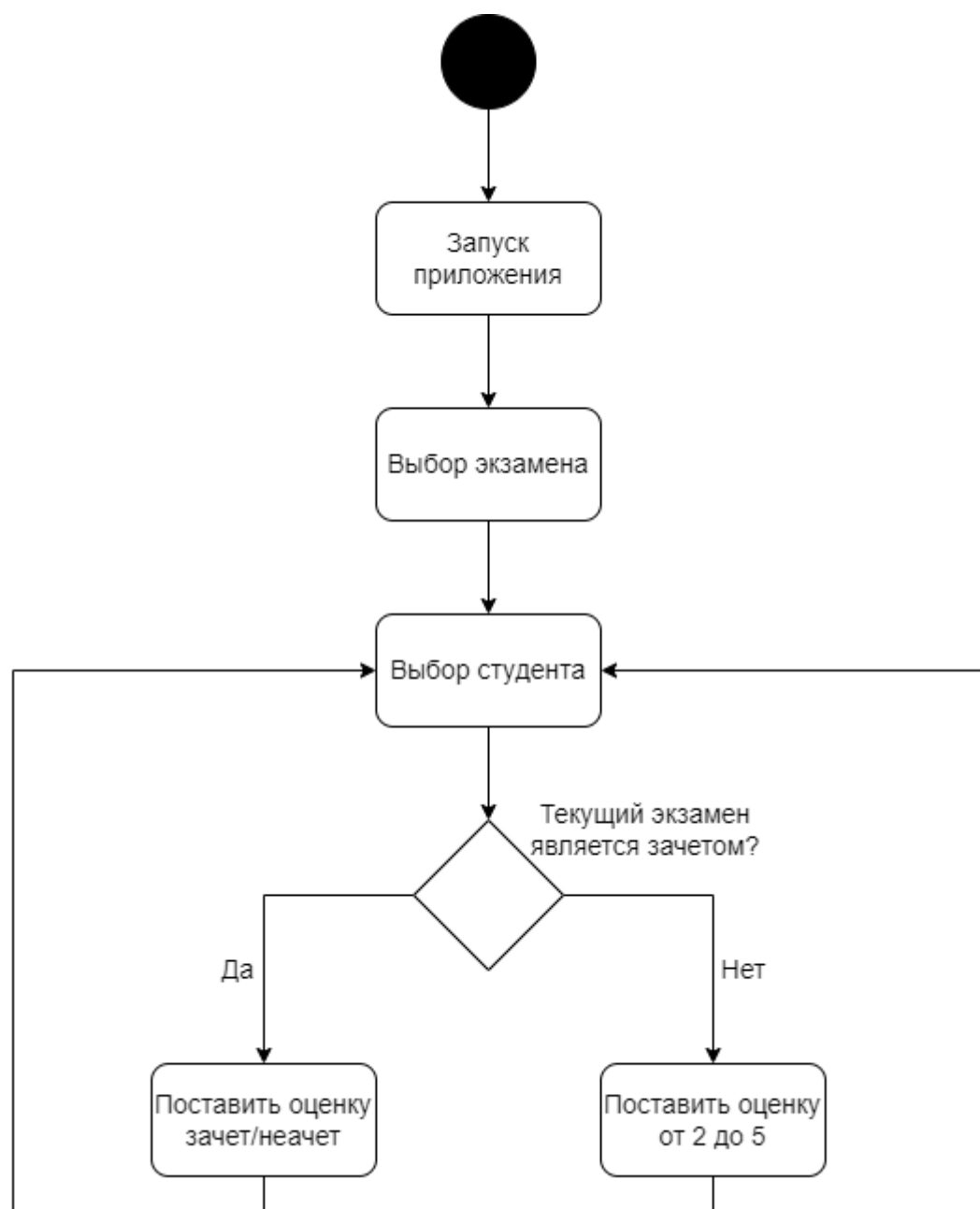


Рисунок 12- UML- диаграмма деятельности выставления оценки за зачет или экзамен

2 Охрана труда

Несмотря на огромную пользу, которую ежедневно приносят компьютеры в жизнь современного человека, они так же могут оказывать негативный эффект на здоровье человека. Для его минимизации необходимо соблюдать некоторые меры безопасности.

Необходимо правильно обустроить рабочее место:

- рабочее пространство должно хорошо освещаться естественным и искусственным источником света;
- естественные источники освещения в виде оконных проемов должны быть чистыми и должны быть оснащены жалюзи или другими устройствами подобного типа;
- естественный свет должен падать сбоку, лучше всего слева;
- искусственное освещение должно быть равномерно распределено в помещении;
- при необходимости рабочее место должно быть оборудовано локальным источником света;
- освещенность рабочей поверхности в зоне размещения рабочего документа должна быть в пределах от 300 лк до 500 лк;
- рабочая поверхность стола должна находиться на высоте 725 мм над уровнем пола или она должна регулироваться в пределах от 600 мм до 800 мм;
- пространство для ног под рабочей поверхностью должно быть не менее 600 мм;
- компьютерное кресло должно быть в меру жестким и регулироваться по высоте, углу наклона спинки и расстоянию между передним краем сиденья и спинкой;

- монитор нужно установить на рабочей поверхности таким образом, чтобы расстояние между глазами рабочего и дисплеем составляло не менее 500 мм;

- ноги должны быть согнуты в коленях под углом в 90 градусов и стоять на плоской поверхности. При необходимости можно использовать подставку;

- несколько рабочих мест в комнате должны быть рассредоточены таким образом, чтобы расстояние между боковыми гранями мониторов составляло от 1,2м и больше, а расстояние между дисплеем одного монитора и задней грани другого не менее 2м.

Для комфортной работы помещение, где стоят компьютеры, должно хорошо проветриваться. Температура воздуха должна быть в пределах от плюс 19 до плюс 23 градусов цельсия, а относительная влажность воздуха около 55%.

Уровень шума в рабочем помещении, регулируется санитарными нормами. На рабочих местах он не должен превышать 50 дБА.

Пользователи ПЭВМ не должны непрерывно работать более одного часа. Каждый час необходимо делать перерыв на 10- 15 минут. В перерывах рекомендуется делать упражнения, которые способствуют снижению усталости глаз:

- перевод взгляда с кончика носа, на дальний угол комнаты;
- частое моргание в течение двух минут;
- массаж глаз с помощью легкого надавливания на веки;
- вращение глазами в виде восьмерки.

При соблюдении указанным ранее мерам безопасности и следованию рекомендациям, обозначенным выше, негативное воздействие на пользователя ПЭВМ снижается к минимуму.

Заключение

В процессе исследования была подтверждена актуальность проекта. Также был проведен анализ целевой аудитории и ее потребностей. На основании потребностей был определен основной функционал приложения. Были рассмотрены аналоги проекта, выявлены их преимущества и недостатки, на основании которых были определены характеристики, которыми должно обладать разрабатываемое приложение. Также были рассмотрены особенности двух самых популярных мобильных операционных систем: *Android* и *IOS*. Было решено, что приложение должно работать на обеих ОС, потому были рассмотрены преимущества и недостатки двух способов разработки: для каждой операционной системы отдельно и кроссплатформенная разработка. Предпочтение было отдано второму способу.

После, в рамках выбора инструментов разработки были рассмотрены два популярных фреймворка для кроссплатформенной мобильной разработки: *Flutter* и *React Native*. На основании их сравнения первый был выбран для разработки данного проекта. Также были проанализированы три среды разработки: *IntelliJ IDEA*, *Android Studio*, *Visual Studio Code*. После рассмотрения их преимуществ и недостатков вторая была выбрана для разработки проекта. На основании предполагаемого функционала приложения для облегчения разработки были выбраны дополнительные библиотеки.

Затем была продумана архитектура приложения. Было решено, что она будет состоять из трех взаимодействующих между собой слоев: *presentation*, *gateway*, *domain*. Согласно продуманной структуре, было разработано приложение для ведения учета успеваемости студентов, а именно были написаны классы отвечающие за графический интерфейс приложения, управление логикой представления, бизнес логику, работу с локальной базой данных, для работы с сервером. В процессе разработки были укреплены

навыки работы с языком Dart, фреймворком для кроссплатформенной разработки Flutter, а именно были изучены различные категории виджетов и паттерн проектирования архитектуры представления BLoC, который рекомендован командой Google. Также были улучшены навыки работы с не реляционной базой данных Hive.

В процессе реализации данного проекта были достигнуты все поставленные цели. Итоговое приложение работает исправно и выполняет все предназначенные ему задачи. В перспективе приложение может расширить спектр своих возможностей. Планируется добавить функционал авторизации пользователей, учета сдачи лабораторных и курсовых работ, выставление промежуточных контрольных точек.

