МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное

учреждение высшего образования

«Ижевский государственный технический университет имени М.Т. Калашникова»

(ФГБОУ ВО «ИжГТУ имени М.Т. Калашникова»)

Факультет «Информационные технологии»

Кафедра «Автоматизированные системы обработки информации и управления»

Работа защищена с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись\_\_\_\_\_\_\_\_\_\_\_\_\_

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по дисциплине «Программная инженерия»

на тему «Автоматизация деятельности охранной организации»

|  |  |
| --- | --- |
| Выполнили  студенты гр. Б23-780-4 | Исупов Э. А., Илдаров К. Р., Мурашова К. И. |
|  |  |
| Принял  к.т.н., доцент каф. АСОИУ | Касимов Д.Р. |

Рецензия:

степень достижения поставленной цели работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

полнота разработки темы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

уровень самостоятельности работы обучающегося\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

недостатки работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ижевск – 2024

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc185271166)

[1 Разработка спецификации требований 4](#_Toc185271167)

[1.1 Функциональные требования 4](#_Toc185271168)

[1.2 Нефункциональные требования 5](#_Toc185271169)

[1.3 Разработка макетов 5](#_Toc185271170)

[1.4 Use-case диаграммы 8](#_Toc185271171)

[2 Планирование программного проекта по методологии Scrum 12](#_Toc185271172)

[2.1 Mindmap 12](#_Toc185271173)

[2.2 Создание задач для разработчиков 12](#_Toc185271174)

[2.3 Подбор команды разработчиков 13](#_Toc185271175)

[2.4 Приоритезация задач 15](#_Toc185271176)

[2.5 Диаграмма Ганта 17](#_Toc185271177)

[3 Проектирование продукта 19](#_Toc185271178)

[3.1 Диаграмма состояний 19](#_Toc185271179)

[3.2 Диаграмма последовательности 20](#_Toc185271180)

[3.3 Диаграмма классов 23](#_Toc185271181)

[4 Управление дефектами программного продукта и совместная работа над проектом 25](#_Toc185271182)

[4.1. Создание тест-кейсов 25](#_Toc185271183)

[4.2 Структура проекта на GitHub 26](#_Toc185271184)

[4.3 Структура проекта в Visual Studio 27](#_Toc185271185)

[5 Оценка качества программного продукта 29](#_Toc185271186)

[ЗАКЛЮЧЕНИЕ 32](#_Toc185271187)

[ПРИЛОЖЕНИЕ А 33](#_Toc185271188)

# ВВЕДЕНИЕ

В современном мире вопросы обеспечения безопасности жилища приобретают все большее значение. Традиционные методы охраны, такие как установка механических замков и решеток, зачастую оказываются недостаточными перед лицом современных методов взлома и проникновения. В связи с этим, на первый план выходят системы охранной сигнализации, которые позволяют оперативно реагировать на нештатные ситуации и предотвращать возможные негативные последствия. Такие приложения позволяют пользователям в режиме реального времени получать информацию о состоянии охранной системы, оперативно реагировать на тревожные события, а также управлять различными функциями системы, например, постановкой и снятием с охраны и просмотром видео с камер наблюдения.

Разработка программного продукта в рамках данной курсовой работы обусловлена возрастающей потребностью в современных и доступных решениях для обеспечения домашней безопасности. Разработка приложения для мониторинга и управления домашней охранной системой позволит создать удобный и функциональный инструмент, который будет способствовать повышению уровня защищенности жилища и обеспечит пользователям чувство спокойствия и уверенности.

# 1 Разработка спецификации требований

Разрабатываемый программный продукт – приложение для мониторинга и управления домашней охранной системы. Оно должно предоставить пользователям интуитивно понятный интерфейс для контроля безопасности своего дома.

Перед началом разработки был составлен набор первичных требований к продукту, User Story, чтобы обеспечить соответствие потребностям пользователей.

Таблица 1 – User Story, первичные требования клиента

|  |
| --- |
| Как клиент охраняемого объекта я хочу:  1. Иметь возможность просматривать сопряженные камеры наблюдения в real-time режиме.  2. Получать уведомление-предупреждение при срабатывании сигнализации.  3. Иметь возможность удаленного доступа к системе управления наблюдением и сигнализацией (включение/отключение элементов).  4. Иметь возможность быстрого вызова соответствующих оперативных служб нажатием одной кнопки. |

На их основе были разработаны следующие первичные требования:

1) Мониторинг и управление системой безопасности.

2) Управление вызовами.

3) Анализ данных и отчетность.

Процесс разработки требований начался с формирования первичных потребностей пользователя. На основе этой информации были сформулированы функциональные требования, которые описывают, что именно должна делать система. Вместе с ними были определены и нефункциональные требования, которые устанавливают критерии качества, которым должна соответствовать система.

# 1.1 Функциональные требования

1) Активация охранной системы по нажатию кнопки "Включить сигнализацию".

2) Возможность смены вызываемой службы безопасности, вызываемой по кнопке "включить сигнализацию".

3) Возможность вызова службы безопасности по нажатию кнопки "ВЫЗОВ".

4) Отправка уведомления от приложения о срабатывании сигнализации.

5) Просмотр трансляции видео с камер видеонаблюдения.

6) Сбор статистики по активации.

7) Сбор статистики срабатываний сигнализации.

8) Сбор статистики по вызовам службы безопасности.

# 1.2 Нефункциональные требования

1) Анализ должен выводиться в процентном формате на экране "Анализ".

2) Приложение должно быть для устройств на Android (не ниже версии 10).

3) Дизайн приложения должен в стиле IOS устройств (См. дизайн-макет).

4) При длительном нажатии на кнопку "ВЫЗОВ", должен активироваться выпадающий список, позволяющий выбрать службу, которая будет вызываться этой кнопкой.

5) Кнопка связи с техническим консультантом должна быть в самом низу, выделяться серым цветом.

6) На экране просмотра видео с камер, должны быть две функциональные кнопки «ВЫЗОВ» и «ВКЛЮЧИТЬ СИГНАЛИЗАЦИЮ».

# 1.3 Разработка макетов

Также, для представления общего внешнего вида программного продукта, были разработаны дизайн-макеты интерфейса приложения, нескольких экранных форм, которые можно увидеть на рисунках 1.1-1.5. Эти макеты дают наглядное представление о том, как будет выглядеть и функционировать приложение. Они демонстрируют основные элементы интерфейса и пользовательские сценарии.

