Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования
«Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий – РТФ

Школа бакалавриата

# ОТЧЕТ

По проекту
«Разработка личного кабинета для аналитики использования приложений компании»

по дисциплине «Проектный практикум»

Заказчик: Чекалова Э.Р.

Куратор: Кузнецов Д.С.

Студенты команды

Алексеева В.А.

Баянкин Д.А.

Еременко С.С.

Корякина А.В.

Екатеринбург, 2025

**СОДЕРЖАНИЕ**

# ВВЕДЕНИЕ

Целью проекта является создание личного кабинета для компании inpad.store, предназначенного для аналитики использования приложений и плагинов, предоставляемых этой компанией.

Задачи проекта:

– реализовать функциональность аутентификации и авторизации пользователей с ролевой моделью доступа;

– создать систему управления ключами: отображение информации о них, активация/деактивация, просмотр даты покупки, количества активаций, оставшегося срока ключа;

– разработать систему аналитики на основе данных о времени использования плагинов, запусках плагинов, количестве кликов, и экономической выгоды от их использования;

– реализовать в графиках фильтрацию отображаемых данных по пользователю и плагину;

– разработать интерфейс, соответствующий стилистике основного сайта компании.

Актуальность проекта состоит в том, что он позволит оптимизировать процесс использования и управления плагинами, повысив эффективность и прозрачность работы со статистикой использования плагинов. У компании-заказчика существует потребность в инструменте, который не только отслеживает лицензии и их использование, но и предоставляет менеджерам возможность анализировать, какие плагины наиболее часто используются, каких приносят наибольшую экономическую выгоду и в целом предоставит возможность проанализировать работу пользователей с плагинами.

На данный момент управление лицензиями и аналитика по использованию продуктов осуществляется вручную или с использованием разрозненных инструментов, что затрудняет контроль за состоянием ключей и использованием плагинов. Проблема заключается в том, что без

централизованного подхода сложно получить полную картину использования плагинов и управлять лицензиями на одном месте.

Решая эту проблему можно так же:

– упростить управление ключами и плагинами;

– повысить контроль над временем использования ключей;

– повысить эффективность управления плагинами за счет возможности регулярно и без больших время- и трудозатрат анализировать статистику использования плагинов в рамках разных временных диапазонов и групп пользователей.

Программный продукт будет использоваться в рамках компании inpad.store для внутреннего анализа и управления плагинами. Основными пользователями системы будут:

– менеджеры, которым нужно контролировать использование лицензий и анализировать эффективность работы плагинов для улучшения сервисов;

– пользователи (клиенты inpad.store), которые смогут отслеживать ключи и плагины, которые им доступны;

Решение будет использоваться в следующих областях:

– управление ключами и плагинами: для распределения и контроля доступа к продуктам и плагинам;

– аналитика использования плагинов: для сбора данных по времени использования, запускам, кликам, а также экономической эффективности использования плагинов.

По завершении проекта ожидается создание личного кабинета, который будет включать в себя такой функционал, как:

1) аутентификацию и ролевую модель доступа: пользователи смогут войти в систему в соответствии с их правами доступа (пользователь, менеджер);

2) управление лицензиями: возможность для менеджеров активировать и деактивировать лицензии, а также отслеживать их срок действия и историю изменений;

3) отслеживание плагинов: возможность для пользователей отслеживать срок действия и статус их ключей, дату покупки их и включаемые для него плагины;

4) аналитику по использованию плагинов: данные о времени использования, количестве запусков, количестве кликов и возможной экономической выгоде от использования плагинов;

5) фильтрацию отображаемых в графиках данных: возможность фильтровать статистику по пользователю или плагину и возможность указать период времени сбора статистики;

6) интуитивно понятный интерфейс дизайна: интерфейс личного кабинета будет удобным и простым в использовании а так же соблюдать стилистику inpad.store.

После завершения проекта компания inpad.store получит централизованный инструмент для управления плагинами, который упростит процесс лицензирования, повысит прозрачность и обеспечит менеджеров точной аналитикой для улучшения работы и принятия более обоснованных решений по экономической эффективности.

## 1 Информация о работе каждого участника в отдельности

### 1. 1 Алексеева Валентина Андреевна, Тимллид

В течение семестра Валентина выполняла следующие задачи:

– координирование работы команды, распределение задач между участниками в зависимости от их специализации и загруженности, а также контроль сроков выполнения поставленных задач;

– осуществление связи команды с куратором проекта, оперативное донесение информации о текущем статусе разработки, возникающих трудностях и результатах работы;

– проведение регулярных командных созвонов для обсуждения прогресса по задачам, распределения новых обязанностей, решения возникающих вопросов и синхронизации по проекту, а также участие в созвонах с куратором для отчётности и получения обратной связи;

– решение возникающих у команды вопросов организационного и технического характера, поиск решений по возникающим затруднениям в процессе разработки, помощь в расстановке приоритетов при выполнении задач;

– отслеживание выполнения задач членами команды на протяжении всего периода работы над проектом, контроль сроков, напоминание о дедлайнах, внесение корректив в планирование по мере необходимости;

– подготовка отчетных материалов, включая запись видеоотчетов, создание презентаций по проекту для контрольных точек, составление и оформление отчетов о проделанной за определённый период работе команды, а также организация финальной документации проекта.