Список используемых источников

1. Приказ Минобразования РФ от 23 августа 2017 г. № 816 «Об утверждении Порядка применения организациями, осуществляющими образовательную деятельность, электронного обучения, дистанционных образовательных технологий при реализации образовательных программ». — URL: <http://publication.pravo.gov.ru/Document/View/0001201709200016> (дата обращения 26.03.2022).
2. Медникс З. Программирование под Android. 2-е изд. — СПб.: «Питер», 2013. — С. 345-475.
3. Андрей Алеев. Быстрый старт Flutter-разработчика. — Екб.: «Издательские решения», 2019. — С. 73-102.
4. Flutter vs React Native vs Native: Deep Performance Comparison. — URL: <https://medium.com/swlh/flutter-vs-react-native-vs-native-deep-performance-comparison-990b90c11433> (дата обращения 07.04.2022).
5. Крис Баккет. Dart в действии — Перевод с английского. — М.: «ДМК Пресс», 2016. — С. 135-309.
6. BLoC. — URL: <https://bloclibrary.dev/#/gettingstarted> (дата обращения 08.04.2022).
7. Hive Docs. — URL: <https://docs.hivedb.dev/#/> (дата обращения 09.04.2022).
8. Гамма Э., Влиссидес Дж., Джонсон Р., Хелм Р. Приемы объектно-ориентированного проектирования. Паттерны проектирования — СПб.: «Питер», 2019. — С. 267-312.

Приложение А

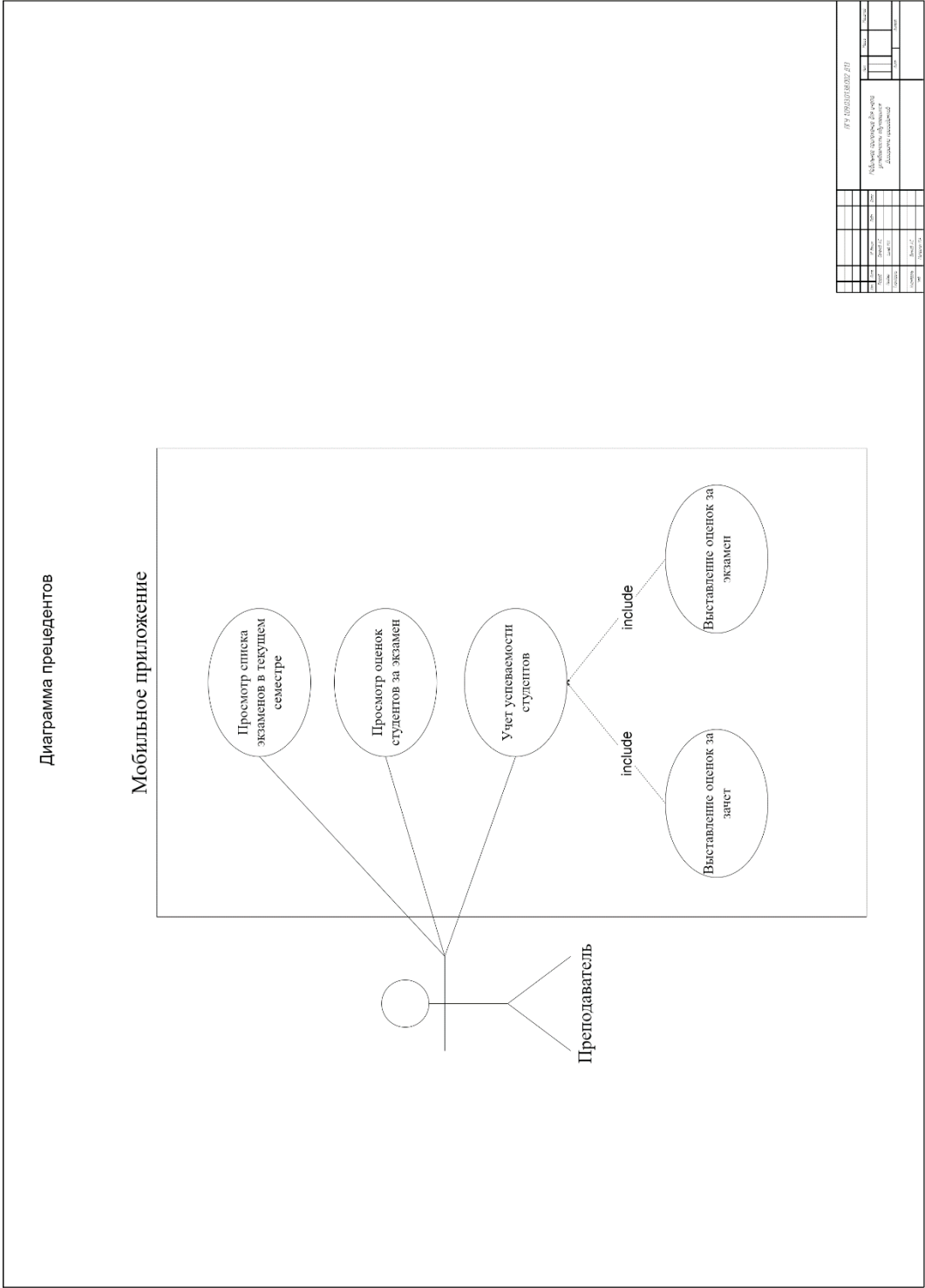
(обязательное)

Мобильное приложение для учета успеваемости студентов

UML- диаграммы

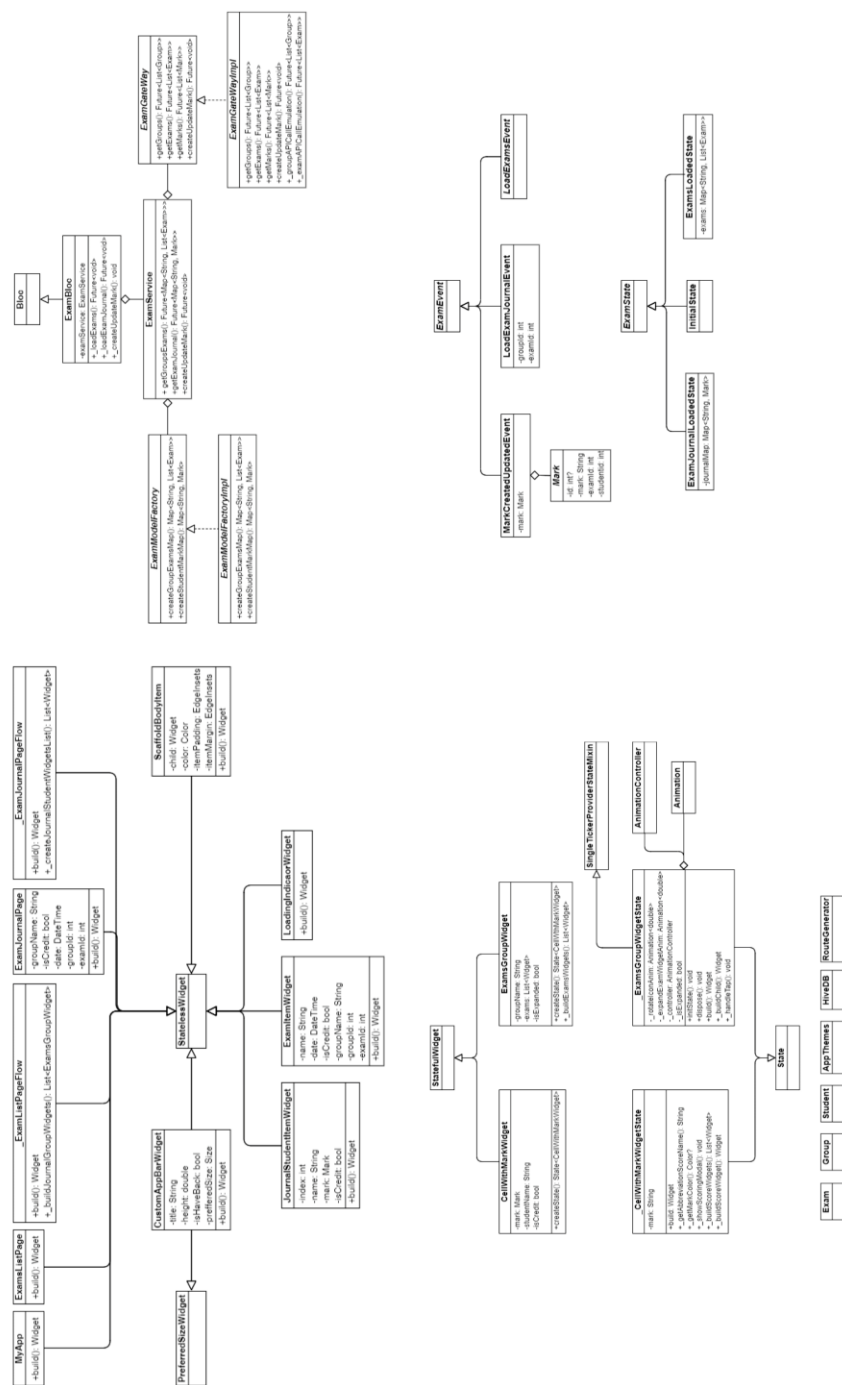
					ПГУ 1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		51