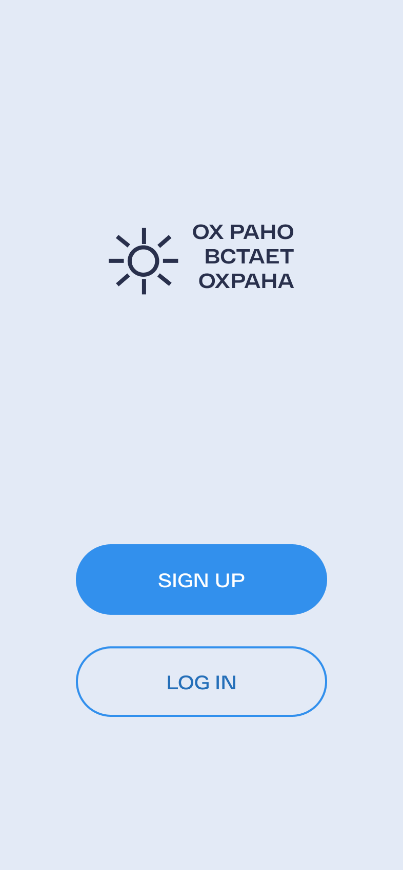


Рисунок 1.1 – Макет страницы входа в приложения

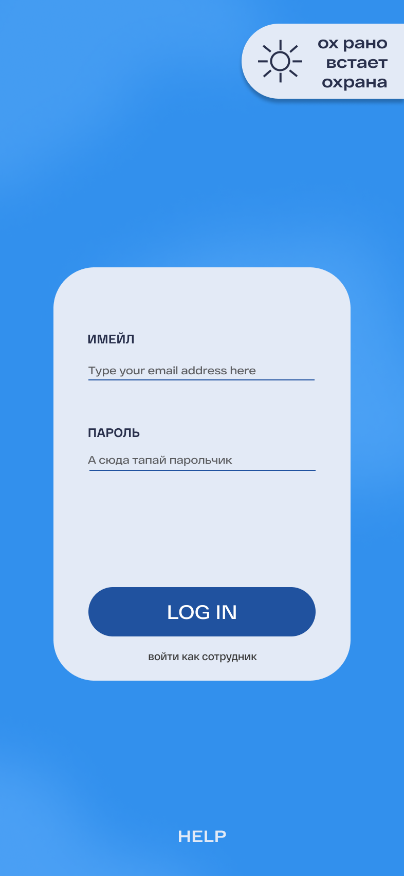


Рисунок 1.2 – Макет страницы авторизации



Рисунок 1.3 – Макет основной страницы пользователя



Рисунок 1.4 – Макет страницы обратной связи



Рисунок 1.5 – «Макет страницы анализа»

# 1.4 Use-case диаграммы

Изучив пожелания пользователей, на их основе были разработаны use-case диаграммы для описания необходимых возможностей программного продукта. На таблице 2, 3 и 4 предоставлены диаграммы для некоторого функционала программы.

Таблица 2 – Use Case просмотра камер

|  |  |
| --- | --- |
| Название | Просмотр изображения с камер |
| Описание | Клиент может выбрать нужную камеру (в т.ч. с помощью фильтров) и просмотреть изображение с нее в реальном времени. |
| Действующие лица | Клиент |
| Предусловия | Клиент зашел в приложение |
| Основной поток событий | 1. Клиент переходит во вкладку со списком сопряженных камер. 2. Клиент выбирает нужную камеру. 3. Система предоставляет изображение с нее. |

Продолжение таблицы 2

|  |  |
| --- | --- |
| Альтернативный поток 1 | 1. На шаге 1 система не отображает список доступных камер. 2. Система выводит на экран сообщение о том, что нет сопряженных камер. |
| Альтернативный поток 2 | 1. На шаге 1 система не отображает список камер по выбранным параметрам. 2. Клиент изменяет параметры поиска камеры. 3. Система отображает новый список доступных камер. 4. Клиент продолжает работу с приложением с шага 2 основного потока событий. |
| Постусловия | Клиент успешно выбрал камеру и получил доступ к изображению с нее. |
| Расширенные атрибуты | 1. Если клиент вводит неверные данные, система сообщает об ошибке и предлагает исправить их. 2. После выбора камеры доступны опции включения/выключения камеры. |
| Диаграмма Use Case |  |
| Рекомендации по реализации | Реализовать удобный и интуитивно понятный интерфейс для выбора камеры (в т.ч. с помощью фильтров). |

Таблица 3 – Use Case управления системой сигнализации

|  |  |
| --- | --- |
| Название | Управление сопряженной системой сигнализации |
| Описание | Клиент может управлять сигнализацией – включать или выключать ее. |
| Действующие лица | Клиент |
| Предусловия | Клиент зашел в приложение |

Продолжение таблицы 3

|  |  |
| --- | --- |
| Основной поток событий | 1. Клиент перешел во вкладку управления сопряженной сигнализацией. 2. Клиент изменил текущее состояние – выключил/включил сигнализацию |
| Альтернативный поток 1 | 1. На шаге 1 система не отображает сопряженную сигнализацию. 2. Система выводит на экран сообщение о том, что нет сопряженных систем безопасности. |
| Постусловия | Клиент успешно получил доступ к сигнализации и управлению ей. |
| Диаграмма Use Case |  |

Таблица 4 – Use Case вызова экстренных служб:

|  |  |
| --- | --- |
| Название | Вызов экстренных служб |
| Описание | Клиент может вызвать определенную экстренную службу. |
| Действующие лица | Клиент |
| Предусловия | Клиент получил уведомление о срабатывании сигнализации. |
| Основной поток событий | 1. Клиент перешел во вкладку вызова экстренных служб. 2. Система предоставила список доступных служб для вызова. 3. Клиент выбрал нужную службу. 4. Система предоставила выбор (звонок диспетчеру или отправка экстренного сообщения). 5. Клиент выбирает 1 из доступных опций (шаг 4) и вызывает экстренную службу. |
| Постусловия | Клиент успешно вызвал экстренную службу. |

Продолжение таблицы 4

|  |  |
| --- | --- |
| Диаграмма Use Case |  |

# 2 Планирование программного проекта по методологии Scrum

# 2.1 Mindmap

Перед началом разработки было решено использовать гибкую методологию Scrum для организации проекта и процессов разработки. Такой подход позволит оперативно реагировать на изменения требований и обеспечит прозрачность на всех этапах. Для визуализации задач и определения необходимых шагов для запуска проекта была составлена Mindmap (ментальная карта), отвечающая на вопрос "Что нужно сделать, чтобы запустить проект?". Карта, которую можно увидеть на рисунке 2.1, стала отправной точкой для планирования спринтов и распределения задач внутри команды.

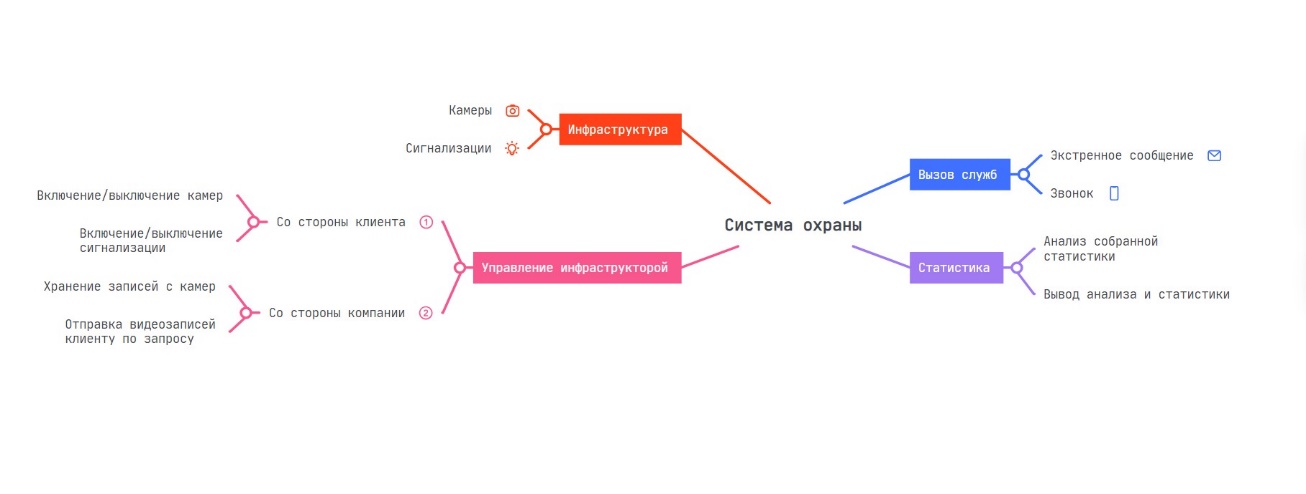


Рисунок 2.1 – Mindmap проекта системы охраны

# 2.2 Создание задач для разработчиков

На основе анализа требований были определены и сформулированы следующие задачи для разработчиков. Эти задачи далее будут распределены по приоритету и сложности, что позволит организовать эффективный процесс разработки и контролировать выполнение каждого этапа.