Валентина успешно справилась с обязанностями тимлида, обеспечив слаженную командную работу, эффективную коммуникацию с куратором и своевременное выполнение всех задач, что позволило команде достичь запланированных результатов проекта.

### 1. 2 Корякина Анна Владимировна, Backend-разработчик

В течение семестра Анна выполнила следующие задачи:

– провела анализ технического задания;

– изучила структуру полученных от заказчика дампов баз данных;

– выявила необходимые для приложения сущности и связи;

– выбрала стек технологий по согласованию с заказчиком;

– реализовала Clean Architecture, разделив приложение на слои Domain, Infrastructure, Application, API;

– создала модели данных в слое Domain и определила интерфейсы для взаимодействия с базой данных;

– в слое Infrastructure настроила работу с базой данных через EF Core. Реализовала репозитории для работы с данными;

– в слое Application использовала паттерн Mediator, реализовала обработчики для получения данных для таблицы по ключу с параметрами пагинации и сортировки, получения данных о пользователях, плагинах и графиках;

– в слое API реализовала контроллеры для HTTP GET-запросов;

– тестировала работу API через Swagger и Postman;

– обнаруживала и решала ошибки;

– написала файл docker-compose.yml для контейнеризации приложения.

Этот проект стал для Анны хорошей практикой разработки web-api с применением ранее изученных подходов, таких как Clean Architecture, а также использование паттернов Mediator, Repository и AutoMapper.

Особенно полезным и интересным для неё стало взаимодействие с объемной внешней базой данных. До этого у неё не было практики работы с большими и заранее заданными структурами, что дало ей ценный опыт и

понимание, как организуется работа с большими данными и существующей базой данных.

Кроме того, Анна углубила свои знания по контейнеризации ПО с помощью Docker, а именно — написание docker-compose файла, что является полезной практикой для разработчика.

Этот проект помог ей закрепить практические навыки разработки web-api и дал опыт работы с большими данными и сложными структурами, что окажется полезным в её будущих проектах.

### 1. 3 Баянкин Даниил Антонович, Frontend-разработчик

В течение семестра Даниил выполнил следующие задачи в таких сферах работы, как:

1. разработка модуля аутентификации**:**

– реализовал компоненты: login - форма для входа пользователя, register - форма для регистрации нового пользователя, auth.service - сервис для управления аутентификацией (регистрация, логин, проверка состояния пользователя);

– реализовал валидацию форм с использованием Angular Forms (проверка email, пароля и других данных).

2. личный кабинет пользователя**:**

– создал компонент personal-info.component с формой для редактирования личных данных пользователя;

– интегрировал сервис users.service для обновления и сохранения информации о пользователе через HTTP-запросы.

3. исправление ошибок**:**

– устранил утечку памяти в auth.service (ошибка #BUG-002), что улучшило производительность;

– оптимизировал работу форм, исправив ошибку #BUG-003, что улучшило стабильность интерфейса.

4. тестирование**:**

– провел тестирование компонентов и сервисов приложения, включая функциональность аутентификации, работы с формами и обновления личной информации;

– проверил взаимодействие с сервером через API.

5. командная работа**:**

– взаимодействовал с командой для интеграции с API и решения вопросов по взаимодействию фронтенда с сервером;

– участвовал в код-ревью и помогал улучшать качество кода.

В ходе работы над проектом Даниил приобрел навыки работы с Angular, включая создание и использование компонентов, сервисов, а также работу с RxJS и Angular Forms. Эти инструменты позволили ему эффективно разрабатывать функциональные и интерактивные элементы на клиентской стороне.

Также он освоил работу с API, что включало создание и отправку HTTP-запросов, а также обработку ошибок, которые могут возникать при взаимодействии с сервером. Этот опыт дал ему уверенность в интеграции данных с внешними сервисами и в правильной обработке возникающих проблем, что значительно улучшает качество взаимодействия пользователя с приложением.

### 1. 4 Еременко София Станиславовна, Дизайнер

На прошедший семестр перед Софией была поставлена задача разработать дизайн личных кабинетов пользователя и менеджера для сайта inpad.store в соответствии с их основным сайтом.

