Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

|  |  |
| --- | --- |
|  | Номер зачетной книжки 15100084 |
|  | Производственная практика зачтена с оценкой  \_\_\_\_\_ (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)  (цифрой) (прописью) |
|  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись руководителя практики от БГУИР)  \_\_\_\_.\_\_\_\_\_.2024 |

**ОТЧЕТ**

**по производственной практике**

Место прохождения практики: ООО “Фабрика инноваций и решений”

Сроки прохождения практики: с 10.06.2024 по 07.07.2024

|  |  |  |
| --- | --- | --- |
| Руководитель практики от предприятия:  \_\_\_\_\_\_\_\_\_\_\_ Д.С. Кисляк  (подпись руководителя)  М.П. |  | Студент группы 151004  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.М. Прокопчук  (подпись студента)  Руководитель практики от БГУИР  Можей Н.П. – кандидат физико-математических наук, доцент. |

Минск 2024

СОДЕРЖАНИЕ

[Введение 3](#_Toc175499331)

[1 ОБЩАЯ ХАРАКТЕРИСТИКА ПРЕДПРИЯТИЯ 4](#_Toc175499332)

[1.1 Структура предприятия 4](#_Toc175499333)

[1.2 Программное обеспечение, используемое на предприятии 6](#_Toc175499334)

[1.3 Правовые, нормативные, социально-экономические и организационные вопросы охраны труда 7](#_Toc175499335)

[2 ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ 10](#_Toc175499336)

[2.1 Общее задание 10](#_Toc175499337)

[2.2 Индивидуальное задание 10](#_Toc175499338)

[3 ВЫПОЛНЕНИЕ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ 12](#_Toc175499339)

[3.1 Используемые технологии 12](#_Toc175499340)

[3.2 Описание вспомогательных инструментов разработки 15](#_Toc175499341)

[4 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ 17](#_Toc175499342)

[5 ТЕСТИРОВАНИЕ 18](#_Toc175499343)

[ЗАКЛЮЧЕНИЕ 20](#_Toc175499344)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 21](#_Toc175499345)

[ПРИЛОЖЕНИЕ А 22](#_Toc175499346)

# Введение

Производственная практика – это один из важнейших этапов подготовки специалистов, в ходе которого происходит первое знакомство студента с производственным процессом. Она проводится в условиях, приближенных к будущей профессиональной деятельности, дополняя теоретическое обучение практическими занятиями и умениями, которые понадобятся в будущем.

Целью практики является выработка профессиональных навыков в области информатики и разработки программного обеспечения.

В качестве места прохождения практики была выбрана компания ООО «Фабрика инноваций и решений».

Прохождение практики ставит следующие задачи:

* ознакомится со структурой предприятия;
* изучить нормативную документацию, должностные обязанности, ППП, используемые на предприятии;
* изучить заданный технологический процесс или все бизнес-процессы на предприятии;
* изучить конкретный (в соответствии с индивидуальным заданием) программный продукт;
* сформировать список требований в рамках поставленной по практике технической задачи;
* собрать материал для формирования технической задачи, которую планируется решать в дипломном проекте.

# Общая характеристика предприятия

## Структура предприятия

Innowise Group - одна из наиболее опытных белорусских компаний в области полного цикла разработки, внедрения и поддержки программного

обеспечения, бизнес-анализа, дизайна, тестирования и управления проектами.

Компания была основана в 2007 году. За 15 лет реализовала более 500

проектов в разных отраслях для клиентов из США, Канады, Великобритании,

Германии, Франции, Италии, Японии и других стран.

Компания работает более чем с 20 бизнес-областями, включая создание

решений с использованием Artificial Intelligence, Machine Learning, Augmented

& Virtual Reality.

Сейчас Innowise Group объединяет более 800 сотрудников в 8 городах:

Минск, Витебск, Гомель, Могилев, Полоцк, Брест, Киев и Варшава. Компания

продолжает расширяться и приглашает в свою команду как опытных

профессионалов, так и молодых специалистов.

Сотрудники Innowise Group работают над масштабными проектами для

международных компаний, а также создают собственные инновационные

продукты. Например, проект VOKA.IO, основанный на применении Mixed

Reality технологий в области хирургии победил в категории всемирного

конкурса инновационных проектов World Summit Awards 2019 и получил

международное признание от врачей всего мира.

В Innowise Group ценят своих сотрудников и знают, что успех –

результат усилий всех членов команды. Вместе компания делает мир более

комфортным и совершенным, трансформирует цифровую реальность и мечты

в готовые решения.

На базе технологий Java, .NET, PHP, Unity, React, Python, Angular, Vue, Solidity,React Native специалистами компании было успешно реализовано более 250 проектов в сфере электронной коммерции, медицины, автомобиле строения, обучения, транспорта и логистики, моды и красоты, AR/VR, AI/ML и др. областях.

Структуру предприятия можно увидеть на рисунке 1.1.