Приложение А.1- Диаграмма прецедентов

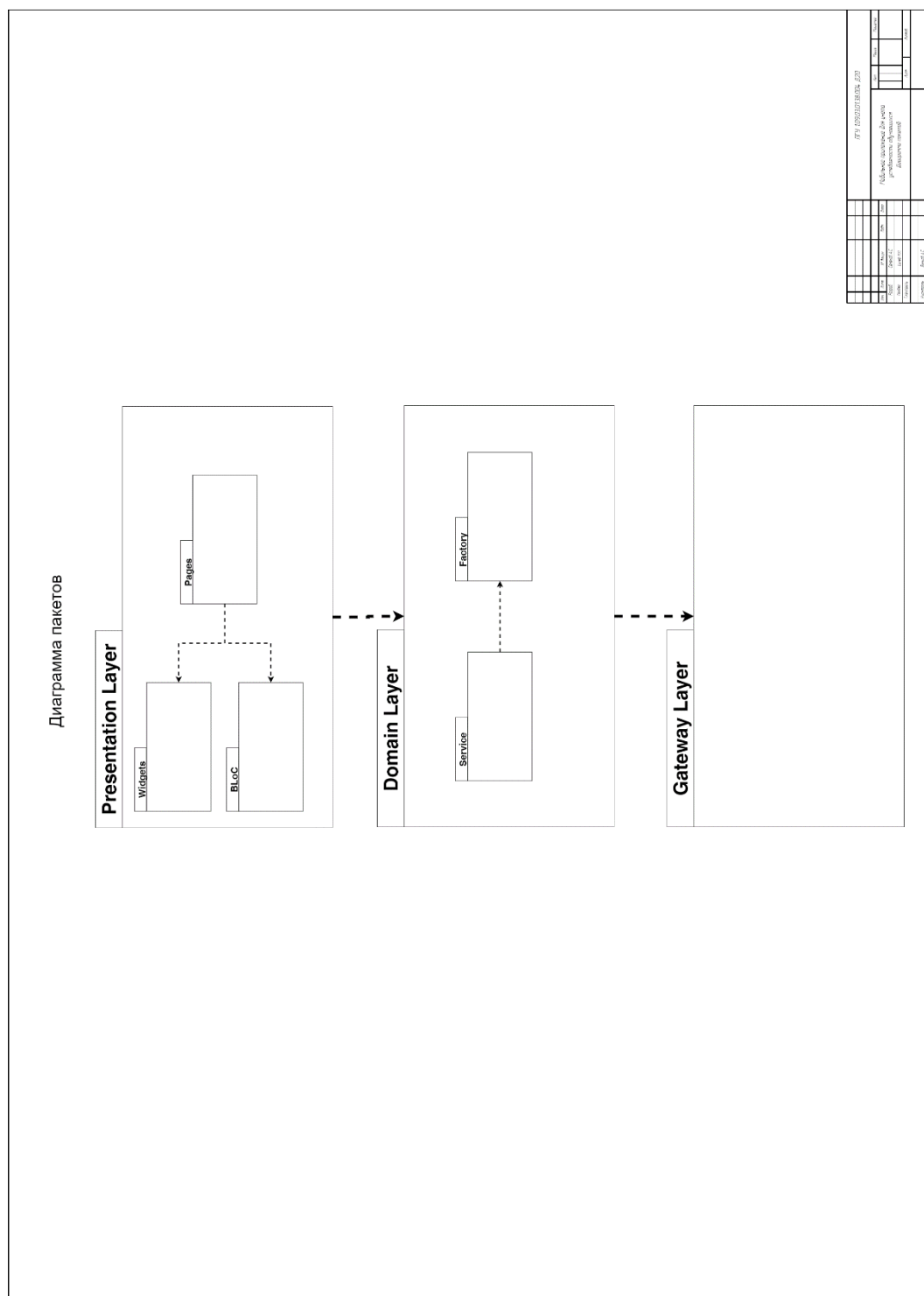


Приложение А.2- Диаграмма классов

Диаграмма классов

[illegible]

Приложение А.3- Диаграмма пакетов

[illegible]

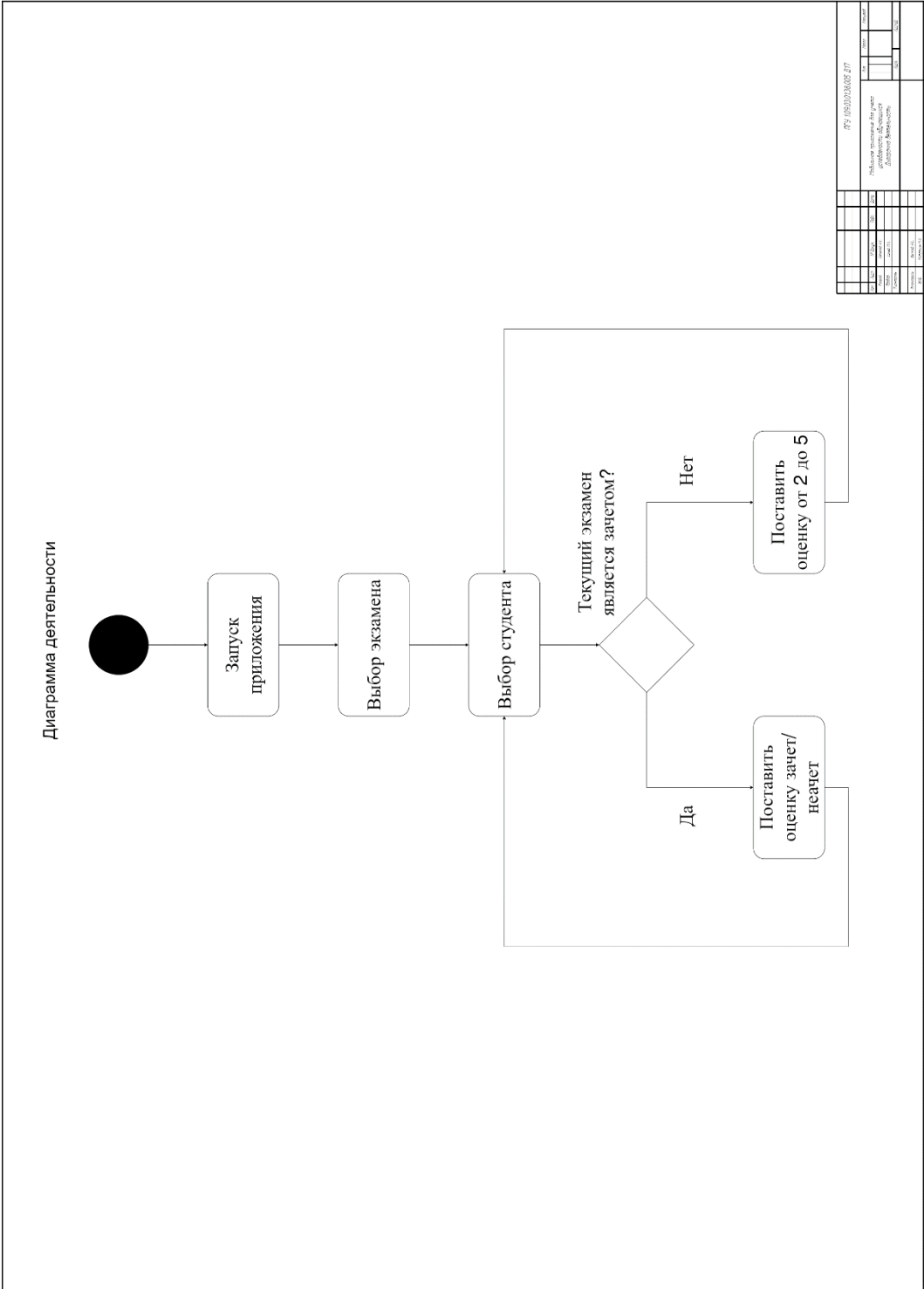
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.38.001 ПЗ