София начала работу с дизайна личного кабинета для пользователя, в котором были созданы две страницы: личная информация о пользователе и информация о приобретенных им плагинах, а также переход на главную страницу.

Личный кабинет для менеджера включает в себя следующие страницы: личная информация о менеджере, информация о лицензионном ключе, оформленная в виде таблицы, статистика, результаты которой преобразованы в графики, информация о плагинах, также представлена в таблице, и переход на главную страницу.

Помимо основных страниц, Софией был разработан дизайн окон регистрации, авторизации и восстановления пароля.

При работе над дизайном она использовала приложение Figma. Дизайн получился минималистичным и простым, в стиле основного сайта. Основные цвета сайта: голубой, белый, серый и черный. Шрифт Open Sans, также взятый с основного сайта. В качестве референса для дизайна страницы, содержащей статистику, была использована Яндекс Метрика.

# 2 Разбор требований к программному продукту

## 2.1 Требования заказчика

Требования заказчика включали в себя такие пункты, как:

1) регистрация и авторизация пользователей:

– пользователи должны проходить регистрацию через плагин и получать учетную запись для доступа в личный кабинет;

– для управления доступом необходима возможность задавать различные роли (Менеджер и Пользователь).

2) статистика по плагинам:

– необходимо отображать данные по каждому лицензионному ключу: данные пользователя, код ключа, дата покупки, количество активаций, статус лицензии, срок действия лицензии;

– необходимо отображать аналитику по использованию плагинов: время использования плагинов, количество кликов, запуски плагинов, экономическая выгода от использования плагинов;

– возможность фильтровать статистику по пользователю, плагину, временному промежутку.

3) ролевое управление доступом:

– менеджер имеет доступ ко всем данным и статистике по плагинам и ключам;

– пользователь видит только свои данные и информацию по ключам, доступным ему.

4) экономическая выгода от использования плагинов:

– для каждого плагина должно быть представлено расчетное значение экономической выгоды.

5) интерфейс и дизайн:

– дизайн должен быть выполнен в стилистике inpad.store;

– важно создать страницы для регистрации, авторизации, отображения данных по ключам, информации о плагинах и аналитике.

## 2.2 Требования пользователей

Требования пользователей включали в себя такие пункты, как:

1) простой и понятный интерфейс:

– личный кабинет должен быть интуитивно понятным и легким в использовании;

– пользователь должен легко находить необходимую информацию.

2) отображение подробной информации в аккаунте менеджера:

– для каждого плагина должна быть доступна информация о количестве активаций, статусе лицензии, экономической выгоде;

– графики и статистика должны быть понятными и наглядными (возможность просмотра данных по периодам).

3) аналитика по использованию плагинов:

– отображение статистики по количеству запусков, времени использования плагинов, а также экономической эффективности.

4) фильтрация и сортировка данных в аналитике использования плагинов:

– возможность фильтровать информацию по пользователю или плагину, а также по времени использования плагинов.

## 3 План работы команды

Был составлен план действий для достижения поставленной цели:

1) сбор и анализ требований:

– анализ предоставленной базы данных, используемой для хранения информации о пользователях, ключах и плагинах;

– определение структуры для хранения данных о лицензиях и статистике по плагинам;

– исследование вариантов интеграции с внешними системами и API;

– определение функционала, доступного каждой роли (Менеджер и Пользователь).

2) разработка дизайна и интерфейса:

– создание дизайна в Figma, учитывая стилистику inpad.store;

– верстка страниц и интерфейса в соответствии с макетами;

– реализация графиков статистики и возможность фильтрации данных для отображения в них;

– проведение тестирования интерфейса на удобство использования.

3) разработка функционала личного кабинета:

– реализация страницы регистрации и авторизации пользователей с помощью предоставленного API;

– реализация страниц для отображения личных данных пользователя и доступных ему лицензионных ключей;

– реализация системы отображения информации по плагинам (название, статус, дата покупки, количество активаций);

– разработка страницы для менеджера с доступом ко всем данным и аналитике по использованию плагинов;

– отображение статистики по всем плагинам и ключам.

4) разработка аналитики и статистики:

– реализация графиков по времени использования плагинов;

– разработка системы отображения количества кликов по месяцам;

– внедрение расчетов экономической выгоды для каждого плагина;

– разработка логики расчета экономической эффективности использования плагинов;

– реализация фильтрации статистики по пользователям, плагинам;

– добавление возможности сортировки данных по дате.

5) тестирование и отчетные работы:

– проведение функциональных тестов для проверки корректности работы всех разделов (регистрация, авторизация, отображение данных, аналитика);

– подготовка отчета о проделанной работе.