Таблица 5 – Список задач для разработчиков

|  |
| --- |
| Создать пользовательский интерфейс для выпадающего списка с доступными службами безопасности. |
| Разработать кнопку "Включить сигнализацию", которая активирует охранную систему. |
| Разработка API для активации охранной системы |
| Реализовать систему отправки уведомлений на устройство пользователя при срабатывании сигнализации. |
| Реализовать обработчик события клика по кнопке для отправки запроса на активацию сигнализации. |

Продолжение таблицы 5

|  |
| --- |
| Добавить кнопку связи с технической поддержкой в нижнюю часть экрана. Кнопка должна выделяться серым цветом, чтобы быть заметной, но не отвлекающей. |
| Создать кнопку "Вызвать службу" |
| Добавить возможность просмотра в реальном времени видеопотоков с камер видеонаблюдения. |
| Видеотрансляции должны быть защищены и доступны только авторизованным пользователям |
| Реализация функционала связи с технической поддержкой |
| Разработать систему авторизации пользователей |

# 2.3 Подбор команды разработчиков

Прежде чем приступать к разработке, требовалось сформировать эффективную команду, определив роли, уровни квалификации и круг обязанностей каждого участника. Этот шаг напрямую связан с бюджетом проекта, поэтому был составлен список необходимых специалистов с подробным расчетом их заработной платы, что позволило оценить необходимые финансовые ресурсы и оптимизировать расходы.

Таблица 6 – Подобранная команда разработчиков

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Должность | Уровень | Зарплата (руб/мес) | Зарплата (руб/час) | Обязанности |
| C# Backend разработчик | Junior | 70000 | 416.6 | 1. Написание кода на C# для серверной части приложения (API, базы данных, интеграции).  2. Поддержка существующего кода.  3. Написание юнит-тестов для серверных компонентов. |

Продолжение таблицы 6

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C# Backend разработчик | Middle | 150000 | 892.8 | 1. Написание кода на C# для серверной части приложения.  2. Поддержка существующего кода.  3. Написание юнит-тестов.  4. Решение возникающих проблем и кураторство junior-разработчика. |
| Frontend разработчик (HTML+CSS+JS+Vue.js) | Junior | 60000 | 357.14 | 1. Разработка пользовательских интерфейсов: Верстка интерфейсов (html+css+js).  2. Интеграция клиентского приложения с бэкендом. |
| Frontend разработчик (HTML+CSS+JS+Vue.js) | Middle | 140000 | 833.3 | 1. Верстка интерфейсов с использованием HTML, CSS, JavaScript и фреймворка Vue.js.  2. Решение вопросов с дизайнером.  3. Интеграция клиентского приложения с бэкендом. |
| Designer | Junior | 80000 | 476.19 | 1. Создание дизайна приложения на основе ТЗ заказчика.  2. Коммуникация с фронтенд-разработчиками.  3. Тестирование макета. |

Продолжение таблицы 6

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Teamleader (Senior C# Fullstack developer) | Senior | 294956 | 1755 | 1. Разработка юзкейсов для тестирования.  2. Написание юнит-тестов для серверных компонентов.  3. Организация работы команды разработчиков, распределение задач, контроль сроков.  4. Разработка архитектуры приложения. |
| Ручной тестировщик | Студент | 40000 | 238 | 1. Уборка в гараже.  2. Проведение ручного тестирования веб-приложений.  3. Разработка сценариев тестирования (функциональное, регрессионное, интеграционное и другие виды).  4. Запись и отслеживание багов в Trello.  5. Проверка работы функционала приложения согласно требованиям. |

От должности Unit Tester (C#) было решено отказаться, так как из-за небольшого объема проекта потребность в детальном юнит-тестировании отсутствует.

Наем аналитика также нецелесообразен, поскольку разработка программного обеспечения ведется строго в соответствии с предоставленным заказчиком техническим заданием.

# 2.4 Приоритезация задач

Далее был разработан список задач, расставленных в порядке приоритетности. Приоритеты обозначены числовой шкалой:

* 0 – соответствует наивысшему приоритету (задачи, которые необходимо выполнить в первую очередь)
* 1 – соответствует среднему уровню приоритета (задачи все еще важны, но уже не так сильно)
* 2 – наименьшему (задачи, выполнение которых можно отложить).

Таблица 7 – Список задач для разработчиков по приоритетам

|  |  |  |  |
| --- | --- | --- | --- |
| Приоритет | Область | Задача | Время (часы) |
| 0 | BACKEND | Разработать систему защищенного доступа к видеотрансляции для авторизованных пользователей | 20 |
| 0 | BACKEND | Добавить возможность просмотра в реальном времени видеопотоков с камер видеонаблюдения. | 12 |
| 0 | BACKEND | Разработка API для активации охранной системы. | 15 |
| 1 | FRONTEND | Разработать кнопку "Включить сигнализацию", которая активирует охранную систему. | 4 |
| 1 | FRONTEND | Реализовать обработчик события клика по кнопке для отправки запроса на активацию сигнализации. | 5 |
| 1 | FRONTEND | Создать кнопку "Вызывать службу". | 4 |
| 1 | BACKEND | Реализовать систему отправки уведомлений на устройство пользователя при срабатывании сигнализации. | 12 |
| 1 | BACKEND | Разработать систему авторизации пользователей. | 9 |
| 2 | FRONTEND | Создать пользовательский интерфейс для выпадающего списка с доступными службами безопасности. | 8 |
| 2 | FRONTEND | Добавить кнопку связи с технической поддержкой в нижнюю часть экрана. (Кнопка должна выделяться серым цветом, чтобы быть заметной, но не отвлекающей.) | 2 |
| 2 | FRONTEND | Реализация функционала связи с технической поддержкой. | 6 |

# 2.5 Диаграмма Ганта

С целью оптимизации процесса разработки, задачи были декомпозированы на итерации продолжительностью две недели (спринты). На основании проведенной оценки трудозатрат была сформирована диаграмма Ганта, представленная на рисунках 2.2–2.5.