Лист

54

Приложение А.4- Диаграмма деятельности



Приложение Б

(обязательное)

Мобильное приложение для учета успеваемости студентов

Листинг программы

					ПГУ 1.090301.38.001 ПЗ	Лист
						56
Изм.	Лист	№ докум.	Подпись	Дата		

exam_list_page.dart

```
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:get_it/get_it.dart';
import
'package:record_of_student_progress/presentation/bloc/journal/ex
am_bloc.dart';
import
'package:record_of_student_progress/presentation/bloc/journal/ex
am_state.dart';
import
'package:record_of_student_progress/presentation/widgets/base/cu
stom_app_bar_widget.dart';
import
'package:record_of_student_progress/presentation/widgets/base/lo
ading_indicator_widget.dart';
import
'package:record_of_student_progress/presentation/widgets/exams/e
xams_group_widget.dart';

import '../models/exam/exam.dart';
import '../services/exam_service.dart';
import '../bloc/journal/exam_event.dart';

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

  @override
  Widget build(BuildContext context) {
    return BlocProvider(
      create: (context) =>
        ExamBloc(examService: GetIt.instance<ExamService>())
    );
  }
}
```

```

        ..add(LoadExamsEvent()),
        child: const JournalClassFlow());
    }
}

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: const CustomAppBarWidget(title: 'Экзамены'),
      body: BlocBuilder<ExamBloc, ExamState>(builder: (context,
state) {
        if (state is ExamsLoadedState) {
          return Padding(
            padding: const EdgeInsets.only(top: 12),
            child:
              ListView(children:
_buildJournalGroupWidgets(state.exams)));
        } else {
          return const LoadingIndicatorWidget();
        }
      })),
    );
  }

  List<ExamsGroupWidget> _buildJournalGroupWidgets(
    Map<String, List<Exam>> mapExams) {
    List<ExamsGroupWidget> journalGroupWidgets = [];

    var count = 0;
    mapExams.forEach((key, value) {

```

```

        journalGroupWidgets.add(ExamsGroupWidget(
            groupName: key,
            exams: value,
            isExpanded: count == 0));
        count += 1;
    });

    return journalGroupWidgets;
}
}

exam_journal_page.dart

import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:get_it/get_it.dart';
import
'package:record_of_student_progress/presentation/bloc/journal/ex
am_state.dart';
import
'package:record_of_student_progress/presentation/colors.dart';
import
'package:record_of_student_progress/presentation/widgets/base/cu
stom_app_bar_widget.dart';
import
'package:record_of_student_progress/presentation/widgets/base/lo
ading_indicator_widget.dart';
import
'package:record_of_student_progress/presentation/widgets/base/sc
affold_body_item.dart';

import '../models/exam/mark.dart';
import '../services/exam_service.dart';
import '../bloc/journal/exam_bloc.dart';
import '../bloc/journal/exam_event.dart';

```

```

import '../widgets/exams/journal_student_item_widget.dart';

class ExamJournalPage extends StatelessWidget {
  final String groupName;
  final bool isCredit;
  final DateTime date;
  final int groupId;
  final int examId;

  const ExamJournalPage(
    {required this.groupName,
    required this.isCredit,
    required this.date,
    required this.groupId,
    required this.examId,
    Key? key})
    : super(key: key);

  @override
  Widget build(BuildContext context) {
    return BlocProvider(
      create: (context) => ExamBloc(examService:
GetIt.instance<ExamService>())
      ..add(LoadExamJournalEvent(groupId: groupId, examId:
examId)),
      child: _JournalExamPageFlow(
        groupName: groupName, isCredit: isCredit, date: date),
    );
  }
}

class _JournalExamPageFlow extends StatelessWidget {
  final String groupName;

```

```

final bool isCredit;
final DateTime date;

const _JournalExamPageFlow(
  {required this.groupName,
   required this.isCredit,
   required this.date,
   Key? key})
  : super(key: key);

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: CustomAppBarWidget(title: groupName),
    body: BlocBuilder<ExamBloc, ExamState>(
      builder: (context, state) {
        if (state is ExamJournalLoadedState) {
          var studentsWidgets =

_createJournalStudentWidgetsList(state.journalMap);
          return Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Container(
                margin: const EdgeInsets.symmetric(vertical:
10),
                padding: const EdgeInsets.symmetric(
                  vertical: 10, horizontal: 30),
                decoration: BoxDecoration(
                  color: Colors.white,
                  borderRadius: BorderRadius.circular(10)),
                child: Row(*mainAxisSize:
MainAxisSize.min,* children: [

```

```

        Text(
          isCredit ? 'Зачет' : 'Экзамен',
          style: const TextStyle(
            fontSize: 20,
            fontWeight: FontWeight.w700,
            color: AppColors.primary),
        ),
        const Spacer(),
        const Text(
          '03.09.2022',
          style: TextStyle(
            fontSize: 18,
            fontWeight: FontWeight.w400,
            color: AppColors.mainTextColor),
        )
      ]),
    ),
    Expanded(
      child: ScaffoldBodyItem(
        itemMargin: const EdgeInsets.only(bottom:
12),

        child: ListView(children:
studentsWidgets)),
      ),
    ],
  );
}

return const LoadingIndicatorWidget();
},
),
);
}

```

```

List<Widget> _createJournalStudentWidgetsList(Map<String, Mark>
journalMap) {
    List<JournalStudentItemWidget> journalStudentWidgets = [];
    var i = 0;
    JournalStudentItemWidget studentWidget;

    journalMap.forEach((key, value) {
        i++;
        studentWidget = JournalStudentItemWidget(
            index: i, name: key, mark: value, isCredit: isCredit);
        journalStudentWidgets.add(studentWidget);
    });
    return journalStudentWidgets;
}

```

exam_bloc.dart

```

import 'package:flutter_bloc/flutter_bloc.dart';
import
'package:record_of_student_progress/presentation/bloc/journal/ex
am_event.dart';
import
'package:record_of_student_progress/presentation/bloc/journal/ex
am_state.dart';
import
'package:record_of_student_progress/services/exam_service.dart';

class ExamBloc extends Bloc<ExamEvent, ExamState> {
    ExamService examService;

    ExamBloc({required this.examService}) : super(InitialState()) {
        on<LoadExamsEvent>((event, emit) => _loadExams(event, emit));
        on<MarkCreatedUpdatedEvent>((
            event, emit) => _createUpdateMark(event, emit));
    }
}

```

```

        on<LoadExamJournalEvent>((event, emit) =>
_loadExamJournal(event, emit));
    }

    Future<void> _loadExams(ExamEvent event, Emitter<ExamState>
emit) async {
        final exams = await examService.getGroupsExams();
        emit(ExamsLoadedState(exams: exams));
    }

    Future<void> _loadExamJournal(
        LoadExamJournalEvent event, Emitter<ExamState> emit) async
    {
        final journalMap = await examService.getExamJournal(
            groupId: event.groupId, examId: event.examId);
        emit(ExamJournalLoadedState(journalMap: journalMap));
    }

    void _createUpdateMark(MarkCreatedUpdatedEvent event,
Emitter<ExamState> emit) {
        examService.createUpdateMark(event.mark.id, event.mark);
    }
}

```

exam_event.dart

```

import '../.../models/exam/mark.dart';
abstract class ExamEvent {}

class LoadExamsEvent extends ExamEvent {}

class LoadStudentsEvent extends ExamEvent {
    final int groupId;

    LoadStudentsEvent({required this.groupId});
}

```

```

}

class LoadExamJournalEvent extends ExamEvent {
  final int groupId;
  final int examId;

  LoadExamJournalEvent({required this.groupId, required
this.examId});
}

class MarkCreatedUpdatedEvent extends ExamEvent {
  final Mark mark;

  MarkCreatedUpdatedEvent({required this.mark});
}

```

exam_state.dart

```

import '../.../models/exam/exam.dart';
import '../.../models/exam/mark.dart';
import '../.../models/exam/student.dart';

abstract class ExamState {}

class InitialState extends ExamState {}

class ExamsLoadedState extends ExamState {
  final Map<String, List<Exam>> exams;

  ExamsLoadedState({required this.exams});
}

class StudentsLoadedState extends ExamState {
  final List<Student> students;
}

```

```

    StudentsLoadedState({required this.students});
  }

class ExamJournalLoadedState extends ExamState {
  final Map<String, Mark> journalMap;

  ExamJournalLoadedState({required this.journalMap});
}

class MarkCreateUpdateState extends ExamState {}
custom_app_bar_widget.dart
import 'package:flutter/material.dart';

class CustomAppBarWidget extends StatelessWidget
  implements PreferredSizeWidget {
  final String title;
  final double height;
  final bool isHaveBack;

  const CustomAppBarWidget(
    {required this.title,
    this.height = 60,
    this.isHaveBack = false,
    Key? key})
    : super(key: key);

  @override
  Widget build(BuildContext context) {
    return AppBar(
      title: Text(title),
      //actions: [IconButton(onPressed: () {}, icon: const
      Icon(Icons.settings))],
    );
  }

