

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

Вычислительной техники

Кафедра

Вычислительная техника

Направление подготовки 09.03.01 «Информатика и вычислительная техника»

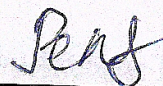
Профиль «Вычислительные машины, комплексы, системы и сети»

БАКАЛАВРСКАЯ РАБОТА

на тему

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ ВЕДЕНИЯ
ДНЕВНИКА
ЗАБОЛЕВШЕГО

Студент



(подпись, дата)

Репин И.А.

31.05.22

Руководитель



(подпись, дата)

Митрохин М.А.

31.05.22

Нормоконтролёр



(подпись, дата)

А.С. Бычков

31.05.22

Работа допущена к защите (протокол заседания кафедры от

02.06.2022 г. № 13)

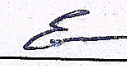
Заведующий кафедрой



М.А. Митрохин

Работа защищена с отметкой  (подпись) (протокол заседания ГЭК от 15.06.22 № 12)

Секретарь ГЭК



(подпись)

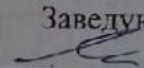
Гурин Е.И.

Пенза, 2022

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

Факультет
вычислительной техники

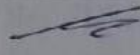
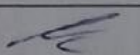
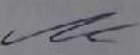
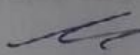
Кафедра
«Вычислительная техника»

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

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Репин Илья Андреевич, гр.18ВВ1 факультета вычислительной техники.
2. Тема работы: «Мобильное приложение ведения дневника заболевшего»
Тема утверждена приказом ПГУ № 357/о от "26" апреля 2022 г.
3. Руководитель бакалаврской работы д.т.н., профессор, зав. каф. ВТ ПГУ Митрохин М.А.
4. Задание на работу (назначение разработки, исходные данные и т.п.)
Мобильное приложение предназначено для самодиагностики и отслеживания динамики течения заболевания.
5. Перечень подлежащих разработке вопросов
Требуется произвести анализ предметной области, разработать архитектуру приложения, выбрать инструменты разработки, спроектировать интерфейс приложения, реализовать приложение с помощью выбранных средств и инструментов.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	50	26.04.2022-30.04.2022	
2.	Анализ требований	10	01.05.2022-02.05.2022	
3.	Выбор технологий, применяемых при проектировании и разработке	50	03.05.2022-07.05.2022	
4.	Описание алгоритмов	40	08.05.2022-11.05.2022	
5.	Разработка приложения «Система извлечения и анализа текстовых и графических данных из социальной сети ВКонтакте для образовательного учреждения»	150	12.05.2022-29.05.2022	
6.	Оформление пояснительной записки	24	30.05.2022-31.05.2022	

Дата выдачи задания "26" апреля 2022 г.

Руководитель бакалаврской работы  26.04.22 Митрохин М.А.
(подпись, дата)

Задание к исполнению принял студент  26.04.22 Репин И.А.
(подпись, дата)

Работу к защите допустить
Декан факультета  14.06.2022 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 80 листов, 38 рисунков, 6 источников, 3 приложения.

ANDROID, БАЗА ДАННЫХ, ИНФОРМАЦИОННАЯ СИСТЕМА, ЛОКАЛЬНОЕ ПРИЛОЖЕНИЕ, СЕРВЕРНОЕ ПРИЛОЖЕНИЕ.

Целью дипломной работы является разработка приложения для ведения дневника заболевшего, предназначенное для самодиагностики и отслеживания динамики развития заболевания. Это достигается путем сбора данных с устройства в процессе прохождения опроса и занесение этих данных *Firestore RealtimeDatabase* для хранения и дальнейшего сбора статистики.

Объект разработки – приложение для ведения дневника заболевшего.

В результате проведенной работы создан проект, протестированы программные средства, удовлетворяющие поставленной задаче.

В пояснительной записке приведен анализ предметной области, обзор технологии разработки и отладки программы.

Графическая часть содержит *UML*-диаграммы программы.

					ПГУ 1.090301.33.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Резин И.А.					
Провер.		Митрохин М.А.					
Контр.		Бычков А.С.					
Утверд.		Митрохин М.А.					
					Лит.	Лист	Листов
						2	80
					ФВТ, каф. ВТ		
					гр. 18ВВ1		

Содержание

Перечень сокращений и обозначений	5
Введение.....	6
1 Мобильное приложение ведения дневника заболевшего	7
1.1 Анализ предметной области	7
1.1.1 Описание предметной области	7
1.1.2 Обзор существующих решений	7
1.1.3 Выбор инструментов разработки	10
1.2 Архитектура приложения.....	13
1.3 Реализация приложения	17
1.3.1 Реализация аутентификации пользователей.....	17
1.3.2 Реализация БД для хранения информации.....	20
1.3.3 Реализация опросника для самодиагностики.....	21
1.3.4 Интерфейс приложения	26
1.4 Руководство пользователя	26
2 Безопасность жизнедеятельности	34
2.1 Опасные факторы при работе с ПЭВМ.....	34
2.2 Классификация помещения.....	35
2.3 Освещение	36
2.4 Расчет освещенности рабочего места	39
Заключение	41
Список используемых источников	42

Приложение А– <i>UML</i> -диаграммы.....	43
Приложение Б – Листинг программы.....	48
Приложение В – Презентация.....	75

					ПГУ 1.090301.33.001 ПЗ	Лист
						4
Изм.	Лист	№ докум.	Подпись	Дата		

Перечень сокращений и обозначений

API (Application Programming Interface) – программный интерфейс приложения

ID (Identifier) - идентификатор

IDE (Integrated Development Environment) – интегрированная среда разработки

MVVM (Model-View-ViewModel) – «Модель-Представление-Контроллер» или «Модель-Вид-Контроллер»

UML (Unified Modeling Language) – унифицирован язык моделирования

UID (User Identifier) – идентификатор пользователя

XML (eXtensible Markup Language) – расширяемый язык разметки

БД – база данных

ОС – операционная система

ПЭВМ – персональная электронно-вычислительная машина

ПК – персональный компьютер

СУБД – система управления базами данных

					ПГУ 1.090301.33.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

Мобильные приложения — одно из самых перспективных направлений разработки на сегодняшний день, поскольку смартфоны уже стали неотъемлемой частью жизни современного человека. Главным достоинством этих приложений является мобильность, пользователь может получить доступ к приложению в любом месте и в любое время, главное чтобы телефон был под рукой. Благодаря этому данная технология получила широкое распространение в мире. Ниши, в которых работают мобильные приложения, разнообразны: игры, офисные программы, интернет-торговля и здоровье.

Актуальность данной темы объясняется тем, что уже на протяжении нескольких лет весь мир страдает из-за эпидемии коронавируса. Мобильное приложение для отслеживания статистики заболеваемости и самодиагностики позволило бы снизить количество запросов в медицинские учреждения, снизив тем самым нагрузку на систему здравоохранения.

					ПГУ 1.090301.33.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

1 Мобильное приложение ведения дневника заболевшего

1.1 Анализ предметной области

1.1.1 Описание предметной области

Создаваемое приложение предназначено для самодиагностики и отслеживания динамики течения коронавируса. Пользователь может пройти опрос для выявления тяжести симптомов его заболевания, посмотреть историю его протекания и различную полезную информацию. На основе истории формируется статистика динамики протекания заболевания. Для сохранения информации в БД и аутентификации, приложение использует интернет.

1.1.2 Обзор существующих решений

Перед тем как начинать создавать собственный проект, было решено изучить аналогичные приложения, чтобы посмотреть как с этой задачей справились другие разработчики. Таким образом были выделены слабые и сильные стороны существующих приложений для того, чтобы улучшить будущий продукт.

В качестве аналогов будущего проекта были выбраны приложения «Symptomate» и «Госуслуги.COVID трекер».

Приложение «Symptomate» встречает пользователя простым интерфейсом, буквально в пару кликов пользователь может узнать наиболее вероятный диагноз, ответив на пару вопросов. Интерфейс приложения представлен ниже на рисунках 1-3.

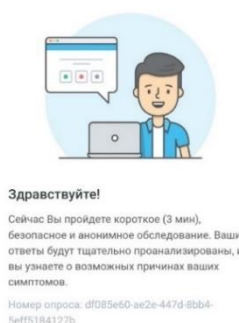


Рисунок 1 - Стартовая страница «Symptomate»

Symptomate

Обследование

Имеются ли у Вас какие-либо из следующих симптомов?

- ☒ Затрудненное дыхание
- ☐ Свистящий звук во время дыхания
- ☒ Патологическая усталость
- ☐ Откашливание крови
- ☐ Повышение температуры тела
- ☐ Насморк

[< Назад](#) [Далее](#)

Рисунок 2 – Опрос симптомов в «Symptomate»

Symptomate

Результаты

Рекомендация

Обратитесь в ближайшее отделение неотложной помощи

Имеющиеся у Вас симптомы вызывают серьезные опасения. Возможно, Вам необходима срочная медицинская помощь. Если у Вас нет возможности обратиться в отделение экстренной медицинской помощи, вызовите скорую помощь.

Тревожные симптомы:

- Внезапное ухудшение одышки
- Кашель в течение менее 3 недель
- Затрудненное дыхание
- Повышение температуры тела

Рисунок 3 – Результат опроса в «Symptomate»

К достоинствам «Symptomate» можно отнести:

- удобный интерфейс;
- возможность быстро узнать стоит обращаться в больницу или нет.

Недостатками данного приложения являются:

- невозможность отслеживания динамики состояния пользователя;
- отсутствие полезной информации касательно коронавируса.

Другое приложение под названием «Госуслуги.COVID трекер», встречает пользователя простым интерфейсом, который представлен на рисунках 4,5.



Рисунок 4 – Возможные контакты в «Госуслуги.COVID трекер»

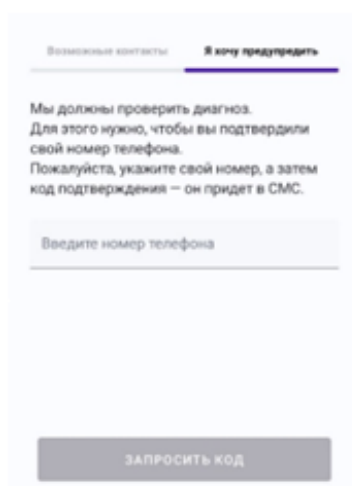


Рисунок 5 – Интерфейс, позволяющий предупредить других о том, что пользователь заболел «Госуслуги.COVID трекер»

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.33.001 ПЗ

Лист

9

Достоинствами приложения «Госуслуги.COVID трекер» являются:

- возможность получить уведомление, в случае если пользователь контактировал с носителем вируса;
- заболевший может предупредить других пользователей о том, что он заразился вирусом.

Слабыми сторонами данного приложения являются:

- отсутствие возможности понять стоит ли обращаться в больницу;
- отсутствие полезной информации касательно коронавируса;
- отсутствие возможности следить за динамикой состояния пользователя;

1.1.3 Выбор инструментов разработки

На данный момент в мире информационных технологий существует огромное количество различных средств разработки приложений на *Android*.

На сегодняшний день самыми востребованными средами являются:

- *Eclipse*;
- *IntelliJ Idea*;
- *Android Studio*.

Каждая *IDE* обладает уникальными характеристиками. О них подробно ниже.

Eclipse — это бесплатная *IDE*, разработанная некоммерческой компанией *Eclipse Foundation*. Эта программа является базой, которая регулирует процессы создания приложений.

К достоинствам «*Eclipse*» можно отнести:

- наличие документации на грамотном русском языке;
- отлично работает на компьютерах с низкой производительностью;
- может подключаться к модулям;
- удобный интерфейс;
- имеет дополнительные функции (для серверной работы и анализа базы данных).

Изм.	Лист	№ докум.	Подпись	Дата

Главным недостатком Эклипс является то, что устарела, и в 2016 году в *Google* отказались от поддержки ее плагинов для *Android*-разработки.

IntelliJ Idea разработала российская компания *JetBrains*. Подобно *Eclipse*, эта среда позволяет создавать приложения и программы на нескольких программных языках. Также *IntelliJ Idea* не перегружает ПК.

Сильными сторонами *IntelliJ Idea* являются:

- более оперативная отладка значений;
- присутствие автозаполнителя методов;
- наличие рефакторинга;
- понятный и лаконичный интерфейс;
- подходит для тех, кто программирует на *Java*.

Огромнейший недостаток *IntelliJ Idea* – её модель распространения. Для возможности использования всего функционала за данную среду разработки нужно заплатить.

Android Studio – интегрированная среда разработки для работы с платформой *Android*, разработанная *Google*. Она являет собой набор интегрированных инструментов для эффективной разработки, отладки и тестирования программ, позволяя создать проект любой сложности, используя стандартные шаблоны *Activity* и наборы все возможных элементов (*Palette*), а так же позволяет эмулировать (клонировать) устройство на базе ОС *Android* (от планшета до мобильного телефона). Это достаточно удобно, так как можно увидеть, как будет выглядеть программа на разных устройствах. Самым большим преимуществом *Android Studio* является её распространённость. Почти все *Android*-девелоперы используют *Android Studio*. Благодаря этому можно без труда найти интересующую по данной среде информацию в интернете.

Единственным недостатком *Android Studio* является её ресурсозатратность. Для работы на *Android Studio* нужен мощный ПК.

Однако достоинств у *Android Studio* гораздо больше:

- наличие удобного *XML* редактора;
- поддержка системы контроля версий;
- возможность эмуляции различных устройств;
- поддержка рендера средствами *GPU*;
- обширная база примеров проектирования (*Samples Browser*).

Проанализировав достоинства и недостатки представленных сред разработки, было принято решение использовать *Android Studio*.

Поскольку основной задачей приложения является сбор и хранение информации, было необходимо найти удобную и безопасную БД для её хранения. *Firebase* отвечает всем этим требованиям.

Особенности *Firebase*:

- он предоставляет облачный сервис, поэтому никаких настроек не требуется;
- данные хранятся в собственном формате *JSON*, поэтому пользователи могут видеть, что они сохранили;
- данные в безопасности, потому что *Firebase* требует 2048-битное шифрование *SSL* для всех передач данных;
- данные отражаются и сохраняются в нескольких безопасных местах, поэтому вероятность потери данных практически отсутствует;
- он прекрасно интегрируется с фреймворками. Так что это позволяет создать приложение в очень короткое время;
- возможность интеграции с авторизацией пользователей, что позволяет более удобно, а главное безопасно хранить данные;
- хранение файлов при поддержке *Google Cloud Storage*.