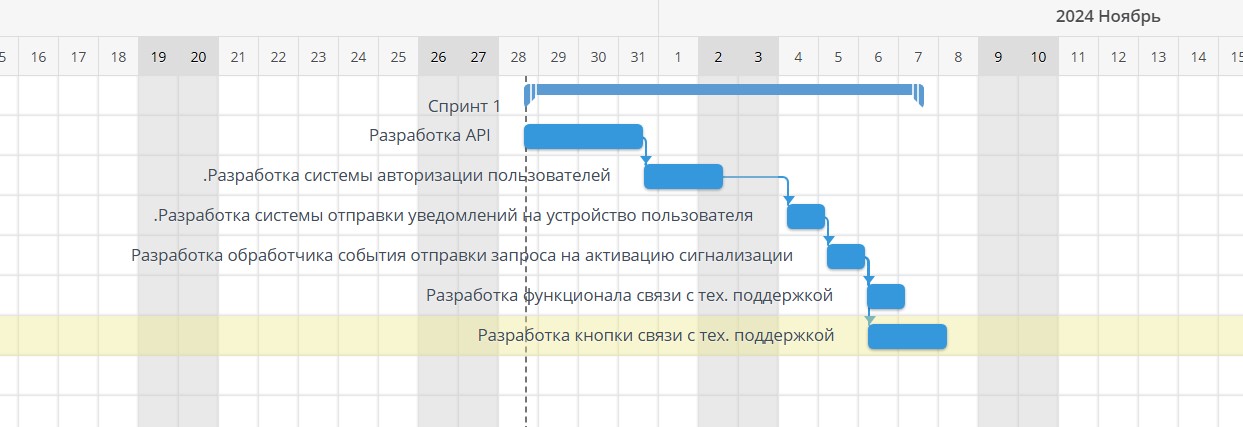


Рисунок 2.2 – Диаграмма Ганта по первому спринту

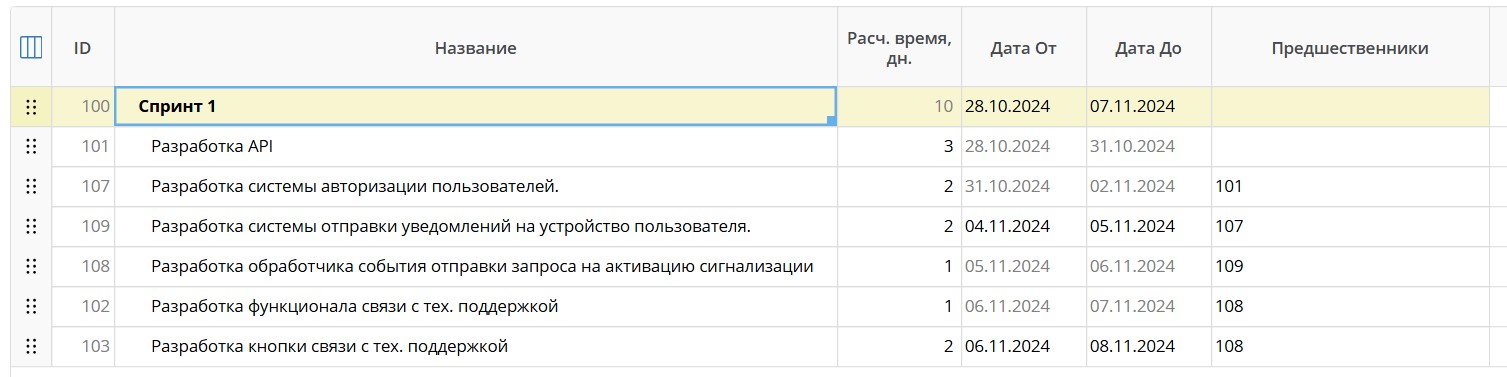


Рисунок 2.3 – Подробная информация Диаграммы Ганта по первому спринту

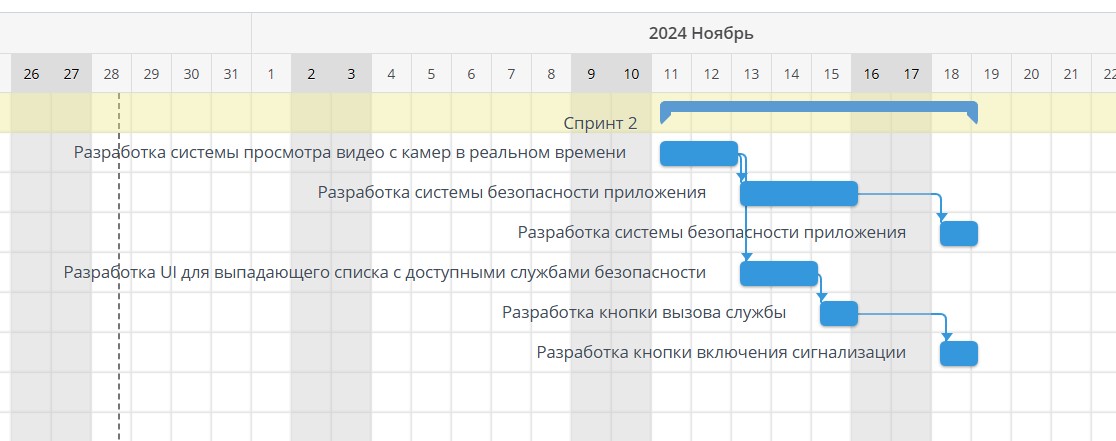


Рисунок 2.4 – Диаграмма Ганта по второму спринту

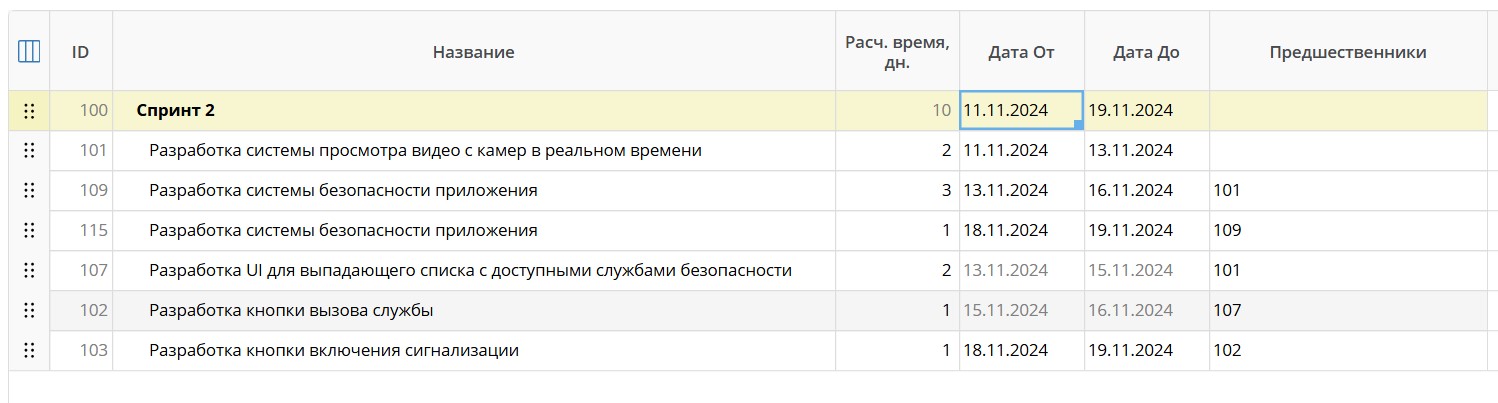


Рисунок 2.5 – Подробная информация Диаграммы Ганта по второмуспринту

В общей сложности вышло, что суммарная стоимость всего проекта с учетом зарплат сотрудникам и иных необходимых вложений – 435,724 рубля.

# 3 Проектирование продукта

В ходе разработки программного продукта было проведено его проектирование. Были созданы UML- программного диаграммы состояний (state machine diagram), последовательности (sequence diagram) и классов (class diagram).

3.1 Диаграмма состояний

На основе основного объекта деятельности разрабатываемого ПО, была составлена UML-диаграмма состояний, приведенная на рисунке 3.1. Данная диаграмма описывает состояния системы при срабатывании сигнализации и включает в себя следующие состояния:

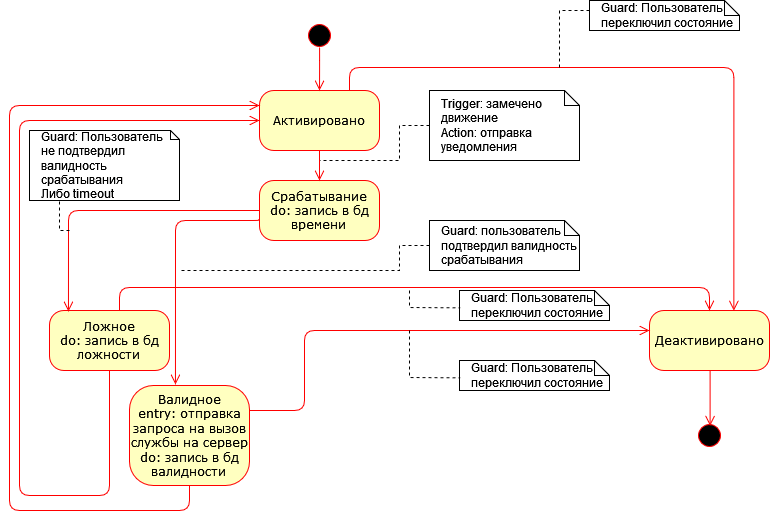
* Активировано – система активирована;
* Срабатывание – реакция системы на активирующие сигнализацию события;
* Ложное – ложное срабатывание системы;
* Валидное – валидное срабатывание системы;
* Деактивировано – система деактивирована.