```

```
}
```

```
@override
```

```
Size get preferredSize => Size.fromHeight(height);
```

```
}
```

loading_indicator_widget.dart

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

```
import
```

```
'package:record_of_student_progress/presentation/colors.dart';
```

```
class LoadingIndicatorWidget extends StatelessWidget {
```

```
  const LoadingIndicatorWidget({Key? key}) : super(key: key);
```

```
@override
```

```
Widget build(BuildContext context) {
```

```
  return const Center(
```

```
    child: CircularProgressIndicator(color:
```

```
AppColors.primary));
```

```
  })
```

scaffold_body_item.dart

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

```
class ScaffoldBodyItem extends StatelessWidget {
```

```
  final Widget child;
```

```
  final Color color;
```

```
  final EdgeInsets itemPadding;
```

```
  final EdgeInsets itemMargin;
```

```
  const ScaffoldBodyItem({
```

```
    Key? key,
```

```
    required this.child,
```

```
    this.color = Colors.white,
```

					ПГУ 1.090301.38.001 ПЗ	Лист
						67
Изм.	Лист	№ докум.	Подпись	Дата		

```

        this.itemPadding = const EdgeInsets.symmetric(vertical: 20,
horizontal: 16),
        this.itemMargin = const EdgeInsets.symmetric(vertical: 0,
horizontal: 0),
    }) : super(key: key);

@override
Widget build(BuildContext context) {
    return Container(
        margin: itemMargin,
        child: ClipRRect(
            borderRadius: const
BorderRadius.all(Radius.circular(12.0)),
            child: Container(
                width: double.infinity,
                padding: itemPadding,
                decoration: BoxDecoration(
                    color: color,
                ),
                child: child,
            ),
        ),
    );
}
}

```

cell_with_mark_widget.dart

```

import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:get_it/get_it.dart';
import
'package:record_of_student_progress/models/exam/exam.dart';

```

```

import
'package:record_of_student_progress/presentation/bloc/journal/ex
am_bloc.dart';

import '../models/exam/mark.dart';
import '../services/exam_service.dart';
import '../bloc/journal/exam_event.dart';
import '../colors.dart';

class CellWithMarkWidget extends StatefulWidget {
  Mark mark;
  final String studentName;
  final bool isCredit;

  CellWithMarkWidget(
    {required this.mark,
    required this.studentName,
    required this.isCredit,
    Key? key})
    : super(key: key);

  @override
  State<CellWithMarkWidget> createState() =>
    _CellWithMarkWidgetState(mark: mark.mark);
}

class _CellWithMarkWidgetState extends State<CellWithMarkWidget>
{
  String mark;

  _CellWithMarkWidgetState({required this.mark});

  @override

```

```

Widget build(BuildContext context) {
  return InkWell(
    onTap: () => _showScoringModal(context),
    child: Container(
      height: 40,
      width: 40,
      decoration: BoxDecoration(
        border: Border.all(width: 0.8, color:
AppColors.grayTextColor),
        borderRadius: BorderRadius.circular(4)),
      child: Center(
        child: Text(_getAbbreviationScoreName(),
          style: TextStyle(fontSize: 18, color:
_getMarkColor()))),
    ),
  );
}

```

```

String _getAbbreviationScoreName() {
  if (mark == 'Не зачет') return 'НЗ';
  if (mark == 'Зачет') return 'Зач';
  return mark;
}

```

```

Color? _getMarkColor() {
  if (mark == 'Не зачет' || mark == '2') {
    return Colors.red;
  }
  return Colors.green;
}

```

```

void _showScoringModal(BuildContext context) {
  showModalBottomSheet(

```

					ПГУ 1.090301.38.001 ПЗ	Лист
						70
Изм.	Лист	№ докум.	Подпись	Дата		

```

        context: context,
        isScrollControlled: true,
        shape: const RoundedRectangleBorder(
            borderRadius: BorderRadius.vertical(top:
Radius.circular(10))),
        builder: (_) => Container(
            padding: const EdgeInsets.all(16),
            child: Column(
                mainAxisAlignment: MainAxisAlignment.min,
                children: [
                    Text(widget.studentName,
                        style: const TextStyle(
                            fontSize: 24,
                            fontWeight: FontWeight.w500,
                            color: AppColors.primary)),
                    Padding(
                        padding: const EdgeInsets.only(top: 30, bottom:
60),
                        child: Row(
                            mainAxisAlignment:
MainAxisAlignment.center,
                            children: _buildScoreWidgets(context)),
                        )
                ],
            )),
    }

List<Widget> _buildScoreWidgets(BuildContext context) {
    List<Widget> scoreWidgets = [];
    if (!widget.isCredit) {
        for (var i = 1; i < 5; i++) {
            var s = (i + 1).toString();

```

```

        var scoreWidget = _buildScoreWidget(s, s == mark,
context);
        scoreWidgets.add(scoreWidget);
    }
} else {
    var success = _buildScoreWidget('Зачет', mark == 'Зачет',
context);
    var failure = _buildScoreWidget('Не зачет', mark == 'Не
зачет', context);
    scoreWidgets.addAll([success, failure]);
}
return scoreWidgets;
}

Widget _buildScoreWidget(
    String score, bool isSelected, BuildContext context) {
    return BlocProvider(
        create: (_) => ExamBloc(examService:
GetIt.instance<ExamService>()),
        child: Builder(builder: (context) {
            var bloc = context.read<ExamBloc>();
            return InkWell(
                onTap: () {
                    widget.mark.mark = score;
                    bloc.add(MarkCreatedUpdatedEvent(mark: widget.mark));
                    setState(() {
                        mark = score;
                    });
                    Navigator.of(context).pop();
                },
                child: Container(
                    height: 50,
                    width: widget.isCredit ? 100 : 50,

```

```

        margin: const EdgeInsets.symmetric(horizontal: 8),
        child: Center(
          child: Text(score,
            style: TextStyle(
              fontSize: 20,
              color: isSelected
                ? Colors.white
                : AppColors.grayTextColor))),
        decoration: BoxDecoration(
          border: Border.all(color:
AppColors.grayTextColor),
          borderRadius: BorderRadius.circular(5),
          color: isSelected ? AppColors.primary :
Colors.transparent),
      ),
    );
  },
);
}
}

```

exam_item_widget.dart

```

import 'package:flutter/material.dart';
import
'package:record_of_student_progress/presentation/colors.dart';

class ExamItemWidget extends StatelessWidget {
  final String name;
  final DateTime date;
  final bool isCredit;
  final String groupName;
  final int groupId;
  final int examId;

```

```

const ExamItemWidget({
  Key? key,
  required this.name,
  required this.date,
  required this.isCredit,
  required this.groupName,
  required this.groupId,
  required this.examId,
}) : super(key: key);

@override
Widget build(BuildContext context) {
  return InkWell(
    onTap: () =>
      Navigator.of(context).pushNamed('/examJournal', arguments: {
        'groupName': groupName,
        'isCredit': isCredit,
        'date': date,
        'groupId': groupId,
        'examId': examId
      }),
    child: Padding(
      padding: const EdgeInsets.symmetric(vertical: 5),
      child: Column(crossAxisAlignment: CrossAxisAlignment.end,
children: [
      Row(children: [
        Container(
          width: 30,
          height: 30,
          child: Center(
            child: Padding(
              padding: const EdgeInsets.all(1),
              child: Text(isCredit ? '3' : '3'),
            ),
          ),
        ),
      ]),
    ),
  ),

```

```

        style: const TextStyle(
          fontSize: 18,
          fontWeight: FontWeight.bold,
          color: Colors.white))),
      decoration: BoxDecoration(
        color: AppColors.primary,
        borderRadius: BorderRadius.circular(300))),
    const SizedBox(width: 10),
    Expanded(
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Row(children: [
            Text(name,
              style: const TextStyle(
                fontSize: 16,
                fontWeight: FontWeight.bold,
                color: AppColors.mainTextColor)),
            const Spacer(),
            const Text('03.09',
              style: TextStyle(
                fontSize: 16, color:
AppColors.grayTextColor)),
          ]),
          Padding(
            padding: const EdgeInsets.only(top: 6),
            child: Text(isCredit ? 'Зачет' : 'Экзамен',
              style: const TextStyle(
                fontSize: 14, color:
AppColors.grayTextColor))),
          ],
        ),
      )
    )