Так же для отслеживания статистики по заболеваемости в мире будет использоваться *API disease.sh*.

Его преимуществами по сравнению с аналогичными *API*:

- все функции бесплатны;

- данные берутся сразу из нескольких верифицированных источников;
- данные обновляются сразу же как обновляется источник;
- информация предоставляется в формате *JSON*.;

1.2 Архитектура приложения

Выбор хорошей архитектуры приложения невероятно важная вещь, поскольку правильно созданная архитектура при необходимости позволяет вносить изменения с минимальными затратами ресурсов.

При выборе архитектуры приложения было решено воспользоваться *MVVM (Model-View-ViewModel)*. Её концепция заключается в том, что пользовательский интерфейс, логика представления данных приложения и управляющая логика делятся на три отдельных компонента.

Разработанная архитектура будущего приложения представлена на рисунке 6.

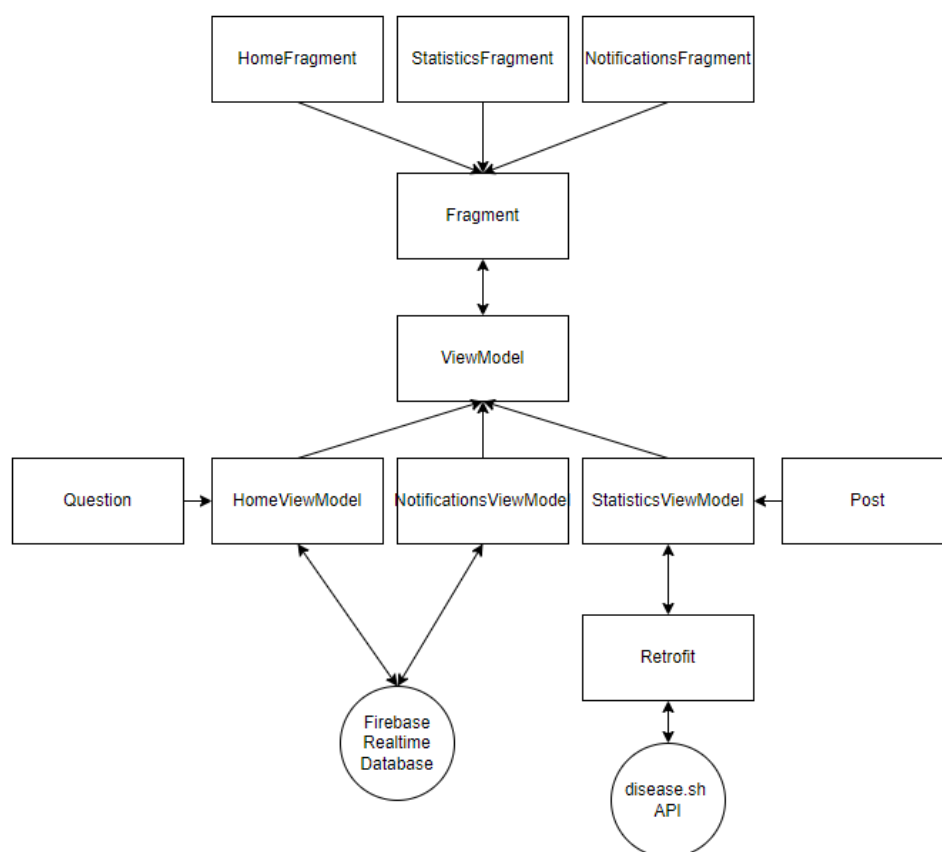


Рисунок 6 – Архитектура приложения

Fragment. Отвечает за взаимодействие между пользователем и приложением. Данный элемент отображает данные, полученные от *ViewModel*, или собирает информацию о действиях пользователя и передаёт их в *ViewModel*.

Fragment состоит из:

- *DiaryFragment* – отвечает за взаимодействия пользователя с БД и добавление новых записей, прохождение опросника. Для удобства хранения вопросов был написан класс *Question*;

- *InfoFragment* – отвечает за получение информации из *API* через библиотеку *Retrofit*;

- *ProfileFragment* – используется для отслеживания динамики заболевания пользователя.

ViewModel. Класс, который будет использоваться в качестве посредника между *Fragment* и *Firebase Database* или веб-сервисами для передачи данных.

Firebase Realtime Database. В данной БД будут храниться данные о состоянии пользователя, его учётной записи и прочая информация.

Disease.sh API. Используется в качестве источника информации для сбора статистики по заболеваемости в РФ. Информация извлекается путём запроса к *API* через библиотеку *Retrofit*. Для получения информации в нужном формате был написан класс *Post*.

Uml-диаграмма классов разработанного приложения представлена на рисунке 7.

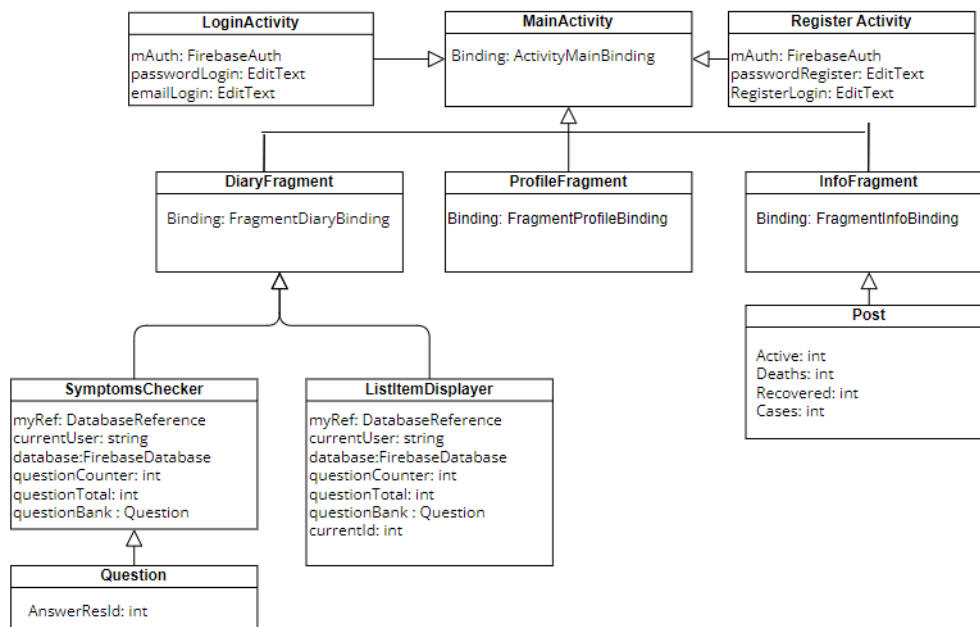


Рисунок 7 – UML-диаграмма классов

Разбираем данную диаграмму:

- *SymptomsChecker* – класс, который используется для опроса пользователя на наличие определённых симптомов. Данный класс хранит в себе результат опроса пользователя и использует функции *ResultParser()* для обработки результата опроса пользователя и внесения полученной информации в БД;
- *ListItemDisplayer* – класс, который используется для отображения конкретного результата опроса, выбранного пользователем. Данный класс хранит в себе *ID* выбранного элемента и с помощью метода *getValue()* возвращает выбранный элемент из БД;
- *LoginActivity*, *RegisterActivity* – классы, использующиеся для авторизации или регистрации пользователя в системе. Хранят в себе *email* и пароль пользователя, и с помощью метода *signInWithEmailAndPassword()* авторизует пользователя в системе;
- *MainActivity* – класс, который используется для отображения и смены фрагментов через *bottomNavigationBar*.

Остальные классы были описаны выше.

Изм.	Лист	№ докум.	Подпись	Дата

Для того, чтобы использовать приложение, пользователь должен авторизоваться, затем он сможет добавлять записи и отслеживать свою динамику течения заболевания, что представлено на рисунке 8 для *UML*-диаграммы вариантов использования.

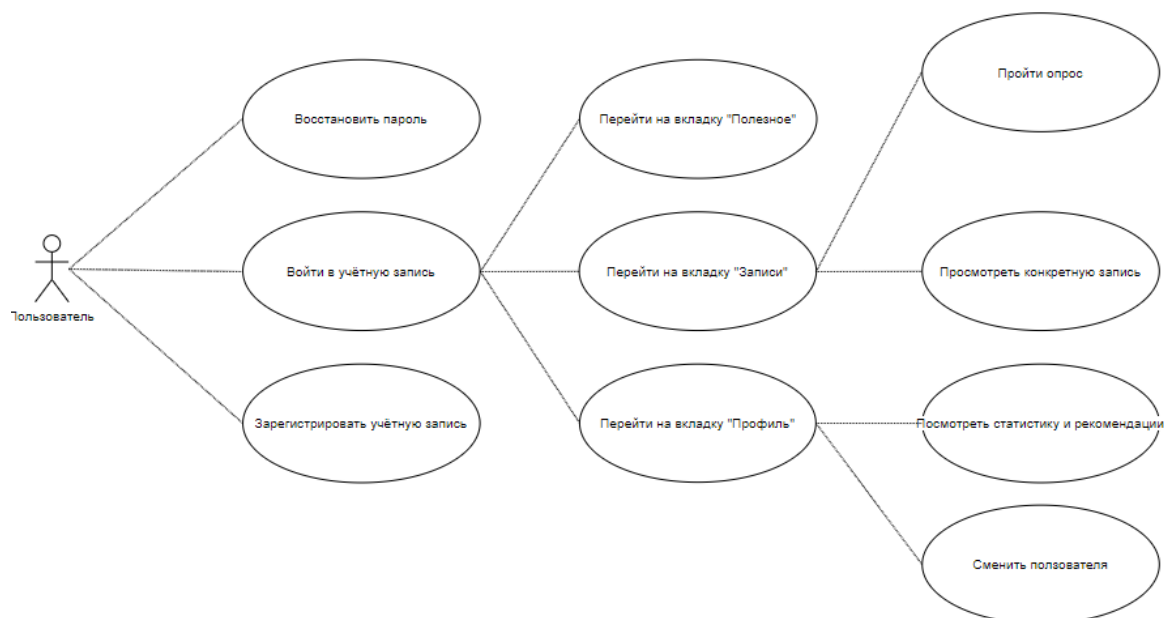


Рисунок 8 – *UML*-диаграмма вариантов использования

Для того, чтобы собирать данные о состоянии пользователя был сделан опрос, его алгоритм работы показан на рисунке 9.

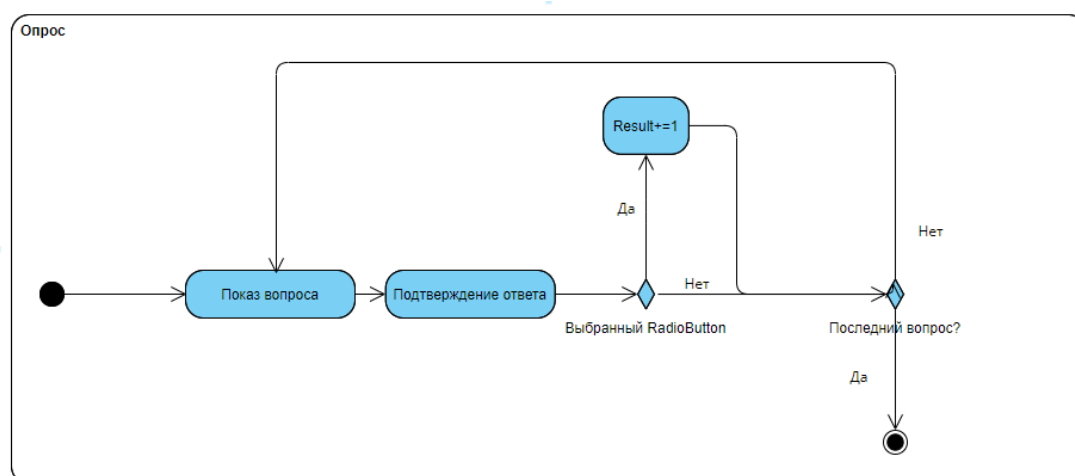


Рисунок 9 – *UML*-диаграмма деятельности опроса

После запуска опроса вызывается функция для показа следующего вопроса, чтобы изменить изначально пустой *TextView*, затем ожидается подтверждение ответа от пользователя, в зависимости от выбранного *RadioButton* в строку *result* заносится единица, что свидетельствует о наличие определённого симптома, затем происходит проверка на наличие других вопросов. Если вопросы ещё не закончились, показывается следующий вопрос, в противном случае опрос завершается, что и было представлено на рисунке 9.

1.3 Реализация приложения

1.3.3 Реализация аутентификации пользователей

Была создана система аутентификации пользователей на основе *Firebase*.

Аутентификация *Firebase* - это функция аутентификации пользователей, предоставляемая *Firebase* в качестве ее серверных служб. Она создана на основе токенов, это обеспечивает легкую интеграцию с большинством платформ. Еще одна замечательная особенность здесь заключается в том, что эта аутентификация легко интегрируется с API от *Firebase*, что делает возможным запустить эту функцию очень быстро и легко.

Аутентификация использует метод *signInWithEmailAndPassword()*, который принимает на вход 2 строки, которые берутся из полей *EditText* для электронной почты и пароля соответственно. Для того, чтобы не передать пустые строки в этот метод была сделана проверка на их заполнение.

Созданный интерфейс для аутентификации пользователей приведён на рисунке 10.

					ПГУ 1.090301.33.001 ПЗ	Лист
						17
Изм.	Лист	№ докум.	Подпись	Дата		

Email address

Password

LOGIN

Forgot password?