Рисунок 3.1 – UML-диаграмма состояний системы

Начальное состояние системы – активирована, из данного состояния возможны только 2 перехода: в состояние “срабатывание” при срабатывании соответствующего триггера “замечено движение”, при котором система отправит уведомление о произошедшем событии на устройство пользователя, а также сделает соответствующую запись о событии в БД; в неактивное состояние “деактивировано” в случае, если пользователь переключит состояние системы вручную, т.е деактивирует систему.

Из состояния “срабатывание” также возможны 2 перехода в состояния ложного и валидного срабатывания. Срабатывание “ложное”, при условии, что пользователь не подтвердит валидность срабатывания системы или ответ от него не будет получен за некоторое определенное время, т.е произойдет timeout. Срабатывание “валидное”, если оно не является ложным, т.е пользователь обязан подтвердить валидность срабатывания системы за некоторое определенное время, чтобы срабатывание считалось валидным, иначе произойдет timeout, что приведет к переходу системы в состояние “ложное”. При валидном срабатывании система сделает соответствующую запись в БД, а также отправит запрос на вызов службы на сервер. Из двух данных состояний возможен переход в состояние “деактивировано” при ручном переключении состояния пользователем, иначе – автоматический переход системы в состояние “активировано”.

# 3.2 Диаграмма последовательности

Затем была составлена диаграмма последовательности, описывающая процессы, происходящие в системе при ее срабатывании – реакция на триггер. Диаграмма приведена на рисунке 3.2.

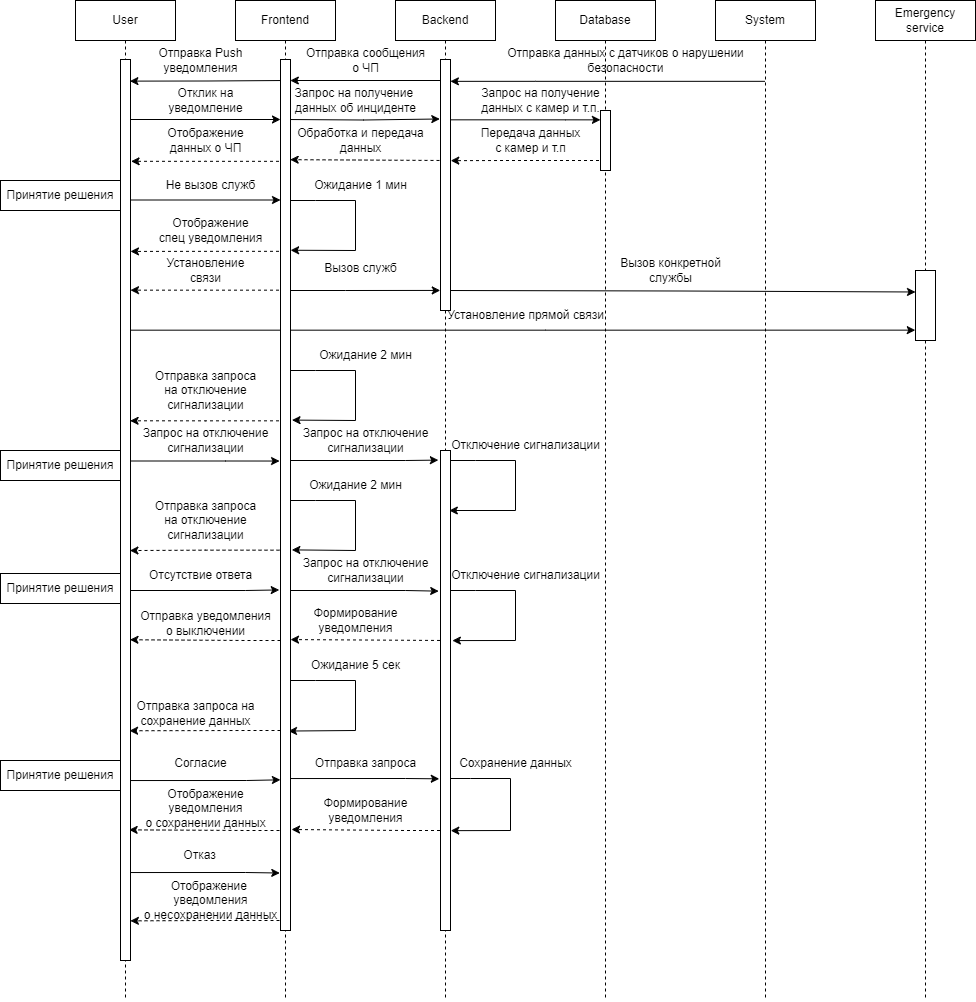


Рисунок 3.2 – Диаграмма последовательности выполняемых процессов

Диаграмма содержит 6 задействованных в данном процессе объектов: user – пользователь, frontend, backend, database – база данных, system – физические элементы системы безопасности (камера, сигнализация), emergency service – экстренная служба, с которой устанавливается связь.

Описание процесса начинается с получения системой триггера о нарушении безопасности на охраняемом объекте – объект system отправляет полученные данные с физических элементов системы объекту backend, который в свою очередь отправляет сообщение о произошедшем ЧП объекту frontend. Frontend формирует и оправляет сообщение пользователю о происшествии в виде push-уведомления.

Далее ожидается отклик от пользователя, т.е ожидается, что user отреагирует на уведомление, на что, в свою очередь, немедленно среагирует frontend отправкой запроса на получение данных об инциденте объекту backend. Backend отправит запрос на получение записанных в БД данных с физических элементов системы базе данных. После обработки запроса database вернет запрашиваемые данные объекту backend, который их обработает и передаст объекту frontend. Frontend сформирует полученные данные и отобразит их пользователю.

В случае, если пользователь не реагирует на уведомление о происшествии в течение определенного времени (1 минута) frontend сформирует специальное уведомление о вызове экстренных служб, которое затем отправит пользователю, и, в то же время, frontend отправит запрос объекту backend на вызов экстренных служб. Backend сформирует необходимые данные о происшествии и вызовет экстренную службу, после чего данные будут отправлены диспетчеру, а также будет установлено прямое соединение между объектами user и emergency service. После этого управление системой вернется к объекту frontend и по истечении определенного времени (2 минуты) будет отправлен запрос пользователю на отключение сработавшей сигнализации.

В случае, если пользователь примет решение об отключении, frontend отправит соответствующий запрос объекту backend, который выключит сигнализацию.

В случае, если frontend не получит ответ от пользователя, то система автоматически отключит сигнализацию – frontend отправит запрос объекту backend, который выключит сигнализацию, после чего backend сформирует уведомление и отправит его объекту frontend. Затем frontend отправит уведомление пользователю о том, что сигнализация была выключена.

Вне зависимости от принятого пользователем решения frontend сформирует запрос о сохранении данных о ЧП и отправит его пользователю по прошествии 5 секунд.

В случае, если пользователь согласится на сохранение данных, frontend сформирует соответствующий запрос и отправит его объекту backend, который в свою очередь сохранит данные. Затем будет сформировано уведомление о сохранении данных (успешно выполнено или не выполнено), которое будет отправлено пользователю.

В случае отказа от сохранения данных будет сформировано соответствующее уведомление объектом frontend, которое затем будет отправлено пользователю.