```

```

    ]),
    const SizedBox(width: 300, child: Divider())
  ]),
),
);
}
}

```

journal_student_item_widget.dart

```

import 'package:flutter/material.dart';
import
'package:record_of_student_progress/presentation/colors.dart';
import
'package:record_of_student_progress/presentation/widgets/exams/c
ell_with_mark_widget.dart';

import '../models/exam/mark.dart';

class JournalStudentItemWidget extends StatelessWidget {
  final int index;
  final String name;
  final Mark mark;
  final bool isCredit;

  const JournalStudentItemWidget(
    {required this.index,
    required this.name,
    required this.mark,
    required this.isCredit,
    Key? key})
    : super(key: key);

  @override

```

```

Widget build(BuildContext context) {
  return Padding(
    padding: const EdgeInsets.symmetric(horizontal: 15,
vertical: 5),
    child: Row(children: [
      Text('$index.',
        style: const TextStyle(
          fontSize: 18,
          fontWeight: FontWeight.w700,
          color: AppColors.primary)),
      const SizedBox(width: 15),
      Text(name,
        style: const TextStyle(fontSize: 18, fontWeight:
FontWeight.w400)),
      const Spacer(),
      CellWithMarkWidget(mark: mark, studentName: name,
isCredit: isCredit)
    ]),
  );
}
}

```

colors.dart

```

import 'package:flutter/material.dart';

class AppColors {
  static const Color primary = Color(0xfffff8b00);
  static const Color scaffoldBackground = Color(0xffff4f3f8);
  static const Color mainTextColor = Color(0xff343434);
  static const Color grayTextColor = Color(0xff7B819B);
}

```

themes.dart

```

import 'package:flutter/material.dart';

```

					ПГУ 1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		77

```

import
'package:record_of_student_progress/presentation/colors.dart';

class AppThemes {
  static ThemeData basicTheme() => ThemeData(
    brightness: Brightness.light,
    colorScheme: ColorScheme.fromSeed(
      seedColor: AppColors.primary, primary:
AppColors.primary),
    scaffoldBackgroundColor: AppColors.scaffoldBackground,
  );
}

```

exam_service.dart

```

import '../models/exam/exam.dart';
import '../models/exam/mark.dart';

abstract class ExamService {
  Future<Map<String, List<Exam>>> getGroupsExams();

  Future<Map<String, Mark>> getExamJournal(
    {required int groupId, required int examId});

  Future<void> createUpdateMark(int? markId, Mark mark);}

```

exam_service_impl.dart

```

import
'package:record_of_student_progress/factory/exam_models_factory.
dart';
import
'package:record_of_student_progress/gateway/exam_gateway.dart';
import
'package:record_of_student_progress/models/exam/student.dart';

```

```

import
'package:record_of_student_progress/services/exam_service.dart';

import '../models/exam/exam.dart';
import '../models/exam/mark.dart';

class ExamServiceImpl implements ExamService {
  ExamGateWay examGateWay;
  ExamModelsFactory examModelsFactory;

  ExamServiceImpl({required this.examGateWay, required
this.examModelsFactory});

  @override
  Future<Map<String, List<Exam>>> getGroupsExams() async {
    final groups = await examGateWay.getGroups();
    final exams = await examGateWay.getExams();
    return examModelsFactory.createGroupExamsMap(groups, exams);
  }

  @override
  Future<Map<String, Mark>> getExamJournal(
    {required int groupId, required int examId}) async {
    final groups = await examGateWay.getGroups();
    final marks = await examGateWay.getMarks();
    return examModelsFactory.createStudentMarkMap(
      groups, marks, groupId, examId);
  }

  @override
  Future<void> createUpdateMark(int? markId, Mark mark) async {
    await examGateWay.createUpdateMark(markId, mark);
  }
}

```

exam_models_factory.dart

```
import '../models/exam/exam.dart';
import '../models/exam/group.dart';
import '../models/exam/mark.dart';

abstract class ExamModelsFactory {
  Map<String, List<Exam>> createGroupExamsMap(
    List<Group> groups, List<Exam> exams);

  Map<String, Mark> createStudentMarkMap(
    List<Group> groups, List<Mark> marks, int groupId, int
    examId);
}
```

exam_models_factory_impl.dart

```
import 'package:collection/collection.dart';

import
'package:record_of_student_progress/factory/exam_models_factory.
dart';
import
'package:record_of_student_progress/models/exam/mark.dart';
import
'package:record_of_student_progress/models/exam/student.dart';

import '../models/exam/exam.dart';
import '../models/exam/group.dart';

class ExamModelsFactoryImpl implements ExamModelsFactory {
  @override
  Map<String, List<Exam>> createGroupExamsMap(
    List<Group> groups, List<Exam> exams) {
    Map<String, List<Exam>> groupExamMap = {};
  }
}
```

```

    for (var group in groups) {
        List<Exam> groupExams = [];
        for (var exam in exams) {
            if (exam.groupId == group.id) groupExams.add(exam);
        }
        groupExamMap.addAll({group.name: groupExams});
    }

    return groupExamMap;
}

@Override
Map<String, Mark> createStudentMarkMap(
    List<Group> groups, List<Mark> marks, int groupId, int
examId) {
    Map<String, Mark> studentMarkMap = {};
    final group = groups.firstWhere((group) => group.id ==
groupId);
    final studentsInGroup = group.students;
    final examMarks = _getExamMarks(examId, marks);
    for (var student in studentsInGroup) {
        var studentMark =
            examMarks.firstWhereOrNull((mark) => mark.studentId ==
student.id);
        studentMarkMap[student.name] =
            studentMark ?? Mark(mark: '', examId: examId,
studentId: student.id);
    }
    return studentMarkMap;
}

List<Mark> _getExamMarks(int examId, List<Mark> marks) {
    List<Mark> examMarks = [];

```

```

    for (var mark in marks) {
        if (mark.examId == examId) examMarks.add(mark);
    }
    return examMarks;
}}

```

exam_gateway.dart

```

import '../models/exam/exam.dart';
import '../models/exam/group.dart';
import '../models/exam/mark.dart';

abstract class ExamGateWay {
    Future<List<Group>> getGroups();

    Future<List<Exam>> getExams();

    Future<List<Mark>> getMarks();

    Future<void> createUpdateMark(int? markId, Mark mark);
}

```

exam_gateway_impl.dart

```

import 'dart:math';

import
'package:record_of_student_progress/gateway/exam_gateway.dart';
import
'package:record_of_student_progress/models/exam/mark.dart';
import 'package:record_of_student_progress/utils/hive_db.dart';

import '../models/exam/exam.dart';
import '../models/exam/group.dart';
import '../models/exam/student.dart';

class ExamGatewayImpl implements ExamGateWay {

```

```

@Override
Future<List<Group>> getGroups() async {
    return await _groupAPICallEmulation();
}

@Override
Future<List<Exam>> getExams() async {
    return await _examAPICallEmulation();
}

@Override
Future<List<Mark>> getMarks() async {
    return await HiveDB.getMarks();
}

Future<List<Group>> _groupAPICallEmulation() async {
    var studentId = 0;
    List<Group> groups = [];
    return await Future.delayed(const Duration(seconds: 2), () {
        for (int i = 0; i < 4; i++) {
            List<Student> students = [];
            for (int j = 0; j < 20; j++) {
                var student = Student(id: studentId, name: 'Студент
$studentId');
                studentId++;
                students.add(student);
            }
            var group = Group(id: i, name: 'Группа $i', students:
students);
            groups.add(group);
        }
        return groups;
    });
}

```

```

}

Future<List<Exam>> _examAPICallEmulation() async {
  List<Exam> exams = [];
  var examId = 0;
  return await Future.delayed(const Duration(seconds: 2), () {
    for (int i = 0; i < 4; i++) {
      for (int j = 0; j < 2; j++) {
        var date = DateTime.now();
        if (j % 2 == 0) date = date.add(Duration(days:
Random().nextInt(20)));

        exams.add(Exam(
          id: examId,
          name: 'Экзамен $examId',
          isCredit: j % 2 == 0,
          date: date,
          groupId: i));
        examId++;
      }
    }
    return exams;
  });
}

@override
Future<void> createUpdateMark(int? markId, Mark mark) async {
  await HiveDB.createUpdateMark(markId, mark);
}

```

exam.dart

```

class Exam {
  int id;
  String name;

```

```

bool isCredit;
DateTime date;
int groupId;

Exam(
    {required this.id,
    required this.name,
    required this.isCredit,
    required this.date,
    required this.groupId});
}