## 4 Анализ и сопоставление существующих аналогов

Существующие решения, предоставляющий схожий с разрабатываемым продуктом, функционал:

1) ручной учет и мониторинг через Excel или Google Sheets;
2) системы управления лицензиями;
3) CRM-системы и инструменты для ведения учета;
4) другие компании, разрабатывающие плагины;
5) личные кабинеты маркетплейсов (решение представлено как референс, а не прямой аналог).

Сравнение функционала решений представлен в Таблице 1.

Талица 1 – Сравнение аналогов разрабатываемого решения

| Критерий / Аналог | **Microsoft Excel / Google Sheets** | **FutureBIM.ru** | **Ozon Seller** | **Битрикс24** | **Личный кабинет Inpad.Store** |
|---|---|---|---|---|---|
| Учет лицензий | Нет встроенной системы | Да | Нет | Да | Да (интеграция с плагинами) |
| Аналитика использования | Ручная настройка | Ограниченная | Да, но без времени использовани | Да, но без аналитики плагинов | Да (клики + время) |
| Фильтрация данных | Ручная настройка | Да | По товарам | Гибкая настройка | Да |
| Ролевая модель доступа | Нет | Ограниченная | Только для продавцов | Гибкая настройка | Для менеджеров и для пользователей |
| Расчет экономической выгоды | Ручная настройка | Нет | Расчет выручки, нет экономии | Нет | да |

Общий функционал решений, представленных выше, включает в себя:

– централизованный учет лицензий;

– ориентирование на мониторинг и учет с возможностью генерации отчетов;

– функционал, требующий значительных усилий для ввода данных и обеспечения актуальности информации.

Функционал аналогов, который будет реализован в продукте:

– функции отображения данных по лицензиям (код ключа, дата покупки, количество активаций, срок действия);

– возможности фильтрации статистики (по плагинам, пользователям);

– генерация отчетов и визуализация данных (графики, таблицы).

Важным моментом проектирования продукта является определение уникальных черт разрабатываемого решения, а именно:

1) решение будет отличаться тем, что оно будет встроено в сайт заказчика, следовательно ни пользователям, ни самому заказчику не нужно будет переходить на сторонние сервисы;

2) решение будет предоставлять только тот функционал, который необходим заказчику, что позволит избежать перегруза функционалом и гарантирует, что все, что необходимо заказчику, будет в функционале решения.

Разработанное решение будет отличаться от существующих подходов своей простотой, удобством и гибкостью, а также точностью и качеством аналитики. Оно обеспечит более высокую степень контроля и экономии для компании при меньших затратах на внедрение и поддержку.

## 5 Планирование работы

В рамках реализации проекта была проведена предварительная работа по планированию деятельности и распределению обязанностей между участниками команды. Это позволило выстроить рабочий процесс, обеспечить контроль за выполнением задач и своевременно реагировать на возникающие трудности.

На старте проекта был сформирован бэклог задач, который включал все этапы разработки программного продукта: от настройки окружения и верстки интерфейсов до работы с базой данных, построения аналитики и подготовки документации для сдачи проекта.

Команда работала по недельным итерациям. В начале каждой недели задачи актуализировались, назначались ответственные, а также определялись приоритетные направления работы в зависимости от текущего состояния проекта и замечаний со стороны заказчика.

Распределение обязанностей в команде:

– Даниил — фронтенд-разработчик, отвечал за разработку пользовательских интерфейсов личного кабинета, регистрацию и авторизацию, а также за внедрение и настройку Docker-контейнеров;

– Анна — backend-разработчик и аналитик, занималась выгрузкой и обработкой данных из базы данных, подготовкой данных для графиков, реализацией авторизации и разработкой серверной части;

– София —дизайнер, отвечала за создание аналитических графиков, их оформление, подписи, настройку визуальных параметров, а также внесение изменений в дизайн макетов;

– Валентина — координатор проекта и аналитик, осуществляла коммуникацию с куратором, уточнение требований, сбор информации о статистике, отслеживание прогресса команды и координацию подготовки отчётных материалов.

Работа команды велась по заранее составленному плану-графику, в котором задачи распределялись по датам. Основные этапы работы:

– подготовка окружения, формулирование требований к продукту и необходимого функционала;

– разработка пользовательского интерфейса;

– реализация авторизации и личного кабинета;

– работа с базой данных, выгрузка и анализ информации;

– построение и верстка графиков;

– подготовка проектной документации, презентации и видеодоклада.

Задачи распределялись с учетом специализации участников и корректировались по мере необходимости. Еженедельно проводились командные встречи, на которых подводились итоги выполненных работ и планировались следующие шаги. Это позволяло участникам команды быть в курсе хода разработки остальных членов команды. Так же еженедельные созвоны помогли избежать перерывов в разработке и поддерживать её стабильный темп.