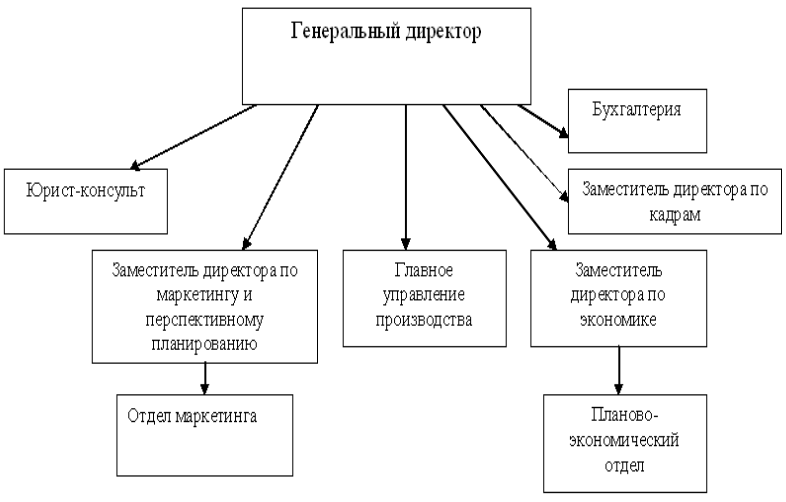


Рисунок 1.1 – организационная структура управления Innowise Group

Во главе предприятия находится генеральный директор. Он организует

всю работу предприятия и несет полную ответственность за результаты

производственно-хозяйственной деятельности. Директор представляет

предприятие во всех учреждениях и организациях, заключает договора, издает

приказы по предприятию, открывает в банках счета предприятия и выполняет

целый ряд других функций.

В непосредственном подчинении директора предприятия находятся три

заместителя: по маркетингу, по экономике и по кадрам, а также главный

бухгалтер и юрист-консультант. Компания специализируется на:

* разработка, тестирование, сопровождение и поддержка заказного

программного обеспечения и бизнес-приложения;

* интеграция приложений;
* миграция приложений;
* создание выделенных центров разработки, центров тестирования и

контроля качества программного обеспечения.

Офис компании можно увидеть на рисунке 1.2.



Рисунок 1.2 – главный офис компании Innowise Group

(ООО «Фабрика инноваций и решений»)

## Программное обеспечение, используемое на предприятии

Microsoft Office - офисный пакет приложений, созданных Microsoft для операционных систем Windows, Windows Phone, Android, macOS, iOS. В состав этого удобного пакета инструментов входит программное обеспечение для удобной корпоративной работы с различными типами документов: текстами, электронными таблицами, базами данных.

Microsoft Office является сервером и его функции могут использоваться другими приложениями, а также самими приложениями Microsoft Office. Поддерживает скрипты и макросы, написанные на VBA.

Microsoft Word - текстовый процессор. Доступен под Windows, Android и macOS. Позволяет подготавливать документы различной сложности. Поддерживает OLE, подключаемые модули сторонних разработчиков, шаблоны и многое другое. Основным форматом в последней версии является позиционируемый как открытый Microsoft Office Open XML, который представляет собой ZIP-архив, содержащий текст в виде XML, а также всю необходимую графику. Наиболее распространенным остается двоичный формат файлов Microsoft Word с расширением DOC. Продукт занимает ведущее положение на рынке текстовых процессоров, и его форматы используются как стандарт в документообороте большинства

предприятий. Word также доступен в некоторых редакциях Microsoft Works.

Microsoft Excel - табличный процессор. Поддерживает все необходимые функции для создания электронных таблиц любой сложности. Занимает ведущее положение на рынке.

Эта программа нужна всякому, кто имеет дело с данными, представленными в виде таблиц. Именно в таком виде можно представить большую часть информации о нашей повседневной деятельности: адресные книги, картотека видеокассет, учет домашних расходов.

В таблицах могут находиться данные самого разного формата, но в целом, конечно же, пакет Excel ориентирован на обработку числовых данных. Самый простой пример - суммирование по столбцам и строкам таблицы. Ненамного сложнее вычислить значение заданных самим пользователем функций. Одна из наиболее привлекательных возможностей пакета – быстрое и наглядное представление данных в виде разнообразных графиков, диаграмм, карт.

Pycharm Professional - это, прежде всего, среда разработки для Python и других языков. С этим языком она дружит больше всего, отлично его понимает и помогает в написании разработчику. Но это не значит, что все заканчивается на нем и собственном языке Python. Не менее важным является поддержка из коробки таких передовых технологий, как HTML, CSS и др.. Не забывают в компании и о трендах, ведь для большинства веб- языков созданы свои отдельные среды. Среди них:

* phpStorm - среда разработки на PHP, которая активно

используется;

* rubyMine - среда для взаимодействия с языком Ruby;
* Intellij IDEA для Java;
* appCode для Objective-C.

## Правовые, нормативные, социально-экономические и организационные вопросы охраны труда

В каждой организации и должна быть разработана, а также соблюдаться система управления охраной труда – это регламентированная законодательством совокупность организационных, технических, гигиенических, лечебно-профилактических и социально-экономических мероприятий, методов и средств управления, направленных на организацию деятельности по обеспечению безопасности, сохранению жизни, здоровья и работоспособности человека в процессе труда, включающих в себя выявление производственных опасностей, оценку рисков гибели и травмирования работников, разработку и реализацию эффективных мер по их снижению, а также осуществление постоянного мониторинга и измерение результативности в области охраны труда.