Create account

Рисунок 10 – Страница аутентификации «SDiary»

Код функции для авторизации пользователей представлен ниже:

```
btn_login.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (email_login.getText().toString().isEmpty() ||
password_login.getText().toString().isEmpty()) {
            Toast.makeText(LoginActivity.this,
"Заполните поля", Toast.LENGTH_SHORT).show();
        } else {
            mAuth.signInWithEmailAndPassword(email_login.getText().toString(
), password_login.getText().toString())
                .addOnCompleteListener(new
OnCompleteListener<AuthResult>() {
                    @Override
                    public void onComplete(@NonNull
Task<AuthResult> task) {
                        if (task.isSuccessful()) {
                            Intent intent = new
Intent(LoginActivity.this, MainActivity.class);
```

```

startActivity(intent);
}else{

Toast.makeText(LoginActivity.this, "У вас есть ошибки",
Toast.LENGTH_SHORT).show();

}

}

});

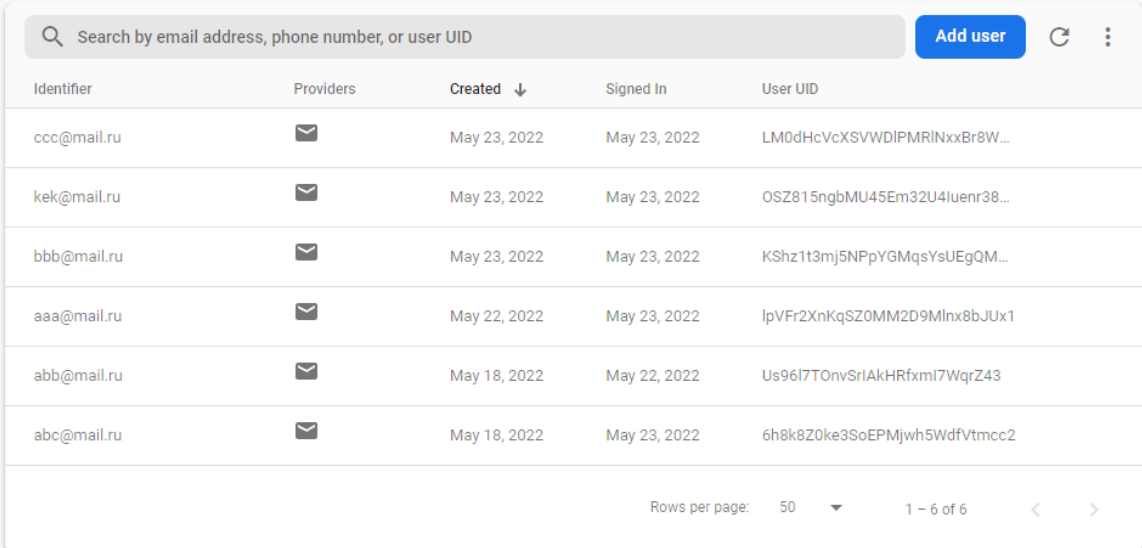
}

});

```

Для отслеживания нажатия на кнопку был использован метод *setOnClickListener*. Чтобы авторизовать пользователя через *FirebaseAuth* используется метод *signInWithEmailAndPassword*.

После авторизации производится переход на *MainActivity*, в которой содержится интерфейс дневника. Зарегистрированные учётные записи пользователей можно посмотреть в консоли *Firebase*, представленной на рисунке 11.



Identifier	Providers	Created ↓	Signed In	User UID
ccc@mail.ru	✉	May 23, 2022	May 23, 2022	LM0dHcVcXSVWDIPMRINxxBr8W...
kek@mail.ru	✉	May 23, 2022	May 23, 2022	OSZ815ngbMU45Em32U4Iuenr38...
bbb@mail.ru	✉	May 23, 2022	May 23, 2022	KShz1t3mj5NPpYGMqsYsUEgQM...
aaa@mail.ru	✉	May 22, 2022	May 23, 2022	IpVFr2XnKqSZ0MM2D9Mlnx8bJUx1
abb@mail.ru	✉	May 18, 2022	May 22, 2022	Us96l7TOnvSriAkHRfxmI7WqrZ43
abc@mail.ru	✉	May 18, 2022	May 23, 2022	6h8k8Z0ke3SoEPMjwh5WdfVtmcc2

Рисунок 11 – Список учётных записей

1.3.2 Реализация БД для хранения информации

База данных (БД) - это совокупность сведений о конкретных объектах реального мира в какой-либо предметной области. Для быстрого нахождения искомых данных необходимо структурировать информацию.

Созданная база данных содержит следующие сущности:

- *CheckHistory* – сущность, которая содержит даты его прохождения опросника для самодиагностики в атрибуте *Symptoms* в формате *String* и в атрибуте *ID*, содержащим *id* текущей записи в формате *Int*;
- *DateHistory* - сущность, которая содержит даты его прохождения опросника для самодиагностики в атрибуте *Date* в формате *String* и в атрибуте *ID*, содержащим *id* текущей записи в формате *Int*;
- *UID* – сущность, которая хранит основную информацию о пользователе. В атрибутах *Curent Check* и *LastCheck* хранится значение тяжести симптомов для прошлого и текущего опроса в формате *Int*. В атрибуте *email* содержится электронная почта пользователя в формате *String*. Атрибут *password* – содержит пароль пользователя в формате *String*. Атрибут *ID* используется для хранения идентификатор для следующей записи в формате *Int*.

Структура БД показана на рисунке 12.

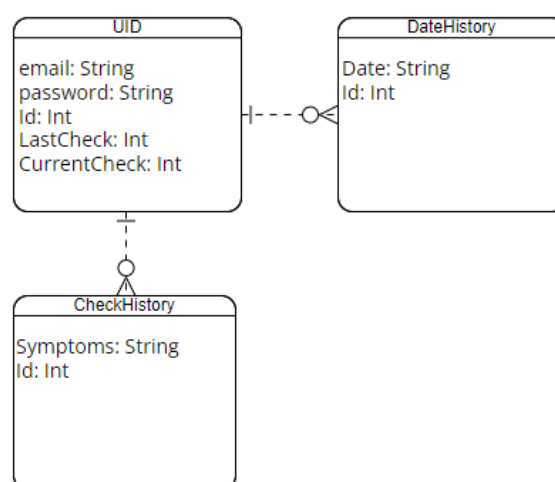


Рисунок 12 – ER-диаграмма получившейся БД

Результаты прохождения и даты прохождения опросника хранятся под идентификатором, начиная с нуля.

1.3.3 Реализация опросника для самодиагностики

Для удобства пользователя опросник было решено сделать в качестве викторины. *TextView*, содержащий вопросы, постоянно меняется благодаря вызову функции *showNextQuestion()*, код которой представлен ниже.

Код функции *showNextQuestion()*:

```
private void showNextQuestion() {
    if (questionCounter < questionCountTotal) {

textViewQuestion.setText(questionBank[questionCounter]
        .getAnswerResId());
        questionCounter++;
        textViewQuestionCount.setText("Вопрос: " +
questionCounter + "/" + questionCountTotal);
        buttonConfirmNext.setText("Ответить");
        if(questionCounter == questionCountTotal)
        {
            buttonConfirmNext.setText("Завершить
тестирование");
        }
        } else {

myRef.child("currentCheck").get().addOnCompleteListener(new
OnCompleteListener<DataSnapshot>() {
    @Override
    public void onComplete(@NonNull
Task<DataSnapshot> task) {
        if (!task.isSuccessful()) {

        }
        else {
```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.33.001 ПЗ

Лист

21

```

myRef.child("lastCheck").setValue(task.getResult().getValue());

        }

    }

});

myRef.child("currentCheck").setValue(severityResultParser(result));

myRef.child("CheckHistory").child(String.valueOf(historyId)).setValue(strResultParser(result));

myRef.child("DateHistory").child(String.valueOf(historyId)).setValue(String.valueOf(historyId) + ". " + currentDateTimeString);
        historyId++;
        myRef.child("ID").setValue(historyId);
        Intent intent = new Intent (SymptomsChecker.this,
MainActivity.class);
        startActivity(intent);
    }
}

```

Каждый раз при вызове функции сравнивается наличие других вопросов с помощью условия *if (questionCounter < questionCountTotal)*. При этом обновляется *TextView*:

```

textViewQuestion.setText(questionBank[questionCounter.getAnswerResId()]);

```

По завершении опроса результаты заносятся в БД с помощью функции *setValue*:

```

myRef.child("currentCheck").setValue(severityResultParser(result));

myRef.child("CheckHistory").child(String.valueOf(historyId)).setValue(strResultParser(result));

```

```

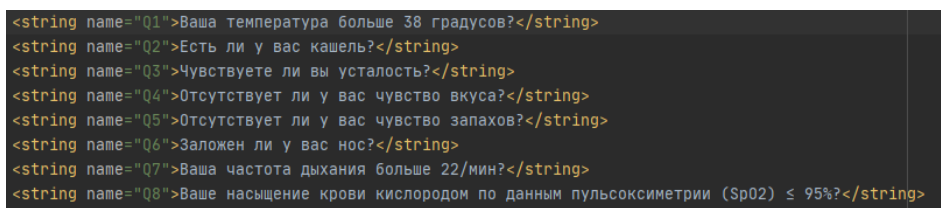
myRef.child("DateHistory").child(String.valueOf(historyId)).setValue(
String.valueOf(historyId)+". " + currentDateTimeString);

    historyId++;
    myRef.child("ID").setValue(historyId);
    Intent intent = new Intent (SymptomsChecker.this,
MainActivity.class);
    startActivity(intent);

```

И происходит возвращение на *MainActivity*.

Сами же вопросы хранятся в файле *strings.xml*, представленном на рисунке 13.



```

<string name="Q1">Ваша температура больше 38 градусов?</string>
<string name="Q2">Есть ли у вас кашель?</string>
<string name="Q3">Чувствуете ли вы усталость?</string>
<string name="Q4">Отсутствует ли у вас чувство вкуса?</string>
<string name="Q5">Отсутствует ли у вас чувство запахов?</string>
<string name="Q6">Заложен ли у вас нос?</string>
<string name="Q7">Ваша частота дыхания больше 22/мин?</string>
<string name="Q8">Ваше насыщение крови кислородом по данным пульсоксиметрии (SpO2) ≤ 95%?</string>

```

Рисунок 13 – Файл с вопросами

В результате формируется строка *result*, состоящая из нулей и единиц. В дальнейшем она парсится с помощью функции *strResultParser*, код которой представлен ниже.

Код функции *strResultParser ()*:

```

String parsedRes = null;

    for(int i = 0; i < result.length(); i++)
    {

        switch (i) {

            case 0:

                if(result.charAt(i) == '1')
                {

                    parsedRes = "Ваша температура выше
38 градусов\n";

                }

```

```

else

    parsedRes = "Ваша температура ниже
38\n";

    break;

case 1:

    if(result.charAt(i) == '1')

    {

        parsedRes += "Присутствует
кашель\n";

    }

    else

        parsedRes += "Отсутствует кашель\n";

    break;

case 2:

    if(result.charAt(i) == '1')

    {

        parsedRes += "Присутствует
усталось\n";

    }

    else

        parsedRes += "Отсутствует
усталость\n";

    break;
    break;

case 5:

    if(result.charAt(i) == '1')

    {


```

```

case 3:

    if(result.charAt(i) == '1')
    {
        parsedRes += "Чувство вкуса
отсутствует\n";
    }
    else
        parsedRes += "Чувство вкуса в
норме\n";

    break;
case 4:
    if(result.charAt(i) == '1')
    {
        parsedRes += "Обоняние
отсутствует\n";
    }
    else
        parsedRes += "Обоняние в норме\n";
        parsedRes += "Нос заложен\n";
    }
    else
        parsedRes += "Нос не заложен\n";
    break;
case 6:
    if(result.charAt(i) == '1')
    {
        parsedRes += "Частота дыхания >
22/мин\n";
    }
    else
        parsedRes += "Частота дыхания <
}22/мин\n";

    break;
case 7:
    if(result.charAt(i) == '1')
    {

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.33.001 ПЗ

Лист

25