```

group.dart

```

import
'package:record_of_student_progress/models/exam/student.dart';

class Group {
    int id;
    String name;
    List<Student> students;

    Group({required this.id, required this.name, required
this.students});
}

```

mark.dart

```

import 'package:hive/hive.dart';

part 'mark.g.dart';

@HiveType(typeId: 0)
class Mark {
    @HiveField(0)
    int? id;
}

```

```

@HiveField(1)
String mark;

@HiveField(2)
final int examId;

@HiveField(3)
final int studentId;

Mark({required this.mark, required this.examId, required
this.studentId});
}

```

mark.g.dart

```

class MarkAdapter extends TypeAdapter<Mark> {
  @override
  final int typeId = 0;

  @override
  Mark read(BinaryReader reader) {
    final numOfFields = reader.readByte();
    final fields = <int, dynamic>{
      for (int i = 0; i < numOfFields; i++) reader.readByte():
reader.read(),
    };
    return Mark(
      mark: fields[1] as String,
      examId: fields[2] as int,
      studentId: fields[3] as int,
    )..id = fields[0] as int?;
  }

  @override
  void write(BinaryWriter writer, Mark obj) {

```

```

writer
  ..writeByte(4)
  ..writeByte(0)
  ..write(obj.id)
  ..writeByte(1)
  ..write(obj.mark)
  ..writeByte(2)
  ..write(obj.examId)
  ..writeByte(3)
  ..write(obj.studentId);
}

@override
int get hashCode => typeId.hashCode;

@override
bool operator ==(Object other) =>
  identical(this, other) ||
  other is MarkAdapter &&
    runtimeType == other.runtimeType &&
    typeId == other.typeId;
}

```

student.dart

```

class Student {
  int id;
  String name;

  Student({required this.id, required this.name});
}

```

hive_db.dart

```

import 'package:hive/hive.dart';
import 'package:path_provider/path_provider.dart';

```

```

import '../models/exam/mark.dart';

class HiveDB {
  static const lastMarkIdBoxName = 'lastMakId';
  static const marksBoxName = 'Marks';

  static Future<void> init() async {
    final dir = await getApplicationDocumentsDirectory();
    Hive.init(dir.path);
    Hive.registerAdapter(MarkAdapter());
  }

  static Future<List<Mark>> getMarks() async {
    final marksBox = await Hive.openBox<Mark>(marksBoxName);
    var marks = marksBox.values.toList();
    marksBox.compact();
    marksBox.close();
    return marks;
  }

  static Future<void> createUpdateMark(int? markId, Mark mark)
  async {
    final lastMarkIdBox = await
Hive.openBox<dynamic>(lastMarkIdBoxName);
    final marksBox = await Hive.openBox<dynamic>(marksBoxName);
    if (markId == null) {
      var lastMarkId = lastMarkIdBox.get('lastMarkId');
      if (lastMarkId == null) {
        lastMarkId = 0;
      } else {
        lastMarkId++;
      }
      lastMarkIdBox.put('lastMarkId', lastMarkId);
    }
  }
}

```

```

        mark.id = lastMarkId;
        markId = lastMarkId;
    }
    marksBox.put('$markId', mark);
    lastMarkIdBox.compact();
    lastMarkIdBox.close();
    marksBox.compact();
    marksBox.close();
  }}

```

route_generator.dart

```

import 'package:flutter/material.dart';
import
'package:record_of_student_progress/presentation/pages/exam_journal_page.dart';
import
'package:record_of_student_progress/presentation/pages/exams_list_page.dart';

class RouteGenerator {
  static Route<dynamic> generateRoute(RouteSettings settings) {
    WidgetBuilder builder;
    final args = settings.arguments as Map<String, dynamic>?;
    switch (settings.name) {
      case '/examsList':
        builder = (BuildContext context) => const
ExamsListPage();
        break;
      case '/examJournal':
        builder = (BuildContext context) => ExamJournalPage(
          groupName: args!['groupName'],
          isCredit: args['isCredit'],
          date: args['date'],
          groupId: args['groupId'],

```

```

        examId: args['examId']));
    break;
  default:
    throw Exception('Invalid route: ${settings.name}');
  }
  return MaterialPageRoute(builder: builder, settings:
settings);
}}

```

dependencies.dart

```

import 'package:get_it/get_it.dart';
import
'package:record_of_student_progress/factory/impl/exam_models_fac
tory_impl.dart';
import
'package:record_of_student_progress/factory/exam_models_factory.
dart';
import
'package:record_of_student_progress/gateway/impl/exam_gateway_im
pl.dart';
import
'package:record_of_student_progress/gateway/exam_gateway.dart';
import
'package:record_of_student_progress/services/impl/exam_service_i
mpl.dart';
import
'package:record_of_student_progress/services/exam_service.dart';
setupDependencies() async {
  final serviceLocator = GetIt.instance;
  serviceLocator
    .registerLazySingleton<ExamGateWay>(() =>
ExamGatewayImpl());

  serviceLocator

```

```

        .registerFactory<ExamModelsFactory>(() =>
ExamModelsFactoryImpl());
    serviceLocator.registerLazySingleton<ExamService>(() =>
ExamServiceImpl(
    examGateWay: serviceLocator<ExamGateWay>(),
    examModelsFactory: serviceLocator<ExamModelsFactory>()));}

```

main.dart

```

import 'package:flutter/material.dart';
import 'package:hive/hive.dart';
import 'package:record_of_student_progress/dependencies.dart';
import
'package:record_of_student_progress/presentation/pages/exams_list_page.dart';
import
'package:record_of_student_progress/presentation/themes.dart';
import 'package:record_of_student_progress/utils/hive_db.dart';
import
'package:record_of_student_progress/utils/route_generator.dart';
void main() async {
    WidgetsFlutterBinding.ensureInitialized();
    await HiveDB.init();
    runApp(const MyApp());
}
class MyApp extends StatelessWidget {
    const MyApp({Key? key}) : super(key: key);
    @override
    Widget build(BuildContext context) {
        setupDependencies();
        return MaterialApp(
            onGenerateRoute: RouteGenerator.generateRoute,
            theme: AppThemes.basicTheme(),
            home: const ExamsListPage(),
        );
    }
}

```

Приложение В

(обязательное)

Мобильное приложение для учета успеваемости студентов

Презентация

Мобильное приложение для учета успеваемости обучающихся

Автор работы: студент группы 18BB1 Семенов А.С.

Научный руководитель: к.т.н., доцент кафедры ВТ Синев М..

Пенза, 2022

Актуальность
проекта

В связи с ограничениями, которые были установлены из-за пандемии COVID-19, подавляющее большинство высших учебных заведений России было вынуждено перейти на дистанционный формат обучения. В соответствии с приказом министерства образования и науки Российской Федерации от 23.08.2017 № 816 "Об утверждении Порядка применения организациями осуществляющими образовательную деятельность, электронного обучения, дистанционных образовательных технологий при реализации образовательных программ", а именно в соответствии с шестым пунктом, прилагаемого порядка, если учебная организация предоставляет полностью дистанционные услуги обучения, она обязана обеспечить функционирование электронной информационнообразовательной среды

Данный проект предполагает интеграцию мобильного приложения в ЭИОС учебного заведения, благодаря чему повысится мобильность процесса учета успеваемости студентов.

					ПГУ 1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		92

Цели работы

- изучение современных инструментов для создания мобильных приложений;
- изучение компонентов для создания мобильного ПО;
- разработка приложения, которое позволит оценивать успеваемость студентов с помощью мобильного устройства.