# 3.3 Диаграмма классов

Позже была составлена диаграмма классов системы, представленная ниже на рисунке 3.3. Данная диаграмма включает в себя 8 классов: MotionSensor – класс сигнализации, Camera – класс камеры, NotificationService – класс отправитель оповещений, EmergencyService – класс связующий элемент системы и экстренной службы, EmergencyObj – класс экстренной службы, GuardedObject – класс охраняемого объекта, User – класс пользователя, AnalysisSystem – класс анализатора данных; а также интерфейс IDevice – интерфейс физических элементов системы (motion sensor и camera).

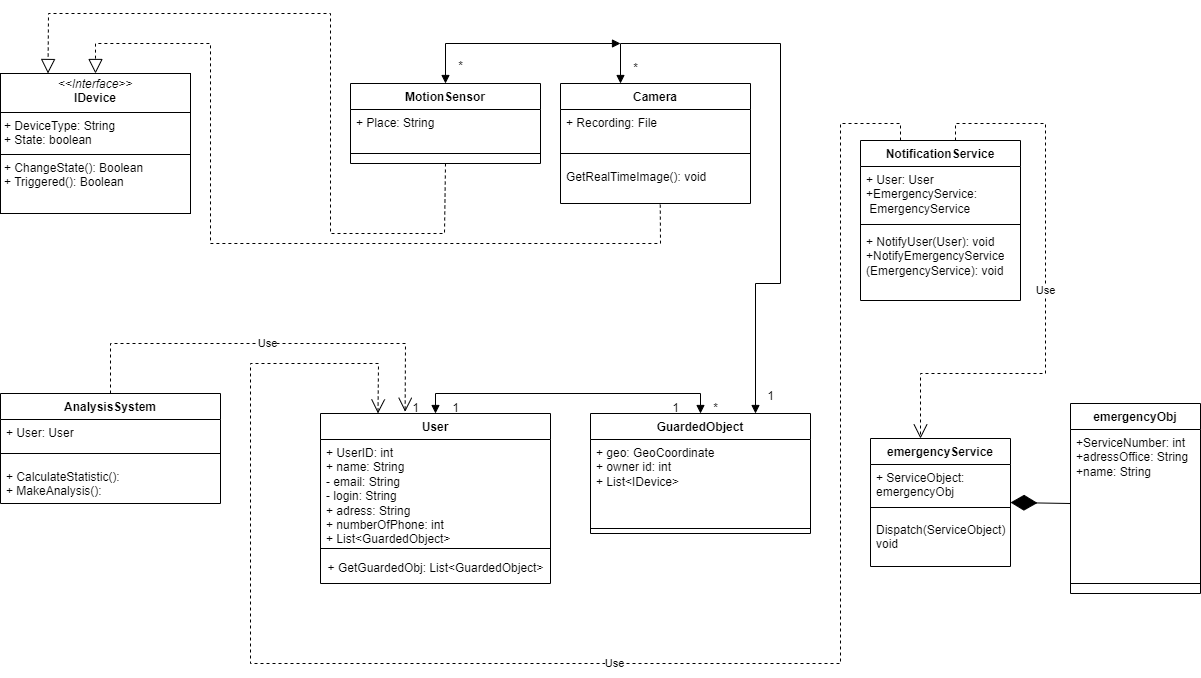


Рисунок 3.3 – Диаграмма классов backend части приложения

Класс motion sensor содержит поле Place типа string – местоположение физического датчика сигнализации в системе и метод Triggered() типа boolean – метод, позволяющий узнать текущее состояние датчика: сработал или находится в покое.

Класс camera содержит поле Recording типа file (media file) – текущая запись и метод GetRealTimeImage() - метод, позволяющий получить изображение с камеры наблюдения в реальном времени.

Классы camera и motion sensor реализуют интерфейс IDevice, который предписывает им содержать поля DeviceType типа string – тип устройства, State типа boolean – текущее состояние устройство (включено или выключено) и методы ChangeState() типа Boolean – метод включения/выключения устройства, Triggered() типа boolean, позволяющий узнать текущее состояние объекта: устройство среагировало на несанкционированные действия или находится в покое.

Класс NotificationService содержит поля: User типа User – пользователь данной системы, EmergencyService типа EmergencyService – связующий элемент между системой и экстренной службой; методы: NotifyUser типа void – метод, принимающий в качестве аргумента пользователя User, который формирует и отправляет оповещения пользователю, и NotifyEmergencyService типа void – метод, принимающий в качестве аргумента связующий элемент системы и экстренной службы EmergencyService, который формирует и отправляет оповещение (запрос на помощь) связующему элементу EmergencyService.

Класс EmergencyObj содержит поля ServiceNumber типа int – номер телефона экстренной службы, AdressOfice типа string – адрес здания экстренной службы, name – название экстренной службы.

Класс EmergencyService содержит поле ServiceObject типа EmergencyObj – экстренная служба, с которой устанавливается соединение через данный класс, а также метод Dispatch() типа void – метод, принимающий в качестве аргумента ServiceObject, который устанавливает соединение между пользователем и диспетчером экстренной службы.

Класс GuardedObject содержит поля: geo типа GeoCoordinate – местоположение охраняемого объекта, owner id типа int – идентификационный номер владельца охраняемого объекта, List<IDevice> - список устройств, установленных на охраняемом объекте.

Класс User содержит поля: UserID типа int – идентификационный номер пользователя – владельца охраняемого объекта, Name типа string – имя пользователя, Email типа string – email пользователя, Login типа string – логин пользователя в системе, Adress типа string – адрес проживания пользователя, NumberOfPhone типа int – номер телефона пользователя, List<GuardedObject> - список охраняемых объектов, подключенных к системе; метод GetGuardedObj возвращающий List<GuardedObj> - список охраняемых объектов.

Класс AnalysisSystem содержит поле User типа User – пользователь, данные о котором необходимо собирать и анализировать, и методы CalculateStatistic() типа void – метод расчета статистических данных, MakeAnalysis() типа void – метод составления полной статистики по всем учитываемым событиям и происшествиям за время работы системы.

# 4 Управление дефектами программного продукта и совместная работа над проектом

В ходе совместной разработки программного продукта были выявлены программные дефекты, которые, при определенных условиях, могли привести к серьезным последствиям, включая потерю доверия пользователей, негативные отзывы и, как следствие, ущерб репутации разработчиков. Для фиксации и будущего устранения этих дефектов были созданы подробные отчеты, которые можно найти в Приложении А.

Некорректная реализация программного кода, характеризующаяся наличием многочисленных синтаксических и логических дефектов, являлась первопричиной проблемы. Данные дефекты обуславливали игнорирование событий пользовательского ввода и инициировали ложный вызов службы реагирования.

Также, в ходе анализа функционирования систем «анализа и статистики», была обнаружена ошибка в логике обработки условий. При расчете показателей безопасности объекта происходило игнорирование ряда условий, что приводило к некорректному формированию отчета, демонстрирующего 100% безопасность объекта, несмотря на регистрацию событий, инициирующих срабатывание системы сигнализации.

С целью оперативного устранения обнаруженных дефектов был назначен ответственный исполнитель, предоставивший предварительную оценку сроков реализации работ в 28 календарных дней.

# 4.1. Создание тест-кейсов

С целью предотвращения возникновения непредвиденных проблем в дальнейшем, было принято решение о применении практики тестирования на основе тест-кейсов. Сотрудники составляют подробные сценарии проверок для каждой функциональности приложения, что обеспечивает своевременное обнаружение дефектов и позволяет разработчикам оперативно их устранять. На рисунке 4.1 можно увидеть примерно некоторых из тест-кейсов

Рисунок 4.1 – «Пример тест-кейсов приложения»

+коммиты с гита

# 4.2 Структура проекта на GitHub

Репозиторий на GitHub организован с использованием порядка 5 веток. Две из них выполняют ключевые функции: dev (ветка разработки) и build (ветка сборки/релиза). Остальные 3 вети закреплены за отдельными сотрудниками команды, каждый из которых отвечает за определенный круг задач. Данная практика позволяет эффективно управлять параллельной разработкой и изолировать изменения, вносимые каждым разработчиком, до момента их интеграции в основную ветку dev. На рисунке 4.2 предоставлен график изменений репозитория и его коммитов за период с 1 сентября по 15 декабря.