```

        parsedRes += "Насыщение крови
кислородом <95%\n";
    }
    else
        parsedRes += "Насыщение крови
кислородом в норме\n";
    break;
}
return parsedRes;
};

```

После завершения опроса в БД вносятся результаты текущего опроса, а так же тяжесть состояния пациента. В зависимости от наличия определённых симптомов параметр *severity* растёт.

Тяжесть состояния пользователя определяется по этому показателю:

- (1-6) - лёгкая форма;
- (7-13) - среднетяжёлая;
- 19+ тяжёлая.

1.3.4 Интерфейс приложения

Создание интерфейса для проекта - то, на чём стоит заострить внимание. Дизайн должен не должен быть слишком сложным, при этом предоставлять всю необходимую информацию пользователю и быть удобным в использовании. Для выполнения этих целей было решено использовать фрагменты и *BottomNavigationView*, который представлен на рисунке 14.



Рисунок 14 – *BottomNavigationView*

Данный компонент интерфейса используется в качестве замены верхнего меню приложения и уже является чуть ли не обязательным элементом современных мобильных приложений.

Fragment — модульная часть *activity*, у которой свой жизненный цикл и свои обработчики различных событий. *Android* добавил фрагменты с *API 11*, для того, чтобы разработчики могли разрабатывать более гибкие пользовательские интерфейсы на больших экранах, таких как экраны планшетов. Через некоторое время была написана библиотека, которая добавляет поддержку фрагментов в более старые версии.

Интерфейсы всех трёх фрагментов представлены на рисунках 15-18 ниже.

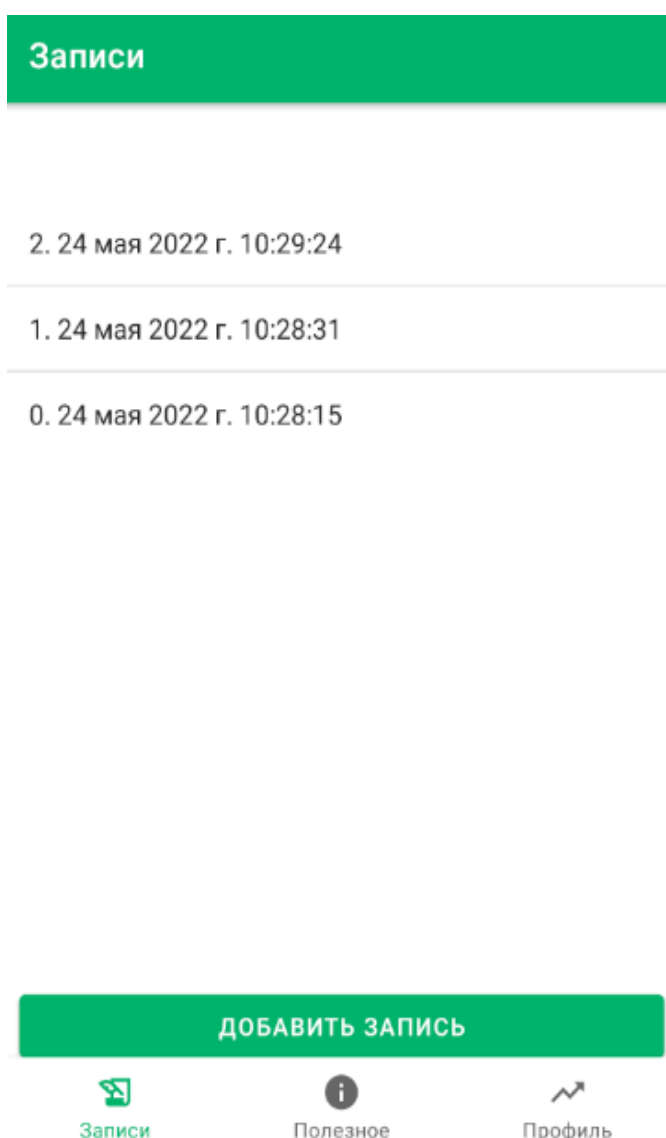


Рисунок 15 – Фрагмент «Записи»

Полезное

Всего заболевших: 18327837
Всего смертей: 379029
Больных на данный момент: 214938
Выздоровело: 17733870

1. При каких симптомах к пациенту на дом обязан прийти врач из районной поликлиники?

Врача на дом можно вызвать, если есть повышение температуры – выше 37,2 градуса и/или признаки ОРВИ: кашель, заложенность носа, затрудненное дыхание, одышка, боли в груди, слабость, головная боль, потеря обоняния.

Кроме того, есть условия, когда вызов врача на дом при таких симптомах просто желателен, а строго необходим. Это:

возраст пациента старше 65 лет;
сопутствующие патологии, то есть хронические болезни, при которых повышается риск тяжелых осложнений COVID-19. В первую очередь это сахарный диабет, сердечно-сосудистые заболевания, хронические болезни почек, органов дыхания, онкологические заболевания; контакт с лицами, у которых диагностирован COVID-19, либо приезд из региона с напряженной эпидситуацией по распространению коронавирусной инфекции.
! Обязательно сообщите об этих фактах, вызывая врача.



Записи



Полезное



Профиль

Рисунок 16 – Фрагмент «Полезное»

Профиль

Судя по вашим записям, вы болеете тяжелой формой. Советуем вам вызвать врача. Ваше состояние ухудшилось.

Единый консультационный центр Роспотребнадзора:
8 800 555-49-43.
Единый номер для вызова врача или записи на прививку: 122.

СМЕНИТЬ ПОЛЬЗОВАТЕЛЯ



Записи



Полезное



Профиль

Рисунок 17 – Фрагмент «Профиль»

При нажатии на один из пунктов вызывается соответствующий фрагмент, который помещается в контейнер выше.

Изм.	Лист	№ докум.	Подпись	Дата

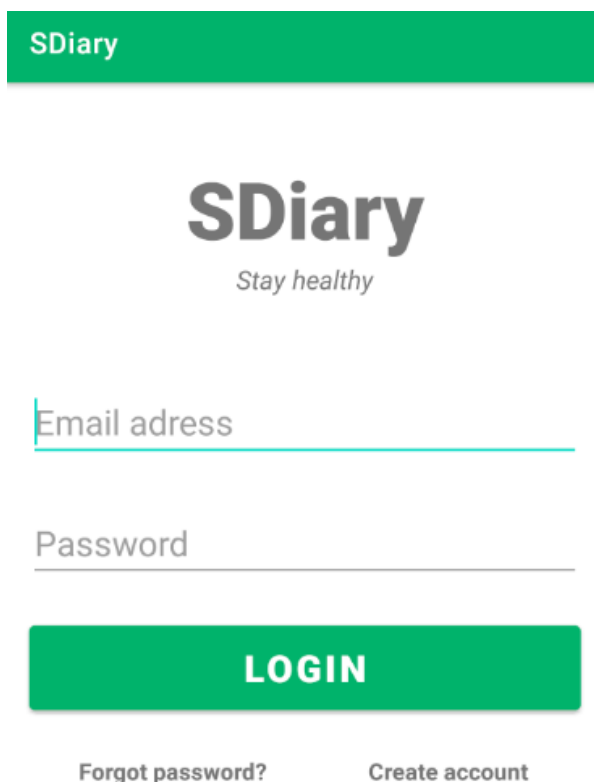
ПГУ 1.090301.33.001 ПЗ

Лист

28

1.4 Руководство пользователя

При запуске приложения пользователю нужно либо зайти в существующий аккаунт, либо зарегистрировать новый, страница аутентификации представлена на рисунке 18.



SDiary

SDiary
Stay healthy

Email adress

Password

LOGIN

Forgot password? Create account

Рисунок 18 – Страница аутентификации

После успешной аутентификации пользователь переходит на вкладку «Записи» приложения. (рисунок 19) Вкладка «Записи» используется для отображения истории результатов самодиагностики и добавления новых записей. При нажатии на кнопку «Добавить запись», пользователь запустит опросник. Интерфейс опросника показан на рисунках 19-23.

Изм.	Лист	№ докум.	Подпись	Дата

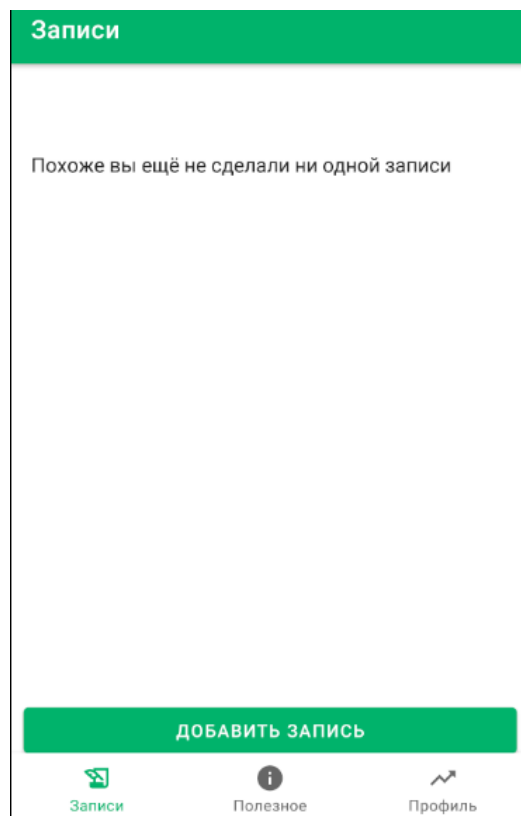


Рисунок 19 – Вкладка «Записи»

Рисунок 20 – Процесс добавления записи

SDiary

Вопрос: 8/8

Ваше насыщение крови кислородом по данным пульсоксиметрии (SpO2) $\leq$ 95%?

☐ Да

☒ Нет

ЗАВЕРШИТЬ ТЕСТИРОВАНИЕ

Рисунок 21. Завершение добавления записи

Записи

0. May 23, 2022 2:58:21 PM

ДОБАВИТЬ ЗАПИСЬ

Записи

Полезное

Профиль

Рисунок 22. Результат добавления записи

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.33.001 ПЗ

Лист

31

SDiary

Вопрос: 8/8

Ваше насыщение крови кислородом по данным пульсоксиметрии (SpO2) $\leq$ 95%?

☐ Да

☒ Нет

ЗАВЕРШИТЬ ТЕСТИРОВАНИЕ

Рисунок 23. Результат добавления записи

После пользователь может получить полезную информацию касательно коронавируса, открыв вкладку «полезное», которая представлена на рисунке 24.

Полезное

Всего заболевших: 18297608
 Всего смертей: 378426
 Больных на данный момент: 225162
 Выздоровело: 17694020

1. При каких симптомах к пациенту на дом обязан прийти врач из районной поликлиники?

Врача на дом можно вызвать, если есть повышение температуры — выше 37,2 градуса и/или признаки ОРВИ: кашель, заложенность носа, затрудненное дыхание, одышка, боли в груди, слабость, головная боль, потеря обоняния.

Кроме того, есть условия, когда вызов врача на дом при таких симптомах просто желателен, а строго необходим. Это:

возраст пациента старше 65 лет;
 сопутствующие патологии, то есть хронические болезни, при которых повышается риск тяжелых осложнений COVID-19. В первую очередь это сахарный диабет, сердечно-сосудистые заболевания, хронические болезни почек, органов дыхания, онкологические заболевания;
 контакт с лицами, у которых диагностирован COVID-19, либо приезд из региона с напряженной эпидситуацией по распространению коронавирусной инфекции.
 ! Обязательно сообщите об этих фактах, вызывая врача.

Записи
 Полезное
 Профиль

Рисунок 24 – Вкладка «Полезное»

Изм.	Лист	№ докум.	Подпись	Дата

Вкладка «Профиль» содержит результаты отслеживания динамики заболевания, полученные на основе записей пользователя и кнопку для смены пользователя. Интерфейс данной вкладки представлен на рисунке 25.

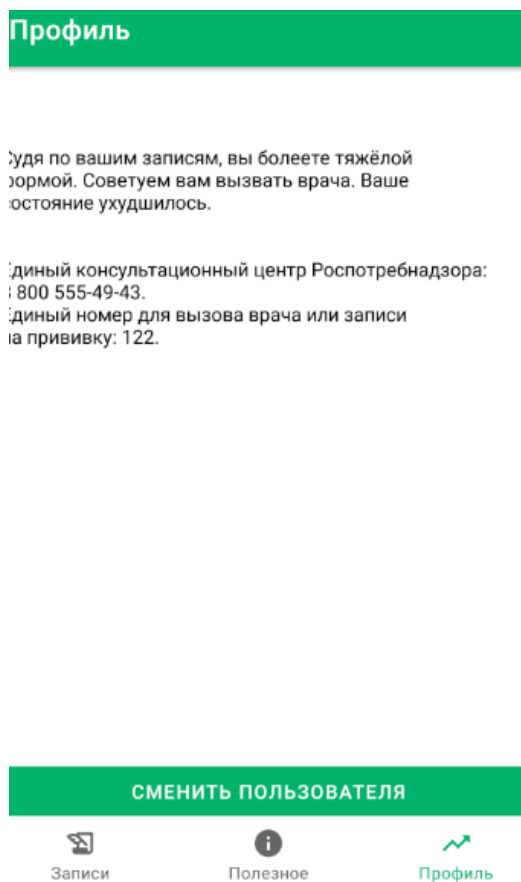


Рисунок 25 – Вкладка «Профиль»

При желании, пользователь может сменить пользователя, нажав на соответствующую кнопку.

2 Безопасность жизнедеятельности

О безопасности жизнедеятельности человека, несомненно, необходимо беспокоиться на всех стадиях жизненного цикла проекта, будь то разработка, внедрение в жизнь или эксплуатация программы.

Создание безопасных и комфортных условий работы в большей степени зависит от правильной оценки опасных, вредных производственных факторов. Одинаковые по тяжести изменения в организме человека могут быть вызваны различными причинами. Это могут быть различные факторы производственной среды, чрезмерная физическая и умственная нагрузка, нервно-эмоциональное напряжение, а также разное сочетание этих причин.

2.1 Потенциально опасные факторы при работе с ПЭВМ

Опасные и вредные производственные факторы по природе возникновения делятся на следующие группы:

- физические;
- химические;
- психофизиологические;
- биологические.

При работе с ПЭВМ на человека могут негативно действовать следующие физические факторы:

- повышенная и пониженная температура воздуха;
- чрезмерная запыленность и загазованность воздуха;
- повышенная и пониженная влажность воздуха;
- превышающий допустимые нормы шум;
- повышенный уровень ионизирующего излучения;
- опасность поражения электрическим током;
- повышенный уровень статического электричества;
- блеклость экрана дисплея;
- недостаточная освещенность рабочего места.

Изм.	Лист	№ докум.	Подпись	Дата

К химически опасным факторам, постоянно действующим на программиста, относятся активные частицы, возникающие в результате ионизации воздуха при работе компьютера.

Биологические вредные производственные факторы в данном помещении отсутствуют.

К психологически вредным факторам, воздействующим на программиста в течение его рабочей смены, можно отнести следующие:

- нервно - эмоциональные перегрузки;
- умственное напряжение;
- перенапряжение зрительного анализатора.

2.2 Классификация помещения

Кабинет программиста является помещением без повышенной опасности категории В1-В4 (пожароопасность) I категории (выполняются легкие физические работы), поэтому должны соблюдаться следующие требования:

- температура 22 °С (допустимая – 20-24 °С);
- оптимальная относительная влажность – 40-60% (допустимая – не более 75%);
- скорость движения воздуха не более 0.1м/с.

Для создания и автоматического поддержания на рабочем месте оптимальных значений температуры, влажности, чистоты и скорости движения воздуха, независимо от наружных условий, в холодное время года используется водяное отопление, в теплое время года применяется кондиционирование воздуха. Кондиционер представляет собой вентиляционную установку, которая с помощью приборов автоматического регулирования поддерживает в помещении заданные параметры воздушной среды.

Руководитель (владелец) предприятия должен обеспечить:

- содержание электрооборудования в работоспособном состоянии и его эксплуатацию в соответствии с требованиями нормативной технической документации;
- своевременное и качественное проведение профилактических работ, ремонта, модернизации и реконструкции энергетического оборудования;
- надежность работы электроустановок и безопасность их обслуживания;
- предотвращение использования технологий и методов работы, оказывающих отрицательное воздействие на окружающую среду;
- учет и анализ нарушений в работе электроустановок, несчастных случаев и принятие мер по устранению причин их возникновения;