Основными нормативными правовыми актами, регулирующими вопросы охраны труда, являются:

* трудовой кодекс Республики Беларусь;
* закон Республики Беларусь «Об охране труда»;
* директива Президента Республики Беларусь от 11 марта 2004 г. № 1 «О мерах по укреплению общественной безопасности и дисциплины».

В организации проводятся следующие виды инструктажей:

* вводный инструктаж. Проводится с сотрудниками, принятыми на работу;
* первичный инструктаж на рабочем месте. Проводится непосредственно перед началом самостоятельной работы нового сотрудника;
* повторный инструктаж на рабочем месте. Проводится каждые 3 месяца для всех, кто проходил первичный инструктаж;
* внеплановый инструктаж. Проводится при приобретении нового оборудования, а также при несчастных случаях, вызванных нарушением работниками требований охраны труда;
* целевой инструктаж. Проводится с целью выполнения разовых работ или мероприятий, направленных на устранение последствий чрезвычайных ситуаций.

Расследование и учет несчастных случаев в ООО «Фабрика инноваций и решений» осуществляется в соответствии с «Правилами расследования и учета несчастных случаев на производстве и профессиональных заболеваний», утвержденных постановлением Совета Министров Республики Беларусь 15.01.2004 №30.

При поступлении на работу или переводе на другую работу (даже временно), не допускается приступать к работе без получения инструктажа по охране труда.

Каждый работающий должен знать и строго соблюдать Правила внутреннего трудового распорядка, трудовую и производственную дисциплину.

Категорически запрещается нахождение в состоянии алкогольного опьянения либо в состоянии, вызванном употреблением наркотических средств, психотропных или токсичных веществ, а также распитие спиртных напитков, употребление наркотических средств, психотропных или токсических веществ на рабочем месте или в рабочее время.

Запрещается самовольно уходить с работы, выполнять посторонние работы, не связанные с производством.

Каждый работник должен выполнять только ту работу, которая ему поручена своим непосредственным руководителем

# ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

## Общее задание

Руководителем практики университета было выдано следующее индивидуальное задание:

* ознакомиться со структурой предприятия;
* описать бизнес-процессы, происходящие на предприятии;
* сформировать список требований в рамках поставленной по практике технической задачи;
* оформить в соответствии со стандартом техническое задание поставленной инженерной задачи, выбрать технологии разработки, разработать алгоритмы, создать интерфейс, описать работу индивидуального проекта;
* выполнить индивидуальное задание;
* собрать материал для формирования технической задачи, которую планируется решать в дипломном проекте;
* оформить и написать отчёт о выполнении задания по производственной практике;
* собрать материал для формирования технической задачи, которую планируется решать в дипломном проекте.

## Индивидуальное задание

Руководителем практики на предприятии было выдано следующее индивидуальное задание: разработка веб-приложения “Social Hub”. Общей целью данного веб-приложения будет создание удобного пространства для создания постов, страниц, оставления лайков и комментариев. Также необходимо реализовать механизмы регистрации и входа для пользователей, а еще необходимо добавить механизм аутентификации, позволяющий использовать токен для авторизации вместо ввода логина и пароля каждый раз. Задание было разделено на части:

* разработка структуры и архитектуры;
* разработка базы данных и интеграция её с приложениемм;
* реализация сервисов с бизнес-логикой приложения;
* реализация регистрации новых пользователей и аутентификация пользователей через JWT;
* создание REST эндпоинтов для асинхронной обработки пользовательских запросов.

Также была поставлена задача разработки фронтенд-части веб-приложения с использованием современного фреймворка React и языка программирования TypeScript для взаимодействия с созданной бэкенд-частью на Python. Проект предполагает создание пользовательского интерфейса для управления пользователями, постами, страницами, подписчиками, а также для механизма аутентификации пользователей.

# ВЫПОЛНЕНИЕ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ

По прибытии на место прохождения практики я был ознакомлен со структурой предприятия, организацией труда, техникой безопасности и основными информационными потоками, документооборотом.

Подготовка специалистов в компанию осуществляется через прохождение стажировки, за время которой претендент изучает технологии, выполняет практические задания, получает тему проекта и выполняет его. Изучение технологий заканчивается контрольной проверкой, где проводится опрос о используемых технологиях, реализованных функциях и изученному материалу.

## Используемые технологии

Python - это объектно-ориентированный язык программирования, разработанный Гвидо Ван Россумом в конце 1980-х. Данный язык обрел большую популярность из-за своей универсальности и простоты написания кода на нём. В настоящее время язык активно развивается, новые версии выпускаются. Язык используется во множестве сфер: машинное обучение, искусственный интеллект, веб-разработка, написание скриптов, разработка ботов, анализ данных.

Python является очень универсальным и гибким языком программирования, предоставляя разработчикам множество способов достижения желаемых целей. Язык, следуя по большей части функциональной парадигме, также поддерживает ряд других парадигм, включая объектно-ориентированное и обобщенное программирование. В рамках реализации индивидуального задания будут использоваться 2 веб-фреймворка Python: Django(DRF) и FastAPI.