Приложение «Журнал преподавателя»

SJournal

ГРУППА

ДИСЦИПЛИНА

ВИД ЗАНЯТИЯ

НОВОЕ ЗАНЯТИЕ

РЕЗУЛЬТАТ

Группа

Дисциплина

Вид занятия

АПЗМА

МНЗМА

МП

НДЗМА

ОПЗ

ТОЗ 34

Методика преподавания ОПЗМА

Методика преподавания ОПЗМА

Методика преподавания ОПЗМА

Методика преподавания ОПЗМА

Методика преподавания ОПЗМА

Методика преподавания ОПЗМА

ТОЗ 34

Методика преподавания ОПЗМА

ТРЗМА

Добавить

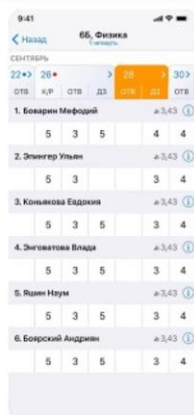
Методика преподавания ОПЗМА

АТ-11-1 / ТОЗ 34 - Теоретические основы электротехники / Лб

№	И.О.	Лб 1 24.09.2013	Лб 2 01.10.2013	Занятие Лб 3 08.10.2013	Занятие Лб 4 15.10.2013	Лб 5 22.10.2013	Лб 6 29.10.2013	Лб 7 05.11.2013	Лб 8 12.11.2013	Лб 9 19.11.2013	Лб 10 26.11.2013
1	Белый Л.Г.			4	4						
2	Богданов С.С.			4	4	Н	Н	Н			Н
3	Буц В.В.	Н	Н	4	4	Н	Н	Н	Н	Н	Н
4	Волошин А.Ю.		Н	4	4		Н		Н	Н	Н
5	Иванов И.И.		Н	4	4				Н	Н	
6	Крамарев В.С.	Н	Н	4	4	Н	Н	Н	Н	Н	Н
7	Кулинич А.Ю.			4	4				Н		
8	Петров Д.П.			4	4	6	Н		Н		
9	Потапов Ю.В.			4	4	Н	Н	Н			Н

9:35

Приложение «Журнал дневник»



Постановка задачи

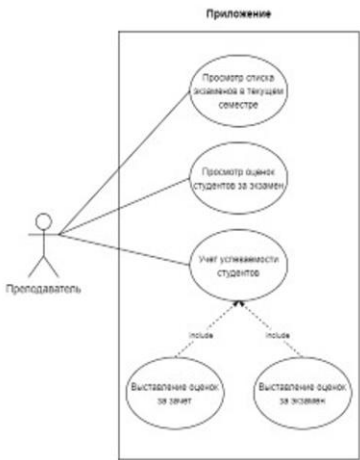


Разрабатываемое приложение должно стабильно работать на операционных системах Android и IOS, обладать современным, интуитивно понятным интерфейсом и иметь возможность взаимодействия с сервером.



Необходимый функционал: просмотр списка экзаменов, просмотр текущих оценок студентов за экзамен, пользователь приложения должен иметь возможность вести учет успеваемости студентов.

UML-диаграмма прецедентов разрабатываемого приложения



Сравнение производительности Flutter и React Native



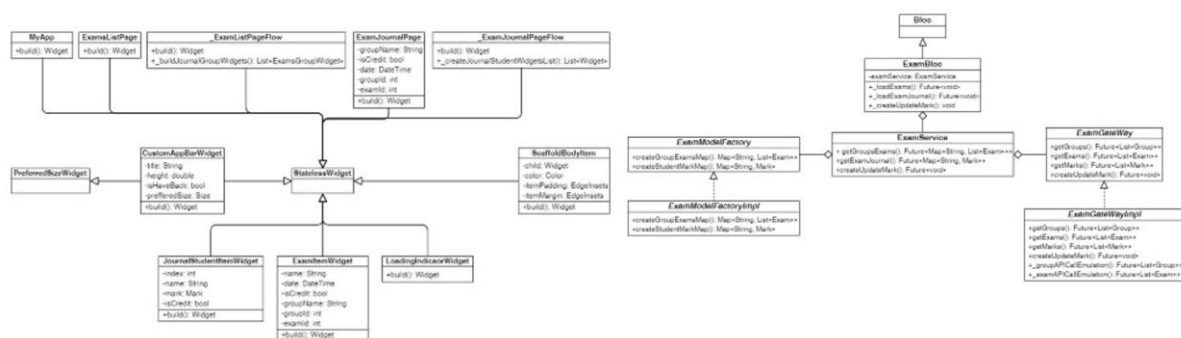
Тестовое приложение

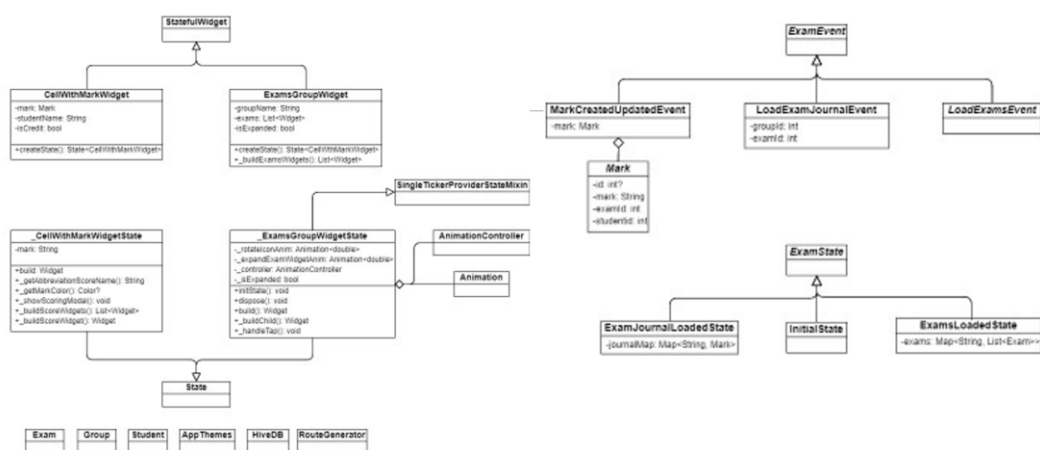
	FPS	CPU	Memory
React Native	7 к/с	8.5%	424мб
Flutter	19 к/с	10.28%	168мб

Результат тестирования

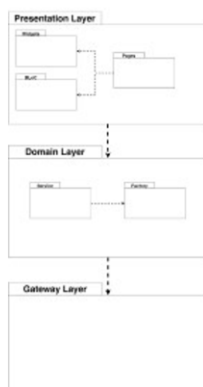
Flutter Dart Android Studio™ hive

UML-диаграмма классов

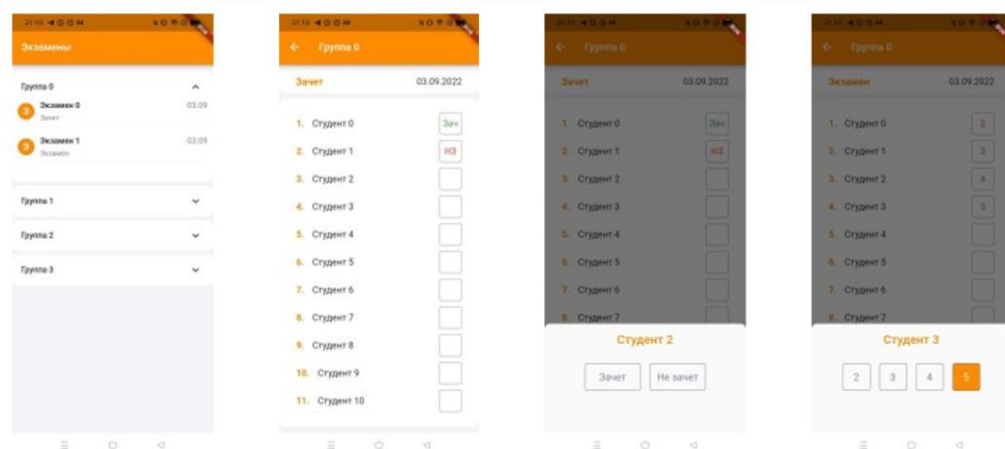




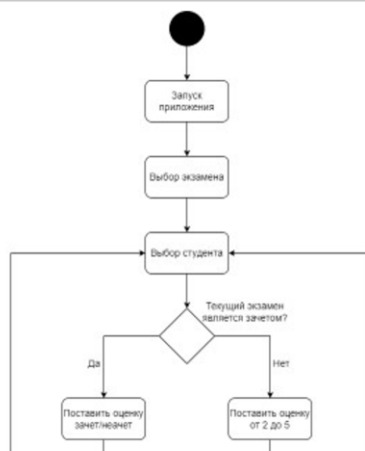
UML- диаграмма пакетов



Работа приложения



UML- диаграмма деятельности выставления оценки за зачет или экзамен



Заключение

В процессе работы над данным проектом:

- была подтверждена актуальность проекта;
- был проведен анализ целевой аудитории, ее потребностей, на основе чего был определен основной функционал приложения;
- были исследованы аналоги проекта, благодаря чему были исключены их недостатки и позаимствованы преимущества;
- в рамках выбора инструментов разработки были рассмотрены два популярных фреймворка для кроссплатформенной мобильной разработки: Flutter и React Native. На основании их сравнения первый был выбран для разработки данного проекта;
- была продумана архитектура приложения и классы;
- было разработано приложение.

Итоговое приложение работает исправно и выполняет все предназначенные ему задачи. В перспективе приложение может расширить спектр своих возможностей. Планируется добавить функционал авторизации пользователей, учета сдачи лабораторных и курсовых работ, выставление промежуточных контрольных точек .

Спасибо за внимание!

					ПГУ 1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		99