- разработку должностных инструкций и производственных инструкций для электротехнического персонала;
- выполнение предписаний органов Госэнергонадзора.

2.3 Освещение

Работа с ПК зачастую происходит в помещениях с искусственным освещением, которое должно обеспечивать правильную работу глаз и приближать к оптимальным условиям зрительное восприятие, какое бывает при естественном солнечном освещении.

Недостаточность освещения приводит к напряжению зрения, ослабляет внимание, приводит к наступлению преждевременной утомленности. Чрезмерно яркое освещение вызывает ослепление, раздражение и резь в глазах. Неправильное направление света на рабочем месте может создавать резкие тени, блики, дезориентировать работающего. Все эти причины могут привести к несчастному случаю или профзаболеваниям, поэтому столь важно уделять внимание освещению.

Человек имеет как центральное (колбочковое), так и периферическое (палочковое) зрение. Первое -- для восприятия цветов и объектов малых размеров, второе -- для восприятия окружающего фона и крупных объектов. Центральное зрение требует большей яркости, а палочковое действует в сумерках или полумраке. Учитывая, что при работе с дисплеями задействовано именно центральное зрение, становится понятной необходимость достаточного освещения помещения, где находится компьютер. Самые общие правила организации освещения заключаются в следующем:

- следует избегать большого контраста между яркостью экрана и окружающего пространства. Оптимальным считается их выравнивание;
- запрещается работа с компьютером в темном или полутемном помещении;
- освещение в помещениях с ПК должно быть смешанным: естественным, -- за счет солнечного света, -- и искусственным.

Хорошо, если окна, обеспечивающие естественное освещение, имеют северную ориентацию. Если нет, необходимо принять меры, благодаря которым интенсивный солнечный свет из южных или западных окон не мешал бы работе. Так например, оконные проемы можно оборудовать жалюзи, занавесями, внешними козырьками. В качестве источников общего искусственного освещения лучше всего использовать осветительные приборы, которые создают равномерную освещенность путем рассеянного или отраженного светораспределения (свет от ламп падает непосредственно на потолок) и исключают блики на экране монитора и клавиатуре. В соответствии с санитарными нормами, это должны быть преимущественно люминесцентные лампы типа ЛБ с рассеивателями или экранирующими решетками. Пульсации света люминесцентных ламп действуют раздражающе на зрение и нервную систему операторов, поэтому для уменьшения коэффициента пульсации используйте лампы, укомплектованные высокочастотными пускорегулирующими аппаратами.

Следует отметить, что существуют специальные люминесцентные лампы, которые излучают свет различного «качества», имитируя таким образом, полный спектр естественного солнечного света. Они меньше раздражают, чем любые другие лампы искусственного освещения.

Источники света необходимо равномерно распределять по комнате, komponуя в сплошные или прерывистые линии. Линии должны располагаться сбоку от рабочих мест параллельно линии зрения пользователя -- при рядом размещении компьютеров; локализовано над рабочим столом -- при размещении рабочих мест по периметру помещения.

Грамотная организация освещения способна повысить производительность труда при зрительной работе средней трудности -- на 5--6%, при очень трудной -- на 15%.

Если деятельность пользователя является комбинированной, т. е. предполагает работу как с компьютером, так и с документами, на рабочие места необходимо устанавливать источники местного освещения -- настольные лампы с регулируемым наклоном шифона и регулируемой яркостью. В этом случае надо следить, чтобы свет от лампы не действовал раздражающе и не создавал бликов на экране.

Значительного эффекта в ослаблении вредных воздействий можно добиться применением для дисплеев защитных фильтров. Для примера можно привести данные из рекламного проспекта защитных экранов шотландской фирмы «Ocli», прошедших европейские стандартные испытания *MIL STD 285*:

- небьющееся стекло;
- абсолютное устранение бликов и мерцания экрана;
- подавление электростатического поля на 99%;
- подавление электромагнитного поля на 98%;
- уменьшение рассеянного отражения на 99.5%.

Изм.	Лист	№ докум.	Подпись	Дата

2.4 Расчет освещенности рабочего места

Правильный выбор и расчет освещенности рабочего места обеспечивает создание нормальных условий для зрения обслуживающего персонала, способствует повышению производительности труда. Освещение, соответствующее санитарным нормам, является главнейшим условием гигиены труда и культуры производства. При хорошем освещении устраняется напряжение зрения, ускоряется темп работы.

Для рационального освещения необходимо выполнение следующих условий:

- постоянная освещенность рабочих поверхностей;
- достаточная и равномерно распределенная яркость освещаемых рабочих поверхностей;
- отсутствие резких контрастов между яркостью рабочей поверхностью и окружающего пространства.

Для рационального освещения производственных помещений и рабочих мест большое значение имеет выбор цвета для окраски потолка, стен и производственного оборудования.

Рассчитаем люминесцентное освещение кабинета, предназначенного для выполнения работ с размером объекта различения от 0,3 мин до 1 мин. Размеры помещений: А=4 м, В=3 м, Н=3 м. Площадь помещения определяем по формуле:

$$S=A*B.$$

Подставив значения получим:

$$S=4*3=12 \text{ м}^2.$$

Освещение проектируется при помощи светильников ШОД минимальной освещенностью $E_{\min}=200$ лк, средней удельной мощностью 17-23 Вт/м². Высота подвеса над рабочей поверхностью $H_p=2$ м.

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.33.001 ПЗ

Лист

39

Освещение выполняется лампами ЛБ-2*40, световой поток ламп $F_{\text{л}}=2480$ лм, длина 1.2 м, коэффициент запаса равен $k=1.5$. Определим показатель помещения по формуле:

$$F = A * B / (h_p (A + B)).$$

Подставив значения получим:

$$F = 4 * 3 / 2 * (4 + 3) = 0.86.$$

Затем, для $F=0.86$, коэффициентов отражения потолка $R_{\text{п}}=0.7$, стен $R_{\text{с}}=0.5$ и расчетной плоскости $R_{\text{р}}=0.3$ находим коэффициент использования светового потока $h=0.37$. Потребное число светильников определяется по формуле:

$$N = E_{\text{мин}} * S * K / F_{\text{л}} * h * Z * n.$$

Где $n=2$ общее число ламп в светильнике.

Подставив значения получим:

$$N = 200 * 12 * 1.5 / 2480 * 0.37 * 0.9 * 2 = 2 \text{ штуки.}$$

Общее количество ламп равно $n=2*2=4$.

Таким образом, выполнение перечисленных выше требований к организации рабочего места обеспечит комфортные условия умственного труда.

Заключение

В результате работы было создано мобильное приложение с удобным и понятным пользовательским интерфейсом. Созданный проект позволяет работать с базой данных симптомов, отслеживать динамику их развития, а так же получать различную полезную информацию с помощью интернета.

Разработка данного приложения позволила улучшить навыки создания клиентских Android приложений, навыки работы с базами данных, API и интерфейсами

Навыки, приобретённые в ходе разработки этого приложения, определённо пригодятся в дальнейшем трудоустройстве и развитии собственных навыков.

					ПГУ 1.090301.33.001 ПЗ	Лист
						41
Изм.	Лист	№ докум.	Подпись	Дата		

Список используемых источников

1. Дейтел П., Дейтел Х., Уолд А. Android для разработчика. 3-е изд. – СПб.: Питер, 2017. – С. 37-374.
2. Мартин Р. Чистый Код: создание, анализ и рефакторинг. Пер. с англ.– М.: Издательский дом “Питер”, 2018. — С. 50-130
3. Гамма Э., Хелм., Джонсон Р., Влиссидес.. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб.: Питер, 2018. . – С. 25-60.
4. Пушников А. Ю. Введение в системы управления базами данных. URL:<http://www.citforum.ru/database/dblearn/index.shtml>
5. *Firebase – RealtimeDatabase*[Электронный ресурс] URL: <https://firebase.google.com/docs/database>
6. *Medium – Android Developers* [Электронный ресурс] URL: <https://medium.com/androiddevelopers>
7. *Retrofit*[Электронный ресурс] URL:<https://square.github.io/retrofit/>

Приложение А

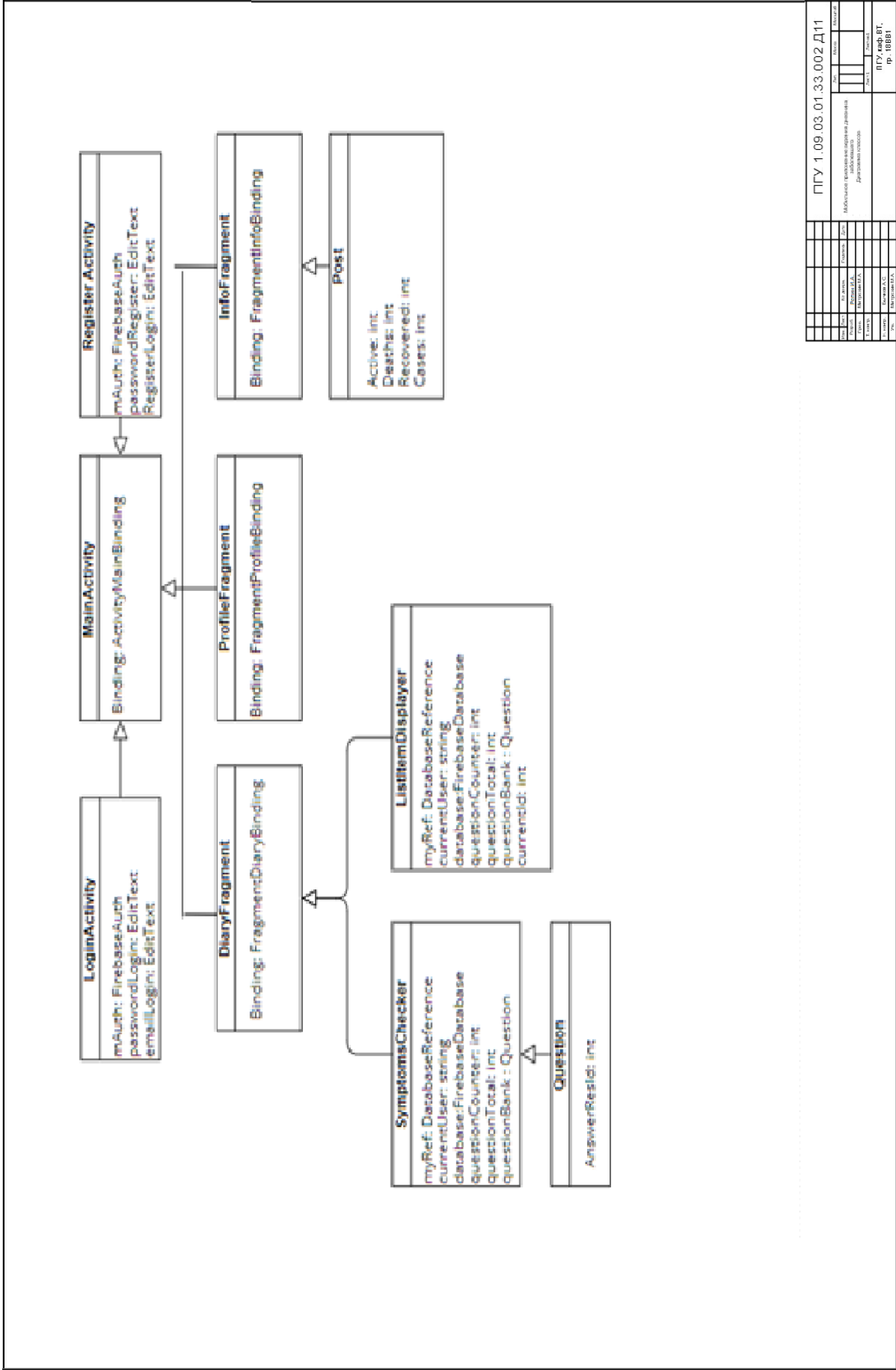
(Обязательное)

Мобильное приложение ведения дневника заболевшего

UML-диаграммы

					ПГУ 1.090301.33.001 ПЗ	Лист
						43
Изм.	Лист	№ докум.	Подпись	Дата		

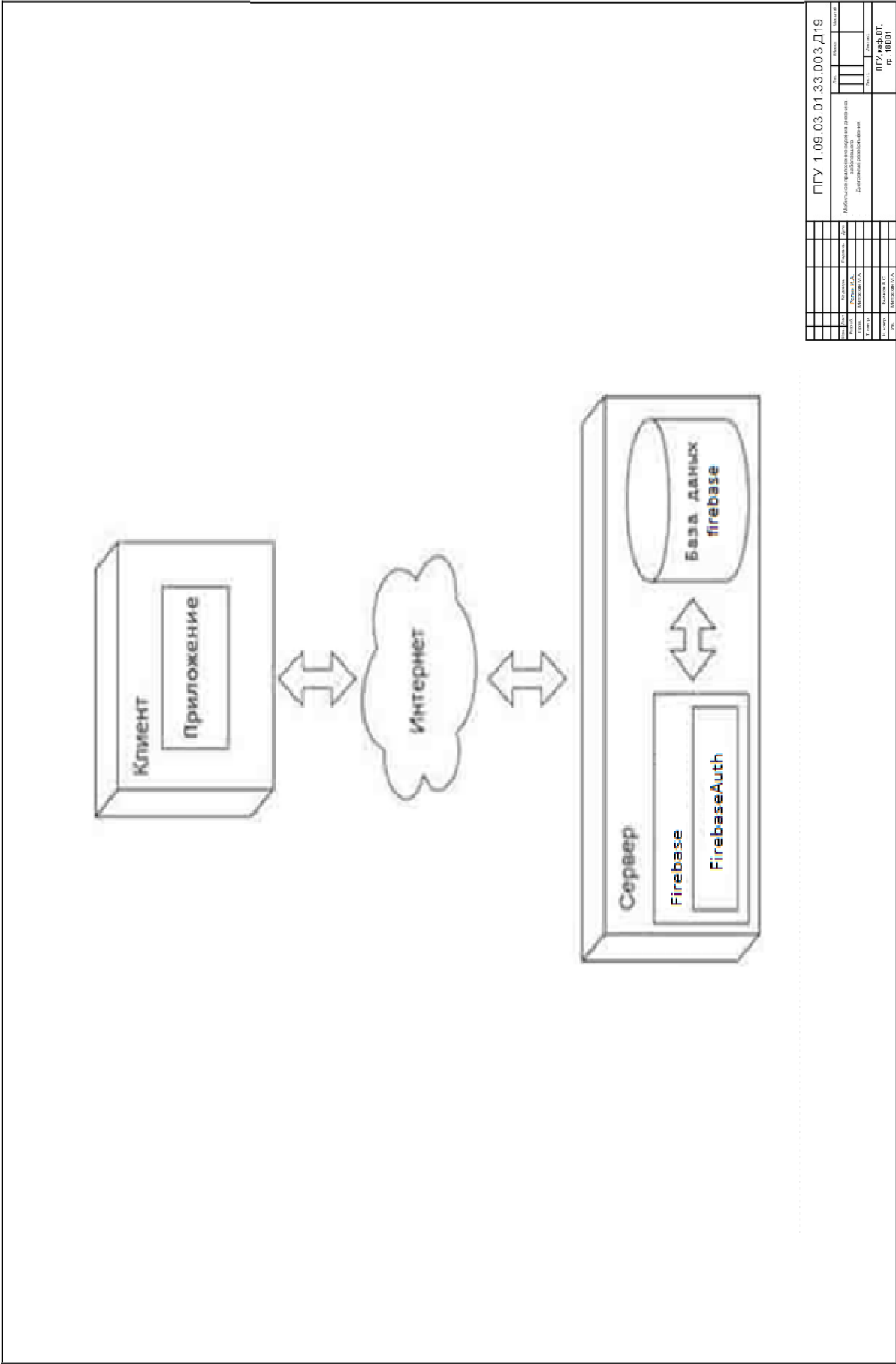
Приложение А.1 - Диаграмма классов



ПГУ 1.09.03.01.33.002 А11									
Изм.	Лист	№ докум.	Подпись	Дата	Исполн.	Провер.	Соглас.	Соглас.	Соглас.
					Модуль: приложение «Симптомы»				
					Исполнитель: [подпись]				
					ПГУ, кафед. ИТ, п. 10881				

Изм.	Лист	№ докум.	Подпись	Дата

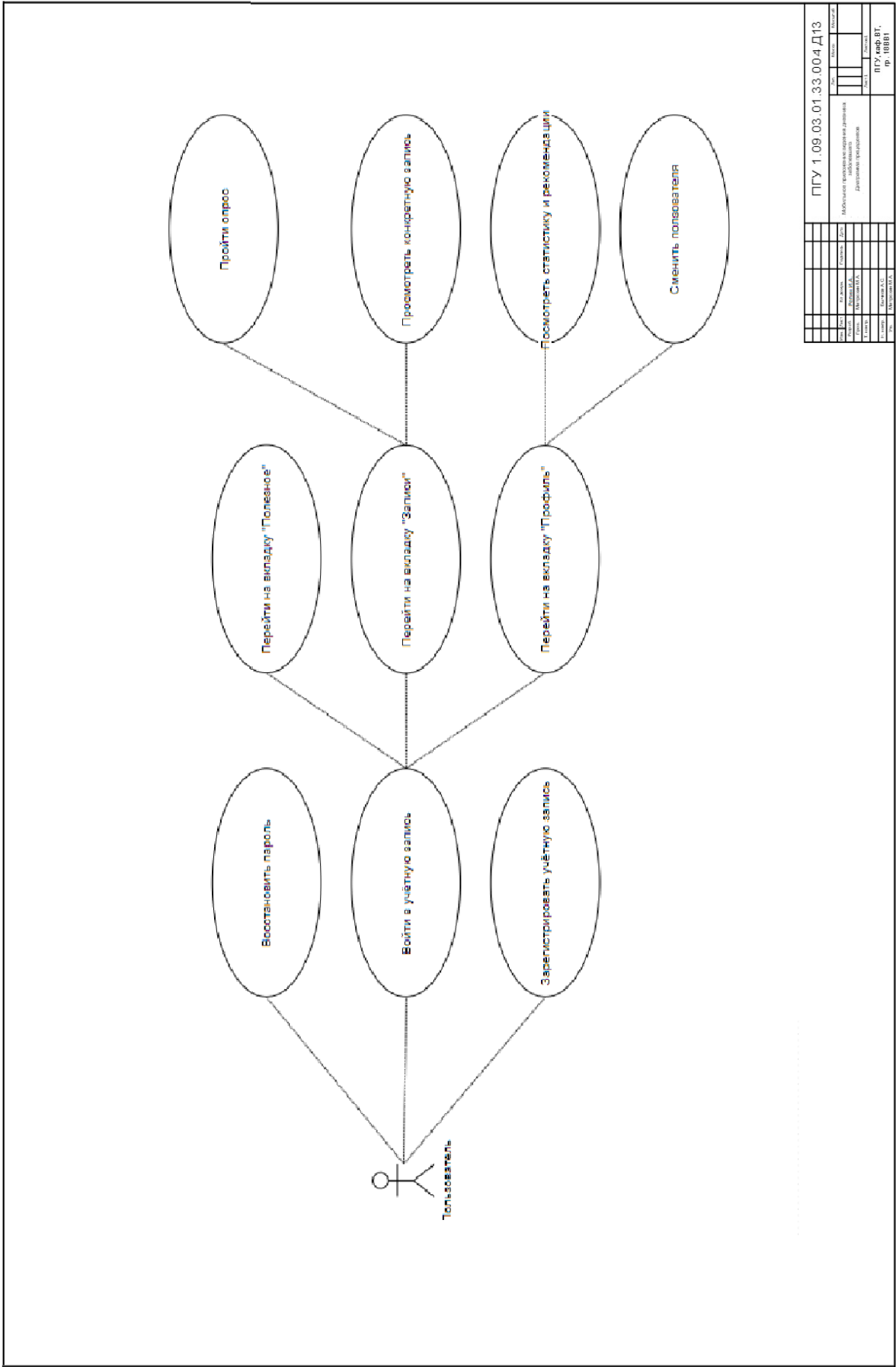
Приложение А.2 - Диаграмма развёртывания



ПГУ 1.09.03.01.33.003 Д19									
Исполнитель: ПГУ, кафе ВТ, р. 188В1									
Должность: Программист									
Имя: _____									
Фамилия: _____									
Подпись: _____									
Дата: _____									

Изм.	Лист	№ докум.	Подпись	Дата

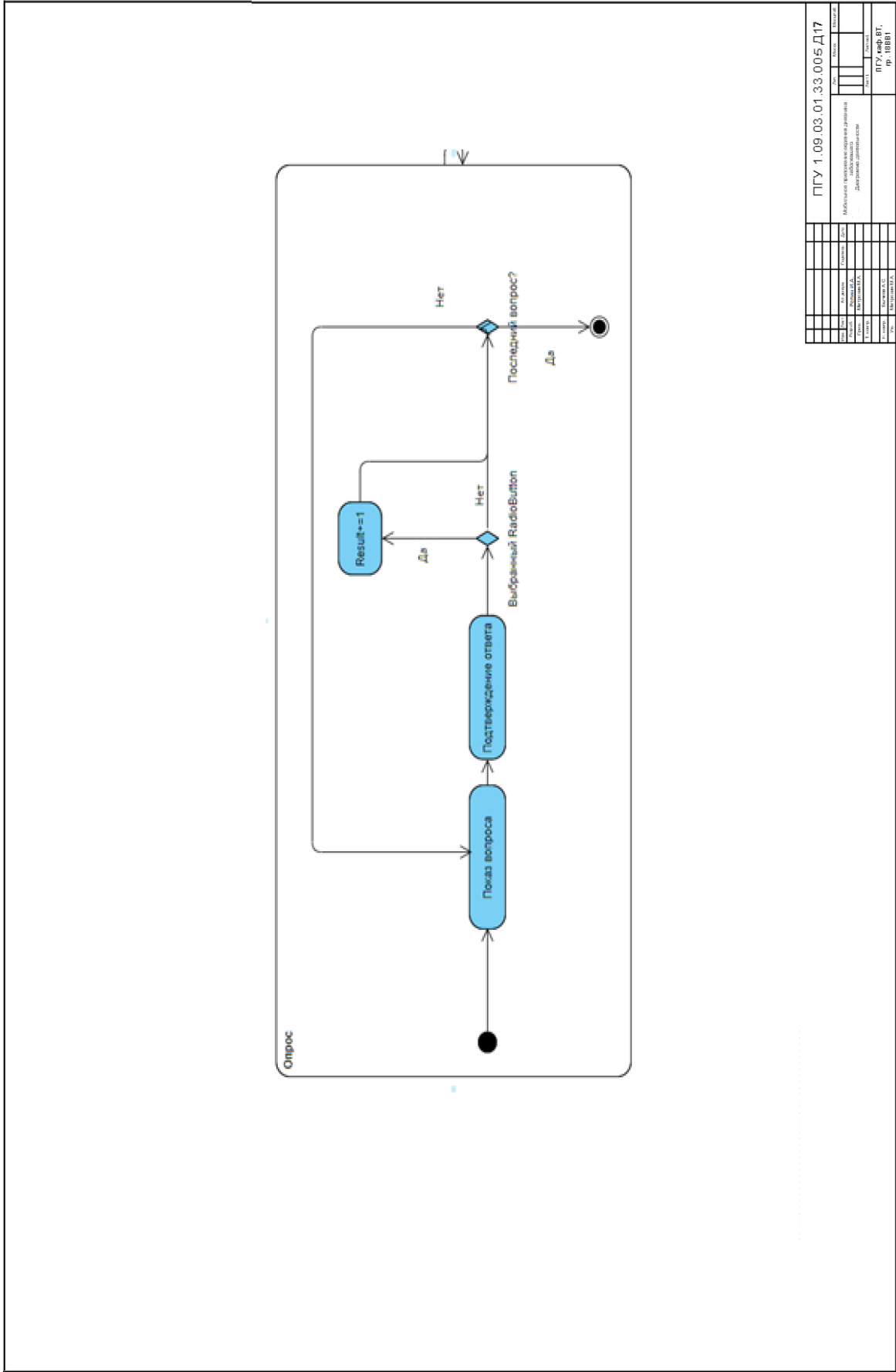
Приложение А.3 - Диаграмма прецедентов



ПГУ 1.09.03.01.33.00.4 Д13									
Исполнитель: _____									
Должность: _____									
ПГУ, мед. ВТ, р. 108801									

Изм.	Лист	№ докум.	Подпись	Дата

Приложение А.4 - Диаграмма деятельности



ПГУ 1.09.03.01.33.005 Д17				
Изм.	Лист	№ докум.	Подпись	Дата
1	1	1	1	1
Матрица контроля качества				
Диагностика деятельности				
ПГУ, каф. ВТ, пр. 10001				

Приложение Б

(Обязательное)

Мобильное приложение ведения дневника заболевшего

Листинг программы

					ПГУ 1.090301.33.001 ПЗ	Лист
						48
Изм.	Лист	№ докум.	Подпись	Дата		

Файл MainActivity.java