**3.1.1** Язык Python

Python является объектно-ориентированным языком программирования. Основной особенностью Python можно назвать динамическую типизацию. Это означает, что типы данных будут определены в момент выполнения кода интерпретатором. Это не означает, что в Python нет типов данных, они есть, но определяться могут динамически.

Стоит также упомянуть, что Python является интерпретируемым языком. Это означает, что сперва код на Python интерпретируется в промежуточный байт-код, который потом запускается на PVM(Python Virtual Machine). Существует несколько интерпретаторов для Python, например: CPython, IronPython, JPython. Все они написаны на языках C, C# и Java соответственно. Самым популярным является CPython, написанный на языке C. При запуске файла на Python он подается интерпретатору, который построчно выполняет команды. Также интерпретатор генерирует .pyc файлы, которые можно расшифровать как python cache. Эти файлы используются для ускорения последующего запуска одного и того же Python кода.

В Python существует одна небольшая проблема, связанная с интерпретатором CPython. CPython сам по себе не является потокобезопасным при обращении к памяти(выделение памяти для новых объектов). Для разрешения данной проблемы был создан GIL(Global Interpreter Lock). GIL позволяет выполняться только коду из одного потока в единицу времени. Таким образом GIL ограничивает возможности многопоточного программирования на Python. Но также существуют случаи, когда GIL отпускается интерпретатором, что позволяет реализовать своего рода параллельность. Это случается, когда в коде встречается операция ввода-вывода(IO). Во время таких операций код обычно должен подождать пока получит данные по сети или пока операционная система откроет файл. В данных случаях выполнение кода может быть в другом потоке.

Все это привело к созданию асинхронного программирования на Python. Была создана библиотека asyncio, которая и заведует всем асинхронным на Python. В контексте Python асинхронный код всегда обозначает 1 поток, в котором существует некий event loop, ответственный за порядок выполнения задач, смену контекста и смену задачи, пока одна из них требует ожидания. Данная технология активно применяется в разработке бэкэнда для веб-приложений. Так как зачастую бэкэнд на Python должен обрабатывать множество запросов в единицу времени, каждый из которых хочет обратиться к БД, что-то узнать или записать.

**3.1.2** Django и DRF

Django - это высокоуровневый фреймворк для веб-разработки на Python, который позволяет создавать сложные веб-приложения быстро и эффективно. Он известен своей архитектурой "батарейки включены", которая предоставляет широкий набор инструментов и библиотек для решения различных задач веб-разработки.

Django был создан в 2005 году группой разработчиков из газеты "The Lawrence Journal-World" в США. Изначально он был разработан для управления веб-сайтом газеты, но вскоре стал популярным среди разработчиков по всему миру.

Django вместе с собой привносит также паттерн проектирования, по которому Django и работает. Паттерн называется MVT(Model View Template), если пройтись детально по каждому компоненту, то Model – это часть отвечающая за сущности, используемые в приложение, и базу данных, то есть это наши данные в веб-приложении, Template – это часть, ответственная за презентацию данных пользователю, чаще всего это html шаблон, в который вставляются какие-то данные из контекста, например, список постов, список лайков и т.д. Последней частью шаблона является View. Это связующая часть между Model и Template, View обрабатывает входящие запросы от пользователей, взаимодействует с Model, а затем передает результаты взаимодействия в Template для создания ответа пользователю.

С 2011 года в состав Django также входит DRF(Django REST Framework). Это удобный инструмент для создания REST API, которые повсеместно используются в современной веб-разработке. DRF предлагает нам новые возможности: сериализация и десериализация данных Python в понятные форматы(например, JSON), новые built-in Views, позволяющие обрабатывать различные запросы от клиентов, автоматическое создание документации для API, валидация данных, поступающих к API, механизмы аутентификации и авторизации.

**3.1.3** FastAPI

FastAPI – это современный веб-фреймворк, предназначенный для создания web-API. При создании автор опирался на уже существующие Flask и Django REST Framework. FastAPI славится своей скоростью, которая обусловлена асинхронностью фреймворка. Также в фреймворке есть встроенная поддержка OpenAPI. Это средство для документации web-API, в данной документации указано, какие есть точки доступа к нашему API, также, что необходимо отправить вместе с запросом и что мы можем получить в ответ. Еще стоит упомянуть, что есть интеграция с Swagger. Swagger – это набор инструментов, позволяющий документировать API, а также взаимодействовать с ним, например: отправлять запросы к API.

Для работы с данными и их хранилищами в FastAPI нет встроенных механизмов, но существует библиотека SQLAlchemy, являющаяся удобным средство взаимодействия с различными базами данных и другими хранилищами данных. Стоит отметить, что FastAPI и SQLAlchemy почти всегда используются вместе, их интеграция является тривиальной задачей, так что для реализации одного из микросервисов будет использоваться FastAPI + SQLAlchemy для доступа к PostgreSQL.