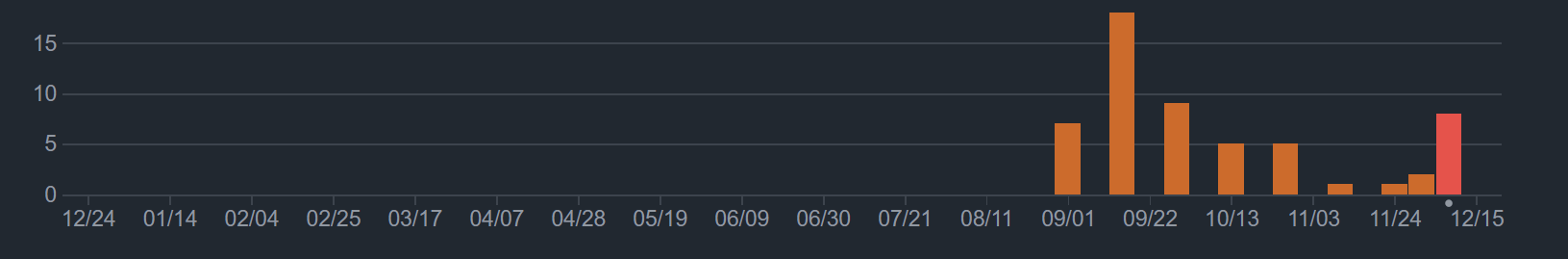


Рисунок 4.2 – График коммитов

# 4.3 Структура проекта в Visual Studio

На верхнем уровне находится папка Backend, объединяющая всю серверную логику. Backend включает в себя три основных компонента: BuisnessLogic, Devices, AnalysisSystem.

BuisnessLogic (Бизнес-логика) – эта папка содержит основную бизнес-логику приложения и разделена на следующие подкомпоненты:

Devices (Устройства) – эта папка содержит интерфейс IDevice, определяющий общий контракт для устройств, а также реализации этого интерфейса в виде классов Camera (Камера) и MotionSensor (Датчик движения). Таким образом, обеспечивается полиморфизм и возможность расширения списка поддерживаемых устройств.

AnalysisSystem (Система анализа) – проект, реализующий функциональность анализа данных, поступающих от устройств.

NotificationService (Сервис уведомлений) – проект, отвечающий за отправку уведомлений пользователям.

EmergencySysyem (Система экстренного реагирования) – проект, обеспечивающий обработку экстренных ситуаций.

DatabaseAPI (API базы данных) – этот компонент отвечает за взаимодействие с базой данных и включает в себя:

Models (Модели) – папка, содержащая классы, представляющие собой объектное отображение таблиц базы данных (ORM - Object-Relational Mapping). Эти модели используются бизнес-логикой для работы с данными.

Repositories (Репозитории) – папка, содержащая реализации шаблона "Репозиторий", предоставляющие CRUD (Create, Read, Update, Delete) операции для работы с моделями базы данных. Репозитории инкапсулируют логику доступа к данным и упрощают взаимодействие с базой данных из бизнес-логики.

DataBaseContext (Контекст базы данных) – проект, представляющий собой контекст подключения к базе данных. Он отвечает за установление соединения с базой данных и управление сессиями.

SecurityAppServer (Сервер приложения безопасности). Данный проект является точкой входа (entry point) для запуска серверной части приложения. Он отвечает за инициализацию и запуск всех необходимых сервисов и компонентов, включая бизнес-логику и API базы данных.

Архитектура проекта обеспечивает четкое разделение ответственности между различными компонентами, что способствует улучшению поддержки, масштабируемости и тестируемости приложения. Бизнес-логика отделена от деталей реализации доступа к данным, а взаимодействие с базой данных осуществляется через репозитории и контекст базы данных. Вся описанная структура Solution-файла проекта, предоставлена на рисунке 4.3.

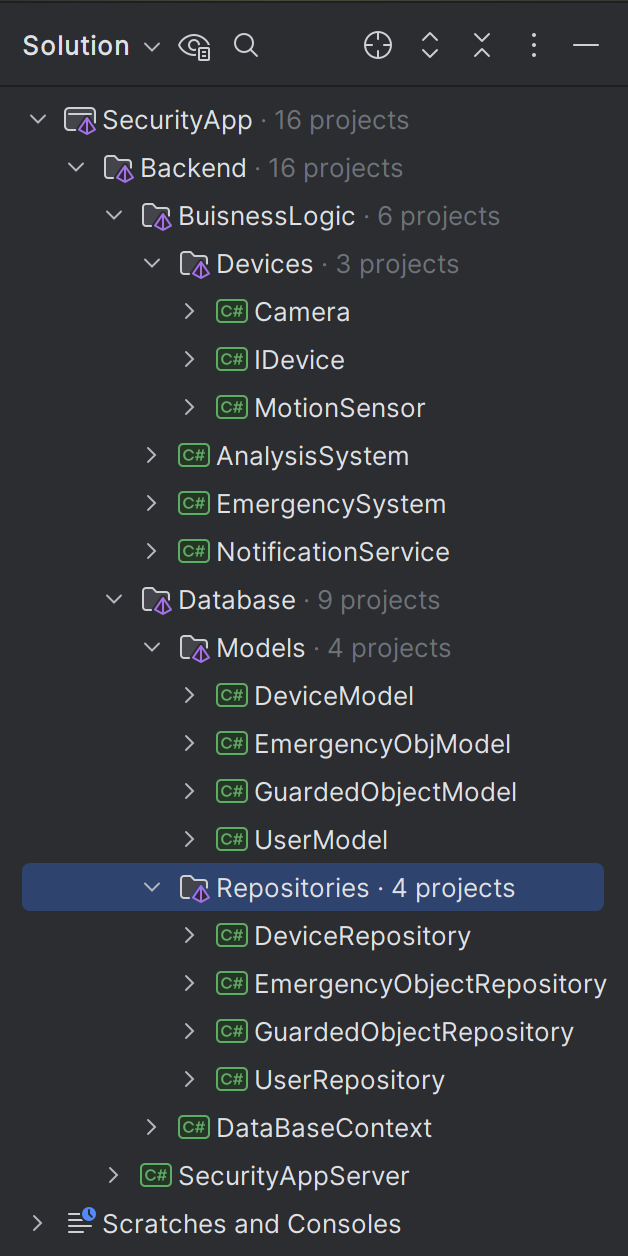


Рисунок 4.3 – Структура Solution-файла серверной части приложения

# 5 Оценка качества программного продукта

По завершении этапа разработки программного обеспечения была проведена комплексная оценка его качества командой разработчиков. Целью данной оценки являлось определение соответствия разработанного продукта заданным критериям качества и выявление потенциальных областей для улучшения.

В качестве основы для оценки были выбраны следующие критерии, охватывающие различные аспекты качества программного обеспечения:

* Функциональность. Данный критерий оценивает полноту реализованных функций, их точность, возможность взаимодействия с другими системами уровень защищенности и согласованность работы всех функций.
* Надежность. Оценивается способность программного обеспечения сохранять работоспособность в течение определенного времени, устойчивость к отказам (отказоустойчивость), возможность восстановления после сбоев и согласованность работы в различных условиях.
* Удобство применения. Данный критерий характеризует легкость освоения и использования программного обеспечения целевой аудиторией, удобство подготовки данных, понятность результатов работы и простоту внесения изменений в программную документацию.
* Сопровождаемость. Оценивается простота диагностики и устранения ошибок, внесения изменений и модификаций, а также адаптация программного обеспечения к изменяющимся требованиям.
* Рациональность. Данный критерий характеризует эффективность использования ресурсов вычислительной системы при работе программного обеспечения, соответствие затраченных ресурсов достигнутому уровню качества.
* Переносимость. Оценивается возможность переноса программного обеспечения на другие платформы или операционные системы, а также необходимые для этого усилия

Результаты оценивания программного продукта предоставлены в таблице 8.