```
package com.example.diploma2;

import android.app.Fragment;
import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;

import
com.google.android.material.bottomnavigation.BottomNavigationVie
w;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.navigation.NavController;
import androidx.navigation.Navigation;
import androidx.navigation.ui.AppBarConfiguration;
import androidx.navigation.ui.NavigationUI;

import com.example.diploma2.databinding.ActivityMainBinding;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebase.database.DatabaseReference;
import com.google.gson.annotations.Expose;
import com.google.gson.annotations.SerializedName;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;
import retrofit2.http.GET;
import retrofit2.http.Path;

public class MainActivity extends AppCompatActivity {

    private ActivityMainBinding binding;

    public static class NetworkService {
        private static NetworkService mInstance;
        private static final String BASE_URL =
"https://disease.sh";
        private Retrofit mRetrofit;

        public NetworkService() {
            mRetrofit = new Retrofit.Builder()
                .baseUrl(BASE_URL)

.addConverterFactory(GsonConverterFactory.create())
                .build();
        }
    }
}
```

```

    public static NetworkService getInstance() {
        if (mInstance == null) {
            mInstance = new NetworkService();
        }
        return mInstance;
    }

    public JSONPlaceholderApi getJSONApi() {
        return mRetrofit.create(JSONPlaceholderApi.class);
    }
}

public class Post {
    @SerializedName("cases")
    @Expose
    private int cases;
    @SerializedName("deaths")
    @Expose
    private int deaths;
    @SerializedName("recovered")
    @Expose
    private int recovered;
    @SerializedName("active")
    @Expose
    private int active;

    public int getCases() {
        return cases;
    }

    public void setCases(int cases) {
        this.cases = cases;
    }

    public int getDeaths() {
        return deaths;
    }

    public void setDeaths(int deaths) {
        this.deaths = deaths;
    }

    public int getRecovered() {
        return recovered;
    }

    public void setRecovered(int recovered) {
        this.recovered = recovered;
    }
}

```

```

        public int getActive() {
            return active;
        }

        public void setActive(int active) {
            this.active = active;
        }
    }

    public interface JSONPlaceholderApi {
        @GET("/v3/covid-19/countries/{country}")
        public Call<Post> getPostWithID(@Path("country") String
country);
    }

    public void onStart() {
        super.onStart();
        // Check if user is signed in (non-null) and update UI
accordingly.
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        binding
ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());

        BottomNavigationView navView
=
findViewById(R.id.nav_view);
        // Passing each menu ID as a set of Ids because
each
        // menu should be considered as top level
destinations.
        AppBarConfiguration appBarConfiguration = new
AppBarConfiguration.Builder(
            R.id.navigation_home,
R.id.navigation_dashboard, R.id.navigation_notifications)
            .build();
        NavController navController
=
Navigation.findNavController(this,
R.id.nav_host_fragment_activity_main);

        NavigationUI.setupActionBarWithNavController(this,
navController, appBarConfiguration);

        NavigationUI.setupWithNavController(binding.navView,
navController);
    }

```

```

        //Fragment stats =
getFragmentManager().findFragmentById(R.id.text_dashboard);
        //((TextView)
stats.getView().findViewById(R.id.text_dashboard)).setText("abs
");
    }

}

```

Файл LoginActivity.java

```

package com.example.diploma2;

import android.content.Intent;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

import android.widget.EditText;

import android.widget.TextView;

import android.widget.Toast;

import androidx.annotation.NonNull;

import androidx.appcompat.app.AppCompatActivity;

import com.google.android.gms.tasks.OnCompleteListener;

import com.google.android.gms.tasks.Task;

import com.google.firebase.auth.AuthResult;

import com.google.firebase.auth.FirebaseAuth;

public class LoginActivity extends AppCompatActivity {

    private EditText email_login;

    private EditText password_login;

    private Button btn_login;

    private TextView register txt;

```

					ПГУ 1.090301.33.001 ПЗ	Лист
						52
Изм.	Лист	№ докум.	Подпись	Дата		

```

private FirebaseAuth mAuth;

@Override

protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_login);

    email_login = findViewById(R.id.email_login);
    password_login = findViewById(R.id.password_login);
    btn_login = findViewById(R.id.signIn);
    register_txt = findViewById(R.id.register);

    mAuth = FirebaseAuth.getInstance();

    register_txt.setOnClickListener(new
View.OnClickListener() {

        @Override

        public void onClick(View v) {

            Intent intent = new Intent (LoginActivity.this,
RegisterActivity.class);

            startActivity(intent);

        }

    });

    btn_login.setOnClickListener(new View.OnClickListener()
{

        @Override

        public void onClick(View v) {

            if

```

```

        (email_login.getText().toString().isEmpty() ||
password_login.getText().toString().isEmpty()) {