**3.1.4** React и TypeScript

Frontend – это часть приложения, которую непосредственно видит пользователь, к примеру, внешний вид сайта, мобильного приложения. Во времена зарождения интернета для написания внешнего вида сайтов использовали только html и css, причем первое время сайты были полностью статичные, просто отображая какую-то информацию, но время идет, а технологии развиваются. В современном мире ни один сайт не обходится без JavaScript. Также недавно появился TypeScript, который по сути является тем же JavaScript, только со строгой типизацией. Строгая типизация позволяет писать более обширные проекты, не переживая за то, что тип переменной будет не соответствовать требуемому.

Однако писать frontend, используя один лишь TypeScript довольно нетривиальная задача. Здесь в дело вступает React. React – это одна из самых популярных JavaScript библиотек для создания пользовательских интерфейсов. Также стоит отметить, что React прекрасно работает и с TypeScript. React предлагает разработчикам компонентную архитектуру. В данной архитектуре есть общий компонент App, он является оберткой над всеми остальными компонентами. Удобство React в том, что можно создавать отдельные компоненты для, например, таблиц, полей ввода и др. и переиспользовать их там, где это необходимо. Также очень легко оборачивать компоненты в другие для прикрепления верхней панели(Header) и нижней панели(Footer) к каждому компоненту.

Также будет использовано библиотека axios, которая позволяет отправлять HTTP запросы. Главной целью использования данной библиотеки будет обращение к backend, написанному на Python.

Еще одна важная особенность React – это использование .tsx файлов. Данные файлы также являются файлами с кодом на TypeScript, но позволяют использовать HTML внутри них.

## Описание вспомогательных инструментов разработки

В качестве среды разработки веб-приложения была выбрана среда разработки Pycharm Professional 2022.1 и язык программирования Python 3.10.

Для хранения данных о пользователях была выбрана СУБД PostgreSQL – система управления базами данных, которая работает с таблицами и связями между ними. Управление базой данных осуществлялось с помощью программы pgAdmin, которая предоставляет полный доступ к администрированию PostgreSQL. Также стоит упомянуть, что Pycharm также предоставляет интерфейс для взаимодействия с базой данных, что очень полезно в ситуациях, когда необходимо проверить содержимое таблиц, не отвлекаясь от написания кода, прямо из IDE. PostgreSQL была интегрирована с FastAPI при помощи библиотеки SQLAlchemy. В данном веб-приложении SQLAlchemy предоставляла доступ к асинхронным сессиям базы данных, чтобы обеспечить взаимодействие с асинхронными функциями из FastAPI.

Для хранения информации о страницах, постах, лайках и др. была использована СУБД MySQL. Данная СУБД также является реляционной, как и PostgreSQL, что значит, что идет взаимодействие с таблицами и связями между ними. MySQL была интегрирована с DRF с помощью встроенных средств Django, таких как DjangoORM.

Для взаимодействия между микросервисами была выбрана технология RabbitMQ.

RabbitMQ – это брокер сообщений, основанный на протоколе AMQP(Advanced Message Queueing Protocol).

RabbitMQ позволяет независимым приложениям асинхронно обмениваться сообщениями. Чаще всего одно приложение выступает как отправитель(producer), а второе как получатель(consumer). Сообщения из себя могут представлять различные форматы данных, как, например, JSON и др. Внутри RabbitMQ мы имеем очереди(queue), в которых хранятся сообщения и exchange – объекты, указывающие, кому будут доставлены сообщения из очереди.

Для реализации контейнеризации была выбрана технология Docker.

Docker – это программная платформа для быстрой разработки, тестирования и развертывания приложений. Docker упаковывает ПО в стандартизованные блоки, которые называются [контейнерами](https://aws.amazon.com/ru/containers/). Каждый контейнер включает все необходимое для работы приложения: библиотеки, системные инструменты, код и среду исполнения. Благодаря Docker можно быстро развертывать и масштабировать приложения в любой среде и сохранять уверенность в том, что код будет работать.

Также была использована технология Docker Compose. Данная утилита позволяет управлять несколькими запущенными Docker контейнерами. Это очень полезно для того, чтобы можно было упаковать бэкэнд вместо с базой данных и брокером сообщений в один docker compose файл, а потом запустить все это одной командой.

Для реализации контроля версий и была выбрана технология Git.

Git – система управления версиями с распределенной архитектурой. В отличие от некогда популярных систем вроде CVS и Subversion (SVN), где полная история версий проекта доступна лишь в одном месте, в Git каждая рабочая копия кода сама по себе является репозиторием. Это позволяет всем разработчикам хранить историю изменений в полном объеме.

Разработка в Git ориентирована на обеспечение высокой производительности, безопасности и гибкости распределенной системы.

# ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

В ходе декомпозиции задачи функционал приложения был разделен на 3 микросервиса: UsersManagement, NotificationService и InnoterService.

Микросервис UsersManagement реализует механизм авторизации и аутентификации пользователя. После регистрации пользователя данный сервис записывает данные в PostgreSQL. Данный микросервис отвечает за выдачу JWT токена, используемого в последствии на остальных микросервисах в http запросах.