Коммуникация с участниками команды проводилась в групповом чате в мессенджере Telegram, в нем же был отдельный чат с куратором, в нем и по средствам Yandex Telemost происходило основное взаимодействие с куратором: при возникновении простых вопросов обращение к куратору происходило в чате с ним, если возникали более крупные вопросы, то организовывался звонок в Yandex Telemost. Задачи, установленные на созвонах отдельно прописывались в чате команды и отдельно в TeamProject.

Такой подход к планированию и распределению задач позволил команде эффективно организовать рабочий процесс, своевременно подготовить программный продукт, сопроводительные материалы и выйти на финальную предзащиту в срок.

## 6 Обзор архитектуры backend-части продукта

Архитектура backend-части приложения реализована по принципам Clean Architecture, что позволяет разделить систему на независимые слои, каждый из которых выполняет свою роль и не зависит от реализации других слоёв. Это способствует улучшению модульности, тестируемости и удобству поддержки кода. Схема архитектуры backend-части представлена на Рисунке 1.

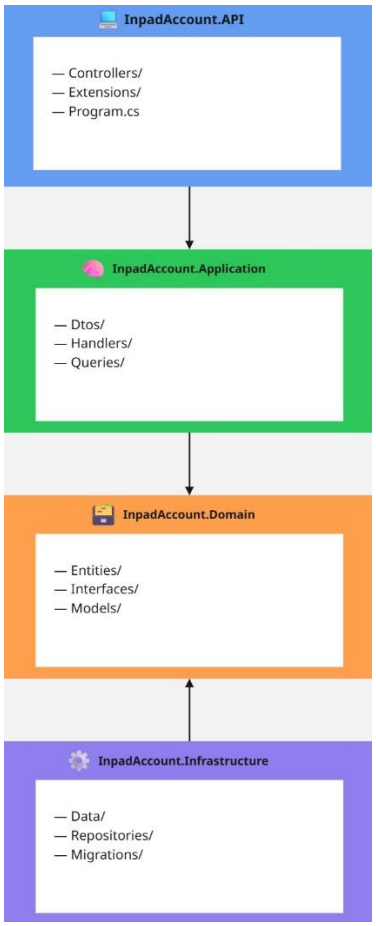


Рисунок 1 – Схема архитекутры backend-части приложения

Архитектура включает в себя четыре основных слоя.

### 6.1 Domain Layer (Слой домена)

Слой Domain является основой архитектуры и не имеет зависимостей от других слоёв. Он содержит все основные объекты, которые представляют

бизнес-логику системы. Этот слой является независимым от того, как будет реализован доступ к данным, интерфейсы взаимодействия с клиентами или другие технологии.

Содержит:

– сущности базы данных — объекты, которые напрямую отражают основные концепты приложения (например, Пользователь, Лицензия, Плагин);

– бизнес-модели — классы и структуры, которые инкапсулируют бизнес-логику и правила работы с доменными сущностями;

– абстракции доступа к данным — интерфейсы репозиториев, которые задают контракты для взаимодействия с базой данных, не привязываясь к конкретным технологиям хранения данных.

### 6.2 Infrastructure Layer (Инфраструктурный слой)

Слой Infrastructure отвечает за взаимодействие с внешними ресурсами, такими как база данных, внешние сервисы, файловая система и другие инфраструктурные компоненты. Этот слой зависит от Domain, но не должен нарушать принципы его независимости.

Содержит:

– настройку Entity Framework Core (EF Core) — настройки подключения к базе данных, конфигурации контекста данных и маппинг сущностей;

– реализации репозиториев — конкретные реализации интерфейсов, предоставляющих доступ к данным через EF Core, отвечающие за работу с базой данных;

– миграции баз данных — инструменты для управления схемой базы данных, позволяющие создавать и обновлять таблицы, индексы и другие объекты базы данных.

### 6.3 Application Layer (Слой приложения)

Слой Application занимается обработкой запросов и управлением бизнес-логикой, необходимой для выполнения операций в системе. Он зависит как от Domain, так и от Infrastructure, но не имеет прямых зависимостей от слоя API или других внешних компонентов.

Содержит:

– бизнес-логику — алгоритмы, которые выполняются в ответ на запросы пользователя или системы, например, расчёт статистики, обработка данных или выполнение транзакций;

– DTO (Data Transfer Objects) — объекты, которые используются для передачи данных между слоями приложения (например, данные пользователя или статистика по активациям), при этом данные от сущностей базы данных не передаются напрямую;

– обработчики команд (Command Handlers) — классы, реализующие паттерн Mediator, отвечающие за обработку команд от клиентской части (например, создание пользователя или обновление данных).