            Toast.makeText(LoginActivity.this,
"Заполните поля", Toast.LENGTH_SHORT).show();

        }else{

mAuth.signInWithEmailAndPassword(email_login.getText().toString(),
password_login.getText().toString())

                .addOnCompleteListener(new
OnCompleteListener<AuthResult>() {

                    @Override

                    public void onComplete(@NonNull
Task<AuthResult> task) {

                        if (task.isSuccessful()) {

                            Intent intent = new
Intent(LoginActivity.this, MainActivity.class);

                            startActivity(intent);

                        }else{

Toast.makeText(LoginActivity.this, "У вас есть ошибки",
Toast.LENGTH_SHORT).show();

                        }

                    }

                });

        }

    }

});

}

}

```

Файл SymptomsChecker.java

```
package com.example.diploma2;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.Button;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.TextView;
import android.widget.Toast;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import org.jetbrains.annotations.NotNull;
import java.util.Arrays;
import java.util.Calendar;
import java.util.Date;

public class SymptomsChecker extends AppCompatActivity {

    private Question[] questionBank = new Question[] {
        // array of objects of class Question
        // providing questions from string
        // resource and the correct ans
        new Question(R.string.Q1),
        new Question(R.string.Q2),
        new Question(R.string.Q3),
        new Question(R.string.Q4),
        new Question(R.string.Q5),
        new Question(R.string.Q6),
        new Question(R.string.Q7),
        new Question(R.string.Q8),
    };

    private String result = "";
    private TextView textViewQuestion;
    private TextView textViewQuestionCount;
    private RadioGroup rbGroup;
    private RadioButton rb1;
    private RadioButton rb2;
    private Button buttonConfirmNext;
```

					ПГУ 1.090301.33.001 ПЗ	Лист
						55
Изм.	Лист	№ докум.	Подпись	Дата		

```

private int questionCounter=0;
    private int questionCountTotal=8;
    private int historyId;

    FirebaseDatabase database = FirebaseDatabase.getInstance();
    String currentuser = FirebaseAuth.getInstance().getCurrentUser().getUid();
    DatabaseReference myRef = database.getReference(currentuser);

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_symptoms_checker);
        textViewQuestion
        findViewById(R.id.text_view_question);
        textViewQuestionCount
        findViewById(R.id.text_view_question_count);
        rbGroup = findViewById(R.id.radio_group);
        rb1 = findViewById(R.id.radio_button1);
        rb2 = findViewById(R.id.radio_button2);
        buttonConfirmNext
        findViewById(R.id.button_confirm_next);

        myRef.child("currentCheck").get().addOnCompleteListener(new
        OnCompleteListener<DataSnapshot>() {
            @Override
            public void onComplete(@NonNull Task<DataSnapshot>
            task) {
                if (!task.isSuccessful()) {
                }
                else {
myRef.child("lastCheck").setValue(task.getResult().getValue());
                }
            }
        });
        myRef.child("ID").addValueEventListener(new ValueEventListener()
        {
            @Override
            public void onDataChange(@NonNull @NotNull DataSnapshot
            snapshot) {
                if(snapshot.exists()) {
                    historyId = snapshot.getValue(Integer.class);
                }
                else {
                    historyId = 0;
                }
            }
        })
    }

```

					ПГУ 1.090301.33.001 ПЗ	Лист
						56
Изм.	Лист	№ докум.	Подпись	Дата		

```

        @Override
        public void onCancelled(@NonNull DatabaseError error) {

        }

    });
    showNextQuestion();

    buttonConfirmNext.setOnClickListener(new
    View.OnClickListener() {
        @Override
        public void onClick(View v) {

            if (rb1.isChecked()) {
                result += "1";
                showNextQuestion();
            } else if (rb2.isChecked()) {
                result += "0";
                showNextQuestion();
            }
            else {
                Toast.makeText(SymptomsChecker.this,
                "Пожалуйста выберите ответ", Toast.LENGTH_SHORT).show();
            }
        }

    });

}

private String strResultParser(String result){
    String parsedRes = null;
    for(int i = 0; i < result.length(); i++)
    {
        switch (i){
            case 0:
                if(result.charAt(i) == '1')
                {
                    parsedRes = "Ваша температура выше 38
градусов\n";
                }
                else
                    parsedRes = "Ваша температура ниже 38\n";
                break;
            case 1:
                if(result.charAt(i) == '1')
                {
                    parsedRes += "Присутствует кашель\n";
                }
                else
                    parsedRes += "Отсутствует кашель\n";
                break;
        }
    }
}

```

```

case 2:
    if(result.charAt(i) == '1')
    {
        parsedRes += "Присутствует усталось\n";
    }
    else
        parsedRes += "Отсутствует усталость\n";
    break;
case 3:
    if(result.charAt(i) == '1')
    {
        parsedRes += "Чувство вкуса отсутствует\n";
    }
    else
        parsedRes += "Чувство вкуса в норме\n";
    break;
case 4:
    if(result.charAt(i) == '1')
    {
        parsedRes += "Обоняние отсутствует\n";
    }
    else
        parsedRes += "Обоняние в норме\n";
    break;
case 5:
    if(result.charAt(i) == '1')
    {
        parsedRes += "Нос заложен\n";
    }
    else
        parsedRes += "Нос не заложен\n";
    break;
case 6:
    if(result.charAt(i) == '1')
    {
        parsedRes += "Частота дыхания > 22/мин\n";
    }
    else
        parsedRes += "Частота дыхания < 22/мин\n";
    break;
case 7:
    if(result.charAt(i) == '1')
    {
        parsedRes += "Насыщение крови кислородом
<95%\n";
    }
    else
        parsedRes += "Насыщение крови кислородом в
норме\n";
    break;
}

}

return parsedRes;

```

```

}

private int severityResultParser(String result){
    int severity = 0;
    for(int i = 0; i < result.length(); i++)
    {
        switch (i){
            case 1:
            case 2:
            case 3:
            case 4:
            case 5:
                if(result.charAt(i) == '1')
                {
                    severity++;
                }
                break;
            case 0:
            case 6:
            case 7:
                if(result.charAt(i) == '1')
                {
                    severity+= 7;
                }
                break;
        }
    }
    return severity;
}

private void showNextQuestion() {

    if (questionCounter < questionCountTotal) {

textViewQuestion.setText(questionBank[questionCounter]
                            .getAnswerResId());
        questionCounter++;
        textViewQuestionCount.setText("Вопрос:      "      +
questionCounter + "/" + questionCountTotal);
        buttonConfirmNext.setText("Ответить");
        if(questionCounter == questionCountTotal)
        {
            buttonConfirmNext.setText("Завершить
тестирование");
        }
        else {
            String      currentDateTimeString      =
java.text.DateFormat.getDateTimeInstance().format(new Date());

```

```

myRef.child("currentCheck").get().addOnCompleteListener(new
OnCompleteListener<DataSnapshot>() {
    @Override
    public void onComplete(@NonNull
Task<DataSnapshot> task) {

if (!task.isSuccessful()) {

        }
        else {

myRef.child("lastCheck").setValue(task.getResult().getValue());
        }
    }
});

myRef.child("currentCheck").setValue(severityResultParser(result
));

myRef.child("CheckHistory").child(String.valueOf(historyId)).set
Value(strResultParser(result));

myRef.child("DateHistory").child(String.valueOf(historyId)).setV
alue(String.valueOf(historyId)+". " + currentDateTimeString);
    historyId++;
    myRef.child("ID").setValue(historyId);
    Intent intent = new Intent (SymptomsChecker.this,
MainActivity.class);
    startActivity(intent);
}
}
}

```

Файл InfoFragment.java

```

package com.example.diploma2.ui.dashboard;

import android.os.Bundle;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.ScrollView;

import android.widget.TextView;

import androidx.annotation.NonNull;

```

```

import androidx.annotation.Nullable;

import androidx.fragment.app.Fragment;

import androidx.lifecycle.Observer;

import androidx.lifecycle.ViewModelProvider;


import com.example.diploma2.MainActivity;

import
com.example.diploma2.databinding.FragmentStatisticsBinding;


import retrofit2.Call;

import retrofit2.Callback;

import retrofit2.Response;


public class StatisticsFragment extends Fragment {

    private StatisticsViewModel statisticsViewModel;

    private FragmentStatisticsBinding binding;

    String result = "";

    String info ="Вирус SARS-CoV-2 распространяется в
микроскопических частицах жидкости, выделяемых больным
человеком во время кашля, чихания, разговора, пения или
тяжелого дыхания и попадает на слизистые носа, рта, либо глаз
другого человека.\nКроме того, вирус может также
распространяться контактно-бытовым путем, когда частицы с
вирусом, находящиеся на поверхностях или предметах, через руки
попадают на слизистые респираторного тракта.\nПередача инфекции
аэрозольным путем происходит в закрытых, переполненных и плохо
вентилируемых помещениях.\n Воздушно-пылевой (аэрозольный) путь
передачи реализуется за счет размер которых <5 мкм. Аэрозоли за
счет своего малого размера могут переноситься на большие
расстояния и оставаться в воздухе в течение нескольких часов.
Необходимо помнить, что аэрозоль-продуцирующие аппараты
(небулайзеры, галокамера и т.д.) запрещены к применению в
учреждениях здравоохранения в период пандемии COVID-19.";

```

					ПГУ 1.090301.33.001 ПЗ	Лист
						61
Изм.	Лист	№ докум.	Подпись	Дата		

```

        public View onCreateView(@NonNull LayoutInflater inflater,
                                   ViewGroup container, Bundle
savedInstanceState) {

        statisticsViewModel =

            new
ViewModelProvider(this).get(StatisticsViewModel.class);

        binding = FragmentStatisticsBinding.inflate(inflater,
container, false);

        View root = binding.getRoot();


        final TextView info = binding.usefulInfo;

        final ScrollView scrollView = binding.SCROLLERID;

statisticsViewModel.getText().observe(getViewLifecycleOwner(), new
Observer<String>() {

        @Override

        public void onChanged(@Nullable String s) {

            MainActivity.NetworkService.getInstance()

                .getJSONApi()

                .getPostWithID("Russia")

                .enqueue(new
Callback<MainActivity.Post>() {

                    @Override

                    public void onResponse(@NonNull
Call<MainActivity.Post> call, @NonNull Response<MainActivity.Post>
response) {

```

```

MainActivity.Post      post      =
response.body();

result="Всего заболевших: " +
post.getCases() + "\n";

result+="Всего смертей: " +
post.getDeaths() + "\n";

result+="Больных на данный
момент: " + post.getActive() + "\n";

result+="Выздоровело: " +
post.getRecovered() + "\n";

```

info.setText(result+"\n\n"+"1. При каких симптомах к пациенту на дом обязан прийти врач из районной поликлиники?\n\nВрача на дом можно вызвать, если есть повышение температуры – выше 37,2 градуса и/или признаки ОРВИ: кашель, заложенность носа, затрудненное дыхание, одышка, боли в груди, слабость, головная боль, потеря обоняния.\n\nКроме того, есть условия, когда вызов врача на дом при таких симптомах не просто желателен, а строго необходим. Это:\n\n1. возраст пациента старше 65 лет;\n2. сопутствующие патологии, то есть хронические болезни, при которых повышается риск тяжелых осложнений COVID-19. В первую очередь это сахарный диабет, сердечно-сосудистые заболевания, хронические болезни почек, органов дыхания, онкологические заболевания;\n3. контакт с лицами, у которых диагностирован COVID-19, либо приезд из региона с напряженной эпидситуацией по распространению коронавирусной инфекции.\n! Обязательно сообщите об этих фактах, вызывая врача.\n\n2. В течение какого времени должен прийти участковый терапевт из «районки»? \n\nПо законодательству врач, вызванный на дом, должен прийти в течение 24 часов.\n\n3. Когда звонить в "скорую"? \n\nСегодня диспетчеры службы скорой медицинской помощи (СМП) занимаются распределением вызовов. Поэтому сюда можно звонить, если чувствуете, что состояние ухудшается, а ждать врача на дом еще долго. Либо сразу при появлении тяжелых симптомов: температура выше 38,5 градусов, боли в груди, чувство нехватки воздуха и т.д.\n\nПри поступлении вызова на пульт СМП оператор проводит опрос позвонившего. Кроме жалоб важно сразу назвать свой возраст, сопутствующие болезни и сообщить, если были контакты с зараженными коронавирусом. По итогам опроса принимается одно из трех решений:\n\n1. отправить к пациенту бригаду "скорой". По правилам она должна приехать в течение 20 минут, если есть угроза жизни больного (в условиях сложной эпидобстановки и большого потока вызовов возможны задержки);\n2. передать вызов в службу неотложной помощи. Если состояние пациента не угрожает его жизни, но требует медпомощи в ближайшее время, в течение двух часов должна прибыть бригада «неотложки»;\n3. передать вызов в поликлинику по месту жительства для прихода врача на дом (в течение 24 часов). Такой вариант возможен, если после консультации врача "скорой" по телефону выяснится, что состояние пациента не вызывает серьезных опасений и не требуется ни экстренная, ни неотложная помощь.\n4.

Что должны сделать прибывшие медики?\n\nПо правилам осмотр пациента включает:\n\nсбор информации о ходе заболевания, наличии хронических заболеваний, аллергических реакций;\nосмотр кожных покровов и видимых слизистых (нос, горло);\nизмерение температуры тела, пульса, артериального давления;\nизмерение частоты дыхания, аускультация («прослушивание») легких и сердца;\nизмерение сатурации (уровня кислорода в крови);\nизмерение сахара крови (при необходимости, в частности, у больных сахарным диабетом).");

}

@Override

public void onFailure(@NonNull Call<MainActivity.Post> call, @NonNull Throwable t) {

info.append("Error occurred while getting request!");

t.printStackTrace();

}

});

}

});

return root;

}

@Override

public void onDestroyView() {

super.onDestroyView();

binding = null;

}

Файл DiaryFragment.java

```
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.Fragment;
import androidx.lifecycle.Observer;
import androidx.lifecycle.ViewModelProvider;

import com.example.diploma2.ListItemDisplayler;

import com.example.diploma2.MainActivity;
import com.example.diploma2.R;
import com.example.diploma2.SymptomsChecker;
import com.example.diploma2.databinding.FragmentHomeBinding;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.google.gson.annotations.Expose;
import com.google.gson.annotations.SerializedName;

import org.jetbrains.annotations.NotNull;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;
import retrofit2.http.GET;
import retrofit2.http.Path;

public class HomeFragment extends Fragment {
    private HomeViewModel homeViewModel;
    private FragmentHomeBinding binding;
```