Микросервис NotificationService отвечает за отправку сообщений на почту пользователям для восстановления пароля. Данный сервис связан с UsersManagement с помощью RabbitMQ. Суть сервиса в постоянном чтении новых сообщений из очереди, обработка данных сообщений, сохранении их в MongoDB и отправлении на почты пользователей с помощью AWS SES.

Микросервис InnoterService отвечает за формирование постов, страниц хранение и редактирование постов. Данный сервис часто может запрашивать данные из UsersManagement для уточнения роли пользователя или вывода списка подписчиков определенной страницы.

UsersManagement и NotificationService следуют архитектуре Ports and Adapters, исходя из которой в самом ядре находится бизнес логика в виде классов use case. Следующим слоем идет абстракция называемая Adapters, которая представляет из себя интерфейсы, через которые бизнес логика будет обращаться к внешним ресурсам, например, базе данных. Последним слоем служит Ports – реализация различных внешних источников данных или утилит, например, интеграция с PostgreSQL или с RabbitMQ. Суть данной архитектуры в отделении бизнес логики от конкретной логики, это позволяет менять реализации источников данных просто написав новый класс в слое Ports. Таким образом имеется ядро бизнес логики, которое практически не меняется, но если мы хотим добавить еще логику, то просто создаем новый use case. Use Case завязаны на интерфейсах, а не на конкретных реализациях. Данная архитектура позволяет полностью абстрагироваться от всех внешних библиотек и фреймворков и использовать в ядре программы чистый Python.

Архитектура InnoterService отличается от других сервисов. По большей части это связано с особенностью Django и DRF. Django предоставляет разработчикам готовую структуру проекта и организация файлов в дереве проекта, а также готовый паттер MVT. В Django имеется файл models.py, отвечающий за сущности программы и бизнес логику. Также есть файл views.py, отвечающий за обработку запросов от пользователей. И папка templates или файл serializers.py(в DRF) отвечающие за презентацию данных пользователю. Таким образом при реализации микросервиса на Django была соблюдена архитектура, которая предоставляется самим фреймворком.

# ТЕСТИРОВАНИЕ

Тестирование программного обеспечения – это метод проверки соответствия фактического программного продукта ожидаемым требованиям, который также необходим, чтобы убедиться, что продукт не содержит дефектов.

Правильно протестированный программный продукт обеспечивает надежность, безопасность и высокую производительность, что в дальнейшем приводит к экономии времени и денег.

По степени автоматизации тестирование можно разделить на 2 основных вида: ручное и автоматизированное.

По объекту тестирования тестирование делится на следующие виды:

* юнит-тестирование – проверка корректной работы отдельных единиц ПО, модулей. Этот вид тестирования могут выполнять сами разработчики;
* интеграционное тестирование – проверка взаимодействия между несколькими единицами ПО;
* сквозное тестирование – проверка полного цикла запроса к веб-сервису.

Переход на каждую новую ступень – движение от микроуровня к макро. Это важный этап тестирования, ведь безошибочно написанные модули могут не работать вместе.

В ходе разработки проекта производилось автоматизированное юнит-тестирование при помощи библиотеки pytest.

Для тестирования сервисов был использован подход ААА. При таком подходе тест разбивается на 3 шага:

* аrrange (настройка) – в этом блоке кода мы настраиваем тестовое окружение тестируемого юнита;
* аct – выполнение или вызов тестируемого сценария;
* аssert – проверка того, что тестируемый вызов ведет себя определенным образом.

Пример разработанного юнит теста для сервиса представлен на рисунке 5.1.

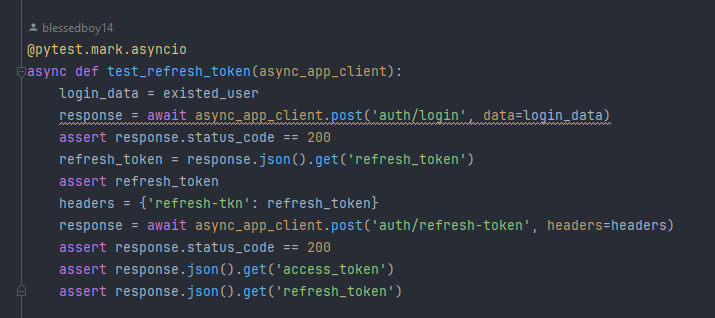


Рисунок 5.1 – end2end тест для сервиса

В ходе тестирования веб-приложения были выявлены дефекты, связанные с некорректной валидацией логина пользователя. В последствии обнаруженные дефекты были исправлены.

Таким образом тестирование проведено успешно и каждый элемент программы выполняет отведённую ему функцию. После тестирования проект готов к использованию пользователем с полностью рабочим функционалом.

**ЗАКЛЮЧЕНИЕ**

В ходе прохождения технологической практики были изучены должностные обязанности техника-программиста, были закреплены теоретические знания по изученным дисциплинам, выполнено индивидуальное задание.

За время прохождения были выполнены задачи практики:

* приобретение профессиональных умений и навыков по специальности;
* закрепление, углубление и систематизация знаний по специальным дисциплинам;
* развитие профессионального мышления и профессиональной самостоятельности;
* формирование навыков организаторской деятельности на производстве и в условиях коллектива;
* повышение уровня квалификации по профессии или приобретение квалификационного разряда по смежной профессии рабочего.