### 6.4 API Layer (Слой API)

Слой API служит точкой взаимодействия с клиентской частью приложения. Этот слой зависит от Application и предоставляет конечные точки для работы с системой через HTTP-запросы, такие как RESTful API. Он отвечает за получение запросов от клиента, передачу данных в слои приложения и возвращение результатов.

Содержит:

– контроллеры — классы, обрабатывающие HTTP-запросы и передающие их в слой Application для дальнейшей обработки, контроллеры обеспечивают интерфейс для работы с функционалом системы;

– точки входа в приложение — маршруты API, которые могут включать авторизацию, управление пользователями, выгрузку данных и другие функции;

– Dockerfile — файл для контейнеризации приложения, который позволяет создать и запустить приложение в Docker-контейнерах, обеспечивая удобство развертывания и масштабирования системы.

### 6.5 Связи между слоями

Архитектура построена таким образом, чтобы каждый слой взаимодействовал только с теми слоями, которые находятся ниже него. Это помогает минимизировать зависимости и упростить тестирование и модификацию отдельных частей системы. Взаимодействие между слоями происходит через абстракции (например, интерфейсы репозиториев в слое Domain), что позволяет легко подменять реализации без влияния на остальные части системы.

Связи между слоями:

– API зависит от Application, предоставляя точки входа в приложение;

– Application зависит от Infrastructure для доступа к данным и выполнения операций;

– Infrastructure взаимодействует с внешними системами (например, базой данных), а Domain остаётся независимым и чистым от внешних зависимостей.

Эта архитектура гарантирует масштабируемость, тестируемость и удобство в развитии системы, позволяя легко внедрять новые функции без необходимости переписывать существующий код.

## 7 Обзор архитектуры frontend-части продукта

Программный продукт представляет собой одностраничное web-приложение, разработанное с использованием Angular, выполненное в модульной архитектуре. Такая структура обеспечивает удобное разделение функциональных областей приложения, повышает читаемость кода, упрощает тестирование и последующее сопровождение проекта. Архитектура приложения построена на принципах компонентного подхода и строгого разделения ответственности между различными модулями и слоями.

Программный продукт состоит из нескольких основных модулей, каждый из которых выполняет строго определённые функции в рамках общей архитектуры:

1) корневой модуль (app) — основной модуль, в который импортируются все остальные функциональные модули, отвечает за общую инициализацию приложения и настройку маршрутизации;

2) модуль account — специализированный модуль, обеспечивающий работу с учетными записями пользователей. В его состав входят подмодули:

– personal-info — предназначен для отображения и редактирования персональной информации пользователя;

– forgot-password — реализует интерфейс и логику восстановления пароля;

– login — отвечает за форму входа в систему;

– register — реализует регистрацию новых пользователей.

3) модуль plugins — включает компоненты и сервисы для работы с подключаемыми плагинами, предоставляя пользователю возможность просматривать список доступных плагинов и управлять ими;

4) модуль shared — модуль общих компонентов, моделей и сервисов, используемых во всех остальных модулях приложения, в его состав входят:

– header.component.ts — компонент, отвечающий за отображение шапки сайта, включая информацию о текущем состоянии аутентификации пользователя;

– модели данных (models), такие как: activation.models — для хранения информации об активации учетных записей, user.models — для хранения информации о пользователях, их параметрах и статусах;

– сервисы (services): activation.service — выполняет активацию учетных записей, users.service.ts — реализует взаимодействие с серверной частью для получения, обновления и удаления данных пользователей.

Связи между компонентами приложения организованы с помощью сервисов, реализующих бизнес-логику и взаимодействующих с серверной частью системы посредством HTTP-запросов. Это обеспечивает слабую связанность компонентов, что соответствует принципам современной архитектуры SPA-приложений.

– аутентификация и регистрация: компоненты login.component и register.component используют auth.service для отправки запросов на сервер и получения токенов аутентификации, после успешной авторизации данные пользователя передаются в header.component, обеспечивая отображение информации о текущем пользователе на всех страницах;

– управление пользователями: сервис users.service используется в personal-info.component для получения и обновления информации о пользователе, а также в других компонентах модуля account; кроме того, activation.service участвует в процессе активации учетных записей после их регистрации;

– работа с плагинами: компонент plugins.component осуществляет загрузку данных о подключаемых плагинах через plugins.service; при этом возможна зависимость от auth.service, если доступ к данным плагинов требует авторизации пользователя;

– общие компоненты: header.component постоянно отслеживает состояние аутентификации пользователя через auth.service и отображает соответствующую информацию в шапке приложения на всех страницах.