```

        FirebaseDatabase database = FirebaseDatabase.getInstance();
        String currentuser =
FirebaseAuth.getInstance().getCurrentUser().getUid();
        DatabaseReference myRef = database.getReference(currentuser);
        public int historyId;

        public View onCreateView(@NonNull LayoutInflater inflater,
                                   ViewGroup container, Bundle
savedInstanceState) {
            homeViewModel =
                new
ViewModelProvider(this).get(HomeViewModel.class);

            binding = FragmentHomeBinding.inflate(inflater, container,
false);
            View root = binding.getRoot();

            final Button button_start_testing =
binding.buttonStartTesting;
            final ListView symptoms_history = binding.symptomsHistory;

            homeViewModel.getText().observe(getViewLifecycleOwner(),
new Observer<String>() {
                @Override
                public void onChanged(@Nullable String s) {

                }
            });
            button_start_testing.setOnClickListener(new
View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    Intent intent = new Intent (getActivity(),
SymptomsChecker.class);
                    startActivity(intent);
                }
            });

            myRef.child("ID").addValueEventListener(new
ValueEventListener() {
                @Override
                public void onDataChange(@NonNull @NotNull DataSnapshot
snapshot) {

                    if(snapshot.exists()) {
                        historyId = snapshot.getValue(Integer.class);
                    }
                    else {
                        historyId = 0;
                    }
                }
            })

```

```

        @Override
        public void onCancelled(@NonNull @NotNull DatabaseError
error) {

            }
        });

        List<String> checkHistory = new ArrayList<>();
        myRef.child("DateHistory").addValueEventListener(new
ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot snapshot) {
                checkHistory.clear();
                if(snapshot.exists()){
                    for (DataSnapshot postSnapshot:
snapshot.getChildren()) {
                        String strbuf =
postSnapshot.getValue(String.class);
                        checkHistory.add(strbuf);
                    }
                    Collections.reverse(checkHistory);
                    ArrayAdapter<String> adapter = new
ArrayAdapter<>(getActivity(),
                        android.R.layout.simple_list_item_1,
checkHistory);
                    symptoms_history.setAdapter(adapter);
                    symptoms_history.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
                        public void onItemClick(AdapterView<?>
parent, View view,
                        int position, long
id) {
                            myRef.child("item").setValue(id);
                            Intent intent = new Intent
(getActivity(), ListItemDisplayer.class);
                            startActivity(intent);
                        }
                    });
                }else{
                    checkHistory.add("Похоже вы ещё не сделали ни
одной записи");
                    ArrayAdapter<String> adapter = new
ArrayAdapter<>(getActivity(),
                        android.R.layout.simple_list_item_1,
checkHistory);
                    symptoms_history.setAdapter(adapter);
                }
            }
        }
    }
}

```

```

        @Override
        public void onCancelled(DatabaseError databaseError) {

        }

    });

    return root;
}

@Override
public void onDestroyView() {
    super.onDestroyView();
    binding = null;
}
}

```

Файл ProfileFragment.java

```

package com.example.diploma2.ui.notifications;

import android.content.Intent;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.Fragment;
import androidx.lifecycle.Observer;
import androidx.lifecycle.ViewModelProvider;

import com.example.diploma2.ListItemDisplay;
import com.example.diploma2.LoginActivity;
import com.example.diploma2.SymptomsChecker;
import com.example.diploma2.databinding.FragmentNotificationsBinding;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;

import org.jetbrains.annotations.NotNull;
import java.util.Collections;

```

```

public class NotificationsFragment extends Fragment {

    private NotificationsViewModel notificationsViewModel;
    private FragmentNotificationsBinding binding;
    FirebaseDatabase database = FirebaseDatabase.getInstance();
    String currentuser =
FirebaseAuth.getInstance().getCurrentUser().getUid();
    DatabaseReference myRef = database.getReference(currentuser);
    private int lastCheck;
    private int currentCheck;

    public View onCreateView(@NonNull LayoutInflater inflater,
                             ViewGroup container, Bundle
savedInstanceState) {
        notificationsViewModel =
            new
ViewModelProvider(this).get(NotificationsViewModel.class);

        binding = FragmentNotificationsBinding.inflate(inflater,
container, false);
        View root = binding.getRoot();

        final Button button_start_testing =
binding.buttonStartTesting;
        final TextView checkResults = binding.checkResults;

        notificationsViewModel.getText().observe(getViewLifecycleOwner(), new
Observer<String>() {
            @Override
            public void onChanged(@Nullable String s) {
            }
        });

        button_start_testing.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {

                Intent intent = new Intent (getActivity(),
LoginActivity.class);
                startActivity(intent);
            }
        });
        myRef.child("currentCheck").addValueEventListener(new
ValueEventListener() {
            @Override
            public void onDataChange(@NonNull @NotNull DataSnapshot
snapshot) {
                if(snapshot.exists()) {
                    currentCheck =
snapshot.getValue(Integer.class);
                    checkResults.setText(ResultParser(currentCheck,
-1));
                }
            }
        });
        myRef.child("lastCheck").addValueEventListener(new ValueEventListener()
{

```

					ПГУ 1.090301.33.001 ПЗ	Лист
						69
Изм.	Лист	№ докум.	Подпись	Дата		

```

        @Override
        public void onDataChange(@NonNull @NotNull
DataSnapshot snapshot) {
            if (snapshot.exists()) {
                lastCheck
snapshot.getValue(Integer.class);
checkResults.setText(ResultParser(currentCheck, lastCheck));
            } else {
            }
        }

        @Override
        public void onCancelled(@NonNull @NotNull
DatabaseError error) {
        }
    });
}
else {
    checkResults.setText("Похоже, вы не ещё не
сделали ни одной записи, чтобы можно было бы судить о вашем состоянии");
}

}

@Override
public void onCancelled(@NonNull @NotNull DatabaseError
error) {
    }
});

return root;
}
private String ResultParser(int currentCheck, int lastCheck){
    String result = "Судя по вашим записям, ";
    if(currentCheck>11){
        result+="вы болеете тяжёлой формой. Советуем вам
вызвать врача. ";
    }
    else if (currentCheck>5)
    {
        result+="вы болеете среднетяжёлой формой. Советуем вам
вызвать врача. ";
    }
}

```

					ПГУ 1.090301.33.001 ПЗ	Лист
						70
Изм.	Лист	№ докум.	Подпись	Дата		

```

        else if (currentCheck>0)
        {
            result+="вы болеете в лёгкой форме. ";
        }
        if(currentCheck == 0)
        {
            result += "вы здоровы. ";
        }
        if(lastCheck == -1){

        }
    else if(currentCheck == lastCheck)
    {
        result+="Ваше состояние стабильно. ";
    }
    else if(currentCheck <lastCheck)
    {
        result+="Ваше состояние улучшилось. ";
    }
    else if(currentCheck > lastCheck)
    {
        result+="Ваше состояние ухудшилось. ";
    }
    result+=
        "\n\n\nЕдиный консультационный центр
Роспотребнадзора: 8 800 555-49-43. \nЕдиный номер для вызова врача или
записи на прививку: 122.";

    return result;
}

@Override
public void onDestroyView() {
    super.onDestroyView();
    binding = null;
}

}

```

Файл RegisterActivity.java

```

package com.example.diploma2;

import android.content.Intent;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

import android.widget.EditText;

import androidx.annotation.NonNull;

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.33.001 ПЗ

Лист

71

```

import androidx.appcompat.app.AppCompatActivity;

import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;

public class RegisterActivity extends AppCompatActivity {

    private EditText email_register;

    private EditText password_register;

    private Button btn_register;

    private FirebaseAuth mAuth;

    private DatabaseReference ref;

    private FirebaseDatabase database;

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_register);

        mAuth = FirebaseAuth.getInstance();

        database = FirebaseDatabase.getInstance();

```

```

ref = database.getReference();

email_register = findViewById(R.id.email_register);

password_register
findViewById(R.id.password_register);

btn_register = findViewById(R.id.registerUser);

btn_register.setOnClickListener(new
View.OnClickListener() {

    @Override

    public void onClick(View v) {

        if
(email_register.getText().toString().isEmpty()
password_register.getText().toString().isEmpty()) {

            Toast.makeText(RegisterActivity.this,
"Fields cannot be empty", Toast.LENGTH_SHORT).show();

        }else{

 mAuth.createUserWithEmailAndPassword(email_register.getText().toStrin
g(), password_register.getText().toString())

                .addOnCompleteListener(new
OnCompleteListener<AuthResult>() {

                    @Override

                    public void onComplete(@NonNull
Task<AuthResult> task) {

                        if (task.isSuccessful()) {

ref.child("Users").child(mAuth.getCurrentUser().getUid()).child("email
1").setValue(email_register.getText().toString());ref.child("Users").
child(mAuth.getCurrentUser().getUid()).child("password").setValue(pas
sword_register.getText().toString());

```

```

Intent intent = new
Intent (RegisterActivity.this, MainActivity.class);

startActivity(intent);

}else{

Toast.makeText (RegisterActivity.this, "You have some errors",
Toast.LENGTH_SHORT).show();

}

}

});

}

}

}

}

}

```

					ПГУ 1.090301.33.001 ПЗ	Лист
						74
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение В

(Обязательное)

Мобильное приложение ведения дневника заболевшего

Презентация

					ПГУ 1.090301.33.001 ПЗ	Лист
						75
Изм.	Лист	№ докум.	Подпись	Дата		

Тема работы : Мобильное приложение ведения дневника заболевшего

Научный руководитель: д.т.н. доцент Митрохин М.А.

Выполнил: студент гр. 18BB1 Репин Илья

Цель работы

- Целью дипломной работы является разработка приложения для ведения дневника заболевшего, предназначенное для самодиагностики и отслеживания динамики развития заболевания. Это достигается путем сбора данных с устройства в процессе прохождения опроса и занесение этих данных Firebase RealtimeDatabase для хранения и дальнейшего сбора статистики.

Изм.	Лист	№ докум.	Подпись	Дата

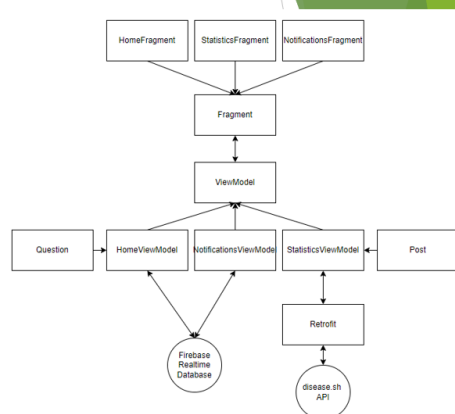
Средства разработки

- ▶ Java
- ▶ Android Studio
- ▶ Firebase
- ▶ Retrofit



Архитектура приложения

1. **Fragment.** Отвечает за взаимодействие между пользователем и приложением. Данный элемент отображает данные, полученные от *ViewModel*, или собирает информацию о действиях пользователя и передаёт их в *ViewModel*.
2. **ViewModel.** Класс, который будет использоваться в качестве посредника между *Fragment* и *Firebase Database* или веб-сервисами для передачи данных.
3. **Firebase Realtime Database.** В данной БД будут храниться данные о состоянии пользователя, его учётной записи и прочая информация.
4. **Disease.sh API.** Используется в качестве источника информации для сбора статистики по заболеваемости в РФ. Информация извлекается путём запроса к API через библиотеку *Retrofit*. Для получения информации в нужном формате был написан класс *Post*.



Изм.	Лист	№ докум.	Подпись	Дата

Аутентификация и регистрация

SDiary

SDiary

Stay healthy

Email address

Password

LOGIN

Forgot password?

Create account

SDiary

SDiary

Stay healthy

Email address

Password

REGISTER USER

Записи

Записи

3. May 25, 2022 11:10:56 AM

2. May 25, 2022 11:10:38 AM

1. May 23, 2022 3:11:00 PM

0. May 23, 2022 2:58:21 PM

ДОБАВИТЬ ЗАПИСЬ

Записи

Полезное

Профиль

Ваша температура выше 38 градусов
Присутствует кашель
Присутствует усталость
Чувство вкуса отсутствует
Обоняние в норме
Нос не заложен
Частота дыхания < 22/мин
Насыщение крови кислородом в норме

Полезная информация и статистика

Полезное

Всего заболевших: 18355200
Всего смертей: 379584
Больных на данный момент: 207863
Выздоровело: 17767753

1. При каких симптомах к пациенту на дом обязан прийти врач из районной поликлиники?

Врача на дом можно вызывать, если есть повышение температуры – выше 37,2 градуса и/или признаки ОРВИ: кашель, заложенность носа, затрудненное дыхание, одышка, боли в груди, слабость, головная боль, потеря обоняния.

Кроме того, есть условия, когда вызов врача на дом при таких симптомах просто желателен, а строго необходим. Это:

возраст пациента старше 65 лет;
сопутствующие патологии, то есть хронические болезни, при которых повышается риск тяжелых осложнений COVID-19. В первую очередь это сахарный диабет, сердечно-сосудистые заболевания, хронические болезни почек, органов дыхания, онкологические заболевания; контакт с лицами, у которых диагностирован COVID-19, либо приезд из региона с напряженной эпидемиологической по распространению коронавирусной инфекции.
! Обязательно сообщите об этих фактах, вызывая врача.

Закрепить
Полезное
Профиль

Профиль

Судя по вашим записям, вы болеете тяжелой формой. Советуем вам вызвать врача. Ваше состояние стабильно.

Единый консультационный центр Роспотребнадзора:
8 800 555-49-43.
Единый номер для вызова врача или записи на прививку: 122.

СМЕНИТЬ ПОЛЬЗОВАТЕЛЯ

Закрепить
Полезное
Профиль

Заключение

- ▶ Приложение обладает простым и понятным интерфейсом
- ▶ Приложение позволяет пользователю оставлять записи о своём самочувствии и отслеживать динамику развития своего заболевания
- ▶ Приложение полностью соответствует поставленной цели

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.33.001 ПЗ

Лист

79

Спасибо за внимание!

					ПГУ 1.090301.33.001 ПЗ	Лист
						80
Изм.	Лист	№ докум.	Подпись	Дата		