При прохождении практики было выдано и выполнено индивидуальное практическое задание – веб-приложение для создания страниц и постов.

В результате выполнения проекта были получены навыки работы с СУБД MongoDB, Docker, RabbitMQ, а также были укреплены знания языка Python, технологий FastAPI и Django и умения выстраивания гибкой архитектуры веб-приложения.

По итогам технологической практики была изучена специальность инженера-программиста, были закреплены теоретические знания по изученным дисциплинам, выполнено индивидуальное задание.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

[1] Django Framework – [Электронный ресурс] – Режим доступа: <https://www.djangoproject.com>/.

[2] Rest API Tutorial – [Электронный ресурс] – Режим доступа: <https://www.restapitutorial.com/>.

[3] Rest cookbook – [Электронный ресурс] – Режим доступа: <https://restcookbook.com/>.

[4] Роберт Мартин – Чистый код: создание, анализ и рефакторинг. Библиотека программиста. – стр. 464.

[5] React – [Электронный ресурс]. – Режим доступа: <https://react.dev/>

[6] «JWT.io» – [Электронный ресурс]. – Режим доступа: <https://jwt.io/>

[7] FastAPI – [Электронный ресурс]. – Режим доступа: https://fastapi.tiangolo.com/.

[8] Э. Таненбаум, Д. Уэзеролл «Компьютерные сети» 5-е изд. (2016)

[9] Django REST Framework – [Электронный ресурс] – Режим доступа: https://www.django-rest-framework.org/.

[10] Pycharm Professional – [Электронный ресурс] – Режим доступа: https://www.jetbrains.com/ru-ru/pycharm/

[11] Про токены, JSON Web Tokens (JWT), аутентификацию и авторизацию. Token-Based Authentication. – [Электронный ресурс] – Режим доступа: <https://gist.github.com/zmts/802dc9c3510d79fd40f9dc38a12bccfc>

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный код основных функций

from datetime import datetime, timedelta

from src.common.security import (

verify\_password,

get\_password\_hash,

create\_refresh\_jwt,

create\_access\_jwt,

decode\_token,

create\_link\_token,

)

from src.domain.entities.reset\_password\_message import (

ResetPasswordMessage,

ResetPasswordResponse,

)

from src.domain.entities.token import Token

from src.domain.entities.user import User, LoginUser, AuthUser

from src.ports.publisher.message\_publisher import MessagePublisher

from src.ports.repositories.images\_repository import ImagesRepository

from src.ports.repositories.user\_repository import UserRepository

from src.ports.repositories.token\_repository import TokenRepository

from src.use\_cases.exceptions import (

UserNotFoundError,

UserIsBlockedError,

PasswordDoesNotMatchError,

TokenIsBlacklistedError,

NotARefreshTokenError,

InvalidTokenError,

)

class CreateUserUseCase:

def \_\_init\_\_(

self, user\_repository: UserRepository, image\_repository: ImagesRepository

):

self.\_user\_repository = user\_repository

self.\_image\_repository = image\_repository

async def \_manage\_file\_uploading(self, content: bytes, email: str):

created\_user = await self.\_user\_repository.get\_by\_email(email)

s3\_filename = await self.\_image\_repository.upload\_image(created\_user, content)

created\_user.image = s3\_filename

await self.\_user\_repository.partial\_update(created\_user)

async def \_\_call\_\_(self, auth\_user: AuthUser, image: bytes | None) -> User:

hashed\_password = get\_password\_hash(auth\_user.password)

user = User(

email=auth\_user.email,

phone=auth\_user.phone\_number,

username=auth\_user.username,

hashed\_password=hashed\_password,

)

user = await self.\_user\_repository.create(user)

if image is not None:

await self.\_manage\_file\_uploading(image, user.email)

return user

class LoginUseCase:

def \_\_init\_\_(

self, user\_repository: UserRepository, token\_repository: TokenRepository

):

self.\_user\_repository = user\_repository

self.\_token\_repository = token\_repository

async def \_\_call\_\_(self, login\_model: LoginUser) -> Token:

database\_user = await self.\_user\_repository.get\_by\_login(login\_model.login)

if not database\_user:

raise UserNotFoundError(login\_model.login)

if database\_user.is\_blocked:

raise UserIsBlockedError(database\_user.id)

if not verify\_password(login\_model.password, database\_user.hashed\_password):

raise PasswordDoesNotMatchError(database\_user.id)

data = {'user\_id': database\_user.id.hex}

refresh\_token = create\_refresh\_jwt(data)

await self.\_token\_repository.set(database\_user.id.hex, refresh\_token)

access\_token = create\_access\_jwt(data)

return Token(

access\_token=access\_token,

refresh\_token=refresh\_token,

type='bearer',

)

class RefreshTokenUseCase:

def \_\_init\_\_(

self, user\_repository: UserRepository, token\_repository: TokenRepository

):

self.\_user\_repository = user\_repository

self.\_token\_repository = token\_repository

async def \_is\_token\_valid(self, token: str, user\_id: str):