Таким образом, модульная структура программного продукта обеспечивает высокую гибкость архитектуры, возможность масштабирования и добавления новых функциональных возможностей без необходимости внесения изменений в уже существующий функционал. Благодаря строгому разграничению ответственности между модулями и слоями обеспечивается удобство в сопровождении и поддержке программного кода, а также возможность независимого тестирования отдельных компонентов приложения.

## 8 Описание процесса разработки

Работа над проектом велась по гибкой методологии Agile:

– проект был разбит на спринты продолжительностью две недели;

– в начале каждого спринта команда формировала бэклог с задачами и оценивала временные затраты на их реализацию;

– каждую неделю проводились командные созвоны для синхронизации, обсуждения текущих проблем, вопросов по реализации и адаптации плана;

– вся работа фиксировалась и контролировалась в системе управления проектами teamproject.urfu.ru.

Процесс разработки:

1) анализ требований:

– изучено техническое задание от заказчика;

– получены дампы баз данных для анализа;

– локально развернуты дампы, изучена структура таблиц, связи между ними;

– проведена декомпозиция на подзадачи.

2) выбор технологий:

– по требованию заказчика проект реализован на .NET 8, EF Core;

– для организации архитектуры выбран подход Clean Architecture;

– использованы паттерны Repository, Mediator и библиотека AutoMapper для удобной работы с данными.

3) реализация слоёв:

– Domain: описание доменных сущностей, интерфейсы репозиториев;

– Infrastructure: настройка EF Core, подключение БД, миграции, реализация репозиториев;

– Application: подключение MediatR, написание обработчиков команд и запросов, бизнес-логика;

– API: создание контроллеров и точек взаимодействия с клиентской частью.

4) разработка клиентской части:

– создание компонентов, сервисов и модулей с использованием Angular;

– реализация сервисов для взаимодействия с REST API, включая настройку HTTP-запросов, обработку ошибок и передачу данных между компонентами;

– интеграция фронтенда с серверной частью приложения через REST API.

5) поэтапное расширение функционала: по мере выполнения задач добавлялись новые сущности, обработчики и репозитории, что обеспечило масштабируемость системы.

Тестирование:

– API тестировался средствами SwaggerUI и Postman;

– для демонстрации работы был подготовлен простой тестовый фронтенд на Blazor;

– организована тестовая среда на Docker Compose с двумя базами данных: дамп заказчика и разработчика;

– проведены E2E-тесты (Cypress/Protractor);

– использовано ручное тестирование критических сценариев.

Тесты охватили:

– получение и отображение информации по ключу;

– пагинацию и сортировку данных;

– получение информации о пользователях;

– вывод данных о плагинах;

– получение данных для построения графиков.

Основные проблемы, возникшие при разработке, и их решение:

– проблема: дамп базы данных имел неверную кодировку, решение: произведено преобразование дампа в формат UTF-8;

– проблема: отсутствие миграций мешало получить доступ к данным, решение: созданы и применены необходимые миграции;

– проблема: контроллеры были перегружены бизнес-логикой, решение: внедрён паттерн Mediator, что позволило распределить логику между командами и обработчиками, упростив поддержку и развитие проекта.

# ЗАКЛЮЧЕНИЕ

Оценка качества программного продукта на основе результатов тестирования, выявление дефектов, ошибок и их влияния на работоспособность продукта

В процессе разработки программного обеспечения была реализована система по принципам Clean Architecture с разделением на слои Domain, Infrastructure, Application и API, что позволило обеспечить модульность, структурированность и удобство сопровождения кода.

На основании проведённых этапов тестирования и проверки работоспособности основных функций программного продукта были получены следующие результаты:

– разделение приложения на слои повысило читаемость и сопровождаемость кода, упростило внедрение новых функций и тестирование отдельных компонентов без влияния на остальные части системы, что положительно сказалось на стабильности работы приложения при доработках;

– интеграция Swagger позволила ускорить процесс тестирования API, обеспечив удобный интерфейс для отправки HTTP-запросов, визуализации полученных данных и своевременного выявления ошибок в передаваемых данных и логике контроллеров;

– внедрение Docker и docker-compose: контейнеризация приложения упростила развертывание и тестирование на различных машинах разработчиков. Это снизило количество ошибок, связанных с различиями в средах выполнения, и ускорило интеграцию клиентской части с серверной;

– применение паттерна Mediator позволило разгрузить контроллеры от бизнес-логики, вынеся обработку команд и запросов в отдельные обработчики, что повысило читаемость кода, облегчило сопровождение и внесение изменений;

– разработка клиентской части с использованием Angular обеспечила удобный и адаптивный интерфейс для пользователей, интеграцию с серверной

частью через REST API и упрощение процессов аутентификации, регистрации и управления плагинами, реализация компонентов и сервисов повысила гибкость и расширяемость приложения, а также улучшила взаимодействие между клиентом и сервером.