Таблица 8 – Оценка прогаммного обеспечения по критериям

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий | Атрибут | Оценка (1-5) | Обоснование |
| Функциональность | Полнота функций | 3 | Минимальный набор необходимых функций |
|  | Интероперабельность | 2 | Ограниченная совместимость с другими системами |
|  | Точность | 3 | Данные в целом точны, но возможны небольшие погрешности |
|  | Защищенность | 2 | Низкий уровень защиты данных |
| Надежность | Отказоустойчивость | 4 | Система устойчива к большинству типичных сбоев |
|  | Восстанавливаемость | 4 | Возможность восстановления данных при необходимости |
|  | Завершенность | 5 | Все необходимые функции реализованы |
|  | Согласованность | 4 | Компоненты системы хорошо согласованы между собой |
| Удобство применения | Понимаемость | 4 | Интуитивно понятный интерфейс |
|  | Обучаемость | 4 | Не требует длительного обучения |
|  | Привлекательность | 3 | Дизайн интерфейса простой, но не запоминающийся |
|  | Согласованность | 4 | Элементы интерфейса логично расположены |
| Сопровождаемость | Анализируемость | 2 | Сложности в понимании логики работы некоторых компонентов |
|  | Изменяемость | 3 | Внесение изменений возможно, но требует значительных усилий |
|  | Стабильность | 3 | Возможны периодические сбои и нестабильное поведение |

Продолжение таблицы 8

|  |  |  |  |
| --- | --- | --- | --- |
|  | Тестируемость | 3 | Тестирование осложнено небольшим количеством тестовых данных |
|  | Согласованность | 4 | Компоненты системы хорошо согласованы между собой |
| Эффективность | Реактивность | 4 | Система работает достаточно быстро |
|  | Использование ресурсов | 4 | Эффективно использует ресурсы устройства |
|  | Согласованность | 4 | Компоненты системы хорошо согласованы между собой |
| Переносимость | Адаптируемость | 1 | Сложности в адаптации к другим платформам |
|  | Простота настройки | 3 | Конфигурация системы относительно проста |
|  | Совместимость | 1 | Совместима только с Android |
|  | Заменяемость | 3 | Замена отдельных компонентов возможна, но требует значительных усилий |
|  | Согласованность | 4 | Компоненты системы хорошо согласованы между собой |

Полученная средняя оценка качества программного продукта, равная 3.12 по пятибалльной шкале, была признана адекватной для текущего этапа разработки и удовлетворяющей всем необходимым требованиям.

# ЗАКЛЮЧЕНИЕ

Настоящая курсовая работа посвящена полному циклу разработки программного продукта, от этапа формирования требований до итоговой оценки. В ходе выполнения работы были успешно реализованы следующие этапы:

* Формирование требований. На основе проведенных ранее исследований были сформулированы первичные, функциональные и нефункциональные требования к разрабатываемому программному обеспечению. Данный этап позволил определить четкие цели и задачи проекта, а также установить критерии, по которым впоследствии оценивалось качество разработанного продукта.
* Планирование проекта. Был проведен анализ необходимого бюджета для реализации проекта, а также сформирован состав команды разработчиков. Данный этап способствовал эффективному распределению ресурсов и определению ролей участников команды.
* Разработка программного продукта. Разработка осуществлялась с применением методологии SCRUM, что обеспечило гибкость и оперативность в процессе разработки, а также возможность адаптации к изменяющимся требованиям. В результате был создан работоспособный программный продукт, отвечающий сформулированным требованиям.
* Тестирование и отладка. В рамках работы были изучены различные методы тестирования программного обеспечения, а также способы выявления и устранения программных дефектов. Проведенный анализ позволит в дальнейшем более эффективно проводить тестирование и отладку разрабатываемого программного обеспечения.
* Оценка качества программного продукта. По завершении разработки была проведена комплексная оценка качества созданного программного продукта. Полученная оценка была признана справедливой для текущего этапа разработки и послужит основой для дальнейшего совершенствования продукта.

В рамках данной курсовой работы был успешно реализован полный цикл разработки программного обеспечения, начиная от определения требований и заканчивая оценкой готового продукта. Полученные результаты демонстрируют практическое применение изученных теоретических знаний, и позволяют сделать вывод об успешном достижении поставленных целей. Дальнейшая работа над проектом будет направлена на устранение выявленных недостатков и улучшение характеристик программного обеспечения, с целью повышения его качества и функциональности.

# ПРИЛОЖЕНИЕ А

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | Краткое описание | Подробное описание | Шаги по воспроизведению | Воспроизводимость | Важность | Срочность | Симптомы | Возможность обойти | Комментарий |
| 1 | Cамопроизвольный вызов/не вызов служб (без подтверждения пользователя) | После срабатывания триггера, устанавливается необоснованная прямая связь со службой, в обход проверки условий.  **Ожидаемый результат:** Вызов служб может быть осуществлен только после подтверждения пользователя или по прошествии 1 минуты после отправки уведомления о ЧС  **Фактический результат:** Служба вызывается при отклонении пользователем данной опции  Служба не вызывается по прошествии 1 минуты после отправки уведомления или отклонении вызова  **Требование**: 2, 3 | 1. Спровоцировать триггер наблюдательного устройства 2. Ожидать реакции приложения до получения уведомления. 3. Перейти в приложение через уведомление. 4. Игнорировать любые действия и закрыть приложение. | Всегда | Высокая | Критическая | Происходит вызов службы, когда никаких действий пользователя не было совершено | Нет | Может привести к серьёзным последствиям. |
| 2 | ПО игнорирует реакцию пользователя на уведомление о ЧП | После срабатывания триггера и отправки пользователю уведомления, программа работает игнорируя класс ввод пользователя  **Ожидаемый результат:** При вызове служб программа действует на основе решения пользователя  **Фактический результат:** ПО игнорирует реакцию пользователя на уведомление о ЧП  **Требование:** 6 | 1. Спровоцировать триггер наблюдательного устройства 2. Зайти во вкладку «Устройства» 3. Перейти во вкладку «Уведомления» 4. Отклонить уведомление от приложения | Всегда | Высокая | Критическая | Происходит вызов службы не по желанию пользователя | нет | Аналогично первой проблеме, ситуация серьёзная. |
| 3 | Некорректный анализ статистики | Программа выдает недействительный результат анализа на основе собранных данных  **Ожидаемый результат:** Вывод анализа срабатываний сигнализации и пользовательской реакции. Анализ должен давать рекомендации по улучшению безопасности объекта.  **Фактический результат**: При любых обстоятельствах анализ выдает 100% сохранность объекта, игнорируя любые срабатывания наблюдательных устройств  **Требование**: 5, 7, 8 | 1. Спровоцировать триггер наблюдательного устройства (от 10 раз) 2. Открыть приложение 3. Перейти на вкладку «Анализ и статистика» 4. Изучить анализ от приложения | Всегда | Средняя | Обычная | Неверный анализ, т.к. при подсчете анализа используя формулы, которые используются приложением, от 10 срабатываний устройств, анализ должен выдавать менее 70% сохранности объекта. | Не провоцировать срабатывание устройств | Не сильно критично, можно оставить на потом. |