В процессе тестирования были зафиксированы следующие дефекты:

– проблема с кодировкой при разворачивании базы данных привела к некорректному отображению и чтению данных, что могло нарушить целостность передаваемой информации, данный дефект был устранён путем преобразования дампов в формат UTF-8;

– отсутствие миграций на раннем этапе препятствовало работе с базой данных, так как невозможно было получить или изменить данные, проблема решена созданием и применением необходимых миграций;

– перегрузка контроллеров бизнес-логикой вызывала усложнение архитектуры API, снижала скорость разработки и увеличивала риск ошибок, Mediator позволило устранить этот дефект.

Функционал, который будет добавлен к защите:

– авторизация и аутентификация пользователей – задержки произошли из-за задержки данных от заказчика;

– вывод графиков со статистикой использования плагинов.

Степень готовности проекта на момент сдачи отчета:

– backend: 80%;

– frontend: 62%.

По оценке темпов разработки, проект будет закончен в течение недели после сдачи отчета.

На основании полученных результатов можно сделать вывод, что программный продукт обладает необходимым уровнем качества, соответствует предъявленным требованиям, но не готов к эксплуатации в связи с доработкой функционала. Выявленные в процессе тестирования дефекты были своевременно устранены, что положительно сказалось на стабильности и надёжности системы. Применённые архитектурные решения

обеспечивают хорошую масштабируемость, модульность и удобство дальнейшего развития продукта.

Таким образом, программное обеспечение готово к финальной доработке и к передаче заказчику для дальнейшего использования и внедрения, также решение предусматривает возможность оперативного добавления нового функционала без риска нарушения стабильности текущей системы.

Для дальнейшего развития продукта можно рассмотреть следующие направления:

1) оптимизация производительности и масштабируемости: внедрение кэширования на уровне базы данных с использованием Redis или Memcached для ускорения доступа к часто используемым данным, также можно рассмотреть использование микросервисной архитектуры и контейнеризацию через Kubernetes для улучшения масштабируемости и гибкости системы в условиях роста нагрузки;

2) улучшение безопасности: добавление двухфакторной аутентификации и регулярные обновления для обеспечения защиты от уязвимостей, можно проводить аудит безопасности кода и базы данных с целью выявления потенциальных угроз и уязвимостей, чтобы минимизировать риски при использовании приложения в продакшн-среде;

3) расширение функционала и UX/UI улучшения: внедрение новых возможностей, таких как поддержка многозадачности или интеграция с внешними сервисами, проведение A/B-тестирования и регулярное улучшение пользовательского интерфейса на основе обратной связи с пользователями для повышения удобства и эффективности взаимодействия с приложением.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Архитектура программного продукта [Электронный ресурс] / Разработка на .NET 8 с использованием EF Core и Clean Architecture. – Москва, 2023. – URL: https://www.dotnetfoundation.org/ (дата обращения: 20.05.2025).

2. Интеграция с Angular: использование компонент и сервисов для взаимодействия с бэкендом [Электронный ресурс] / Официальная документация Angular. – URL: https://angular.io/docs (дата обращения: 20.05.2025).

3. Docker и контейнеризация: Внедрение Docker в процессы разработки [Электронный ресурс] / Docker Inc. – Сан-Франциско, 2023. – URL: https://www.docker.com/resources/what-container (дата обращения: 20.05.2025).

4. Swagger UI для тестирования API [Электронный ресурс] / Swagger. – URL: https://swagger.io/tools/swagger-ui/ (дата обращения: 20.05.2025).

5. Паттерн Mediator в архитектуре приложений [Электронный ресурс] / Описание паттерна Mediator. – URL: https://refactoring.guru/design-patterns/mediator (дата обращения: 20.05.2025).

6. Официальная документация Blazor: создание фронтенда с использованием Blazor [Электронный ресурс] / Microsoft. – URL: https://docs.microsoft.com/en-us/aspnet/core/blazor/ (дата обращения: 20.05.2025).

7. Паттерн Repository в архитектуре программного обеспечения [Электронный ресурс] / Мартин Фаулер. – URL: https://martinfowler.com/eaaCatalog/repository.html (дата обращения: 20.05.2025).

8. EF Core: Применение Entity Framework для работы с базами данных [Электронный ресурс] / Microsoft. – URL: https://docs.microsoft.com/en-us/ef/core/ (дата обращения: 20.05.2025).

9. Применение паттерна Mediator в современных веб-приложениях [Электронный ресурс] / Описание применения паттерна Mediator в современных системах. – URL: https://dilfuruzhassan.medium.com/mediator-pattern-in-

net-architecture-23db7db6a0d5 (дата обращения: 20.05.2025).