return (await self.\_token\_repository.get(user\_id)) == token

async def \_blacklist\_token(self, token: str):

await self.\_token\_repository.blacklist(token)

async def \_\_call\_\_(self, refresh\_token: str) -> Token:

payload = decode\_token(refresh\_token)

user\_id = payload.get('user\_id')

if not (await self.\_is\_token\_valid(refresh\_token, user\_id)):

raise TokenIsBlacklistedError()

token\_mode = payload.get('mode')

if token\_mode and token\_mode == 'refresh\_token':

await self.\_token\_repository.remove(user\_id)

data = {'user\_id': user\_id}

access\_token = create\_access\_jwt(data)

refresh\_token = create\_refresh\_jwt(data)

await self.\_token\_repository.set(user\_id, refresh\_token)

return Token(

access\_token=access\_token,

refresh\_token=refresh\_token,

type='bearer',

)

else:

raise NotARefreshTokenError()

class ResetPasswordUseCase:

def \_\_init\_\_(self, user\_repository: UserRepository, publisher: MessagePublisher):

self.\_user\_repository = user\_repository

self.\_message\_publisher = publisher

@staticmethod

def \_generate\_reset\_password\_url(email: str, user\_id: str):

payload = {

'email': email,

'type': 'reset\_password',

'expiry': (datetime.utcnow() + timedelta(days=30)).isoformat(),

}

# logger.debug(f'generating reset password link for: {email}')

token = create\_link\_token(payload)

reset\_link = (

f'https://127.0.0.1:8000/auth/reset-password/{user\_id}?token=' + token

)

return reset\_link

async def \_\_call\_\_(self, email: str) -> ResetPasswordResponse:

user = await self.\_user\_repository.get\_by\_email(email)

if not user:

raise UserNotFoundError(email)

message = ResetPasswordMessage(

user\_id=str(user.id),

subject=f'Resetting Password To Your Account: {email}',

body='Click this link to reset your password '

f'{self.\_generate\_reset\_password\_url(email, str(user.id))}',

email=email,

published\_at=datetime.utcnow().isoformat(),

)

self.\_message\_publisher.publish(message)

return ResetPasswordResponse(

message='Reset link was sent to your email', email=email

)

class AuthUserWithJWTUseCase:

def \_\_init\_\_(self, user\_repository: UserRepository):

self.\_user\_repository = user\_repository

async def \_\_call\_\_(self, access\_token: str) -> User:

payload = decode\_token(access\_token)

user\_id = payload.get('user\_id')

if user\_id is None:

raise InvalidTokenError('Token invalid')

user = await self.\_user\_repository.get\_by\_id(user\_id)

if user is None:

raise InvalidTokenError("User don't exist")

return user

from dataclasses import replace

from src.domain.entities.user import User, UserFastInfo

from src.ports.repositories.images\_repository import ImagesRepository

from src.ports.repositories.user\_repository import UserRepository

from src.use\_cases.exceptions import (

EmptyUpdateDataError,

MethodNotAllowedError,

UserNotFoundError,

)

from src.drivers.rest.routers.schema import RoleEnum, UserId

class PatchMeUseCase:

def \_\_init\_\_(self, user\_repository: UserRepository):

self.\_user\_repository = user\_repository

async def \_\_call\_\_(self, updated\_user: dict | None, current\_user: User) -> User:

if not updated\_user:

raise EmptyUpdateDataError(current\_user.id)

updated\_model = replace(current\_user, \*\*updated\_user)

return await self.\_user\_repository.partial\_update(updated\_model)

class DeleteMeUseCase:

def \_\_init\_\_(self, user\_repository: UserRepository):

self.\_user\_repository = user\_repository

async def \_\_call\_\_(self, current\_user: User) -> None:

await self.\_user\_repository.delete(str(current\_user.id))

class DeleteUserByIdUseCase:

def \_\_init\_\_(self, user\_repository: UserRepository):

self.\_user\_repository = user\_repository

async def \_\_call\_\_(self, current\_user: User, user\_id: str) -> None:

if current\_user.role is RoleEnum.ADMIN:

await self.\_user\_repository.delete(user\_id)

else:

raise MethodNotAllowedError(current\_user.id)

class PatchUserByIdUseCase:

def \_\_init\_\_(self, user\_repository: UserRepository):

self.\_user\_repository = user\_repository

async def \_\_call\_\_(

self, current\_user: User, user\_id: str, updated\_user: dict | None

) -> User:

if current\_user.role is not RoleEnum.ADMIN:

raise MethodNotAllowedError(current\_user.id)

if not updated\_user:

raise EmptyUpdateDataError(current\_user.id)

requested\_user = await self.\_user\_repository.get\_by\_id(user\_id)

if requested\_user:

updated\_model = replace(requested\_user, \*\*updated\_user)

return await self.\_user\_repository.partial\_update(updated\_model)

else:

raise UserNotFoundError(user\_id)

class GetUserByIdUseCase:

def \_\_init\_\_(self, user\_repository: UserRepository):

self.\_user\_repository = user\_repository

async def \_\_call\_\_(self, current\_user: User, user\_id: str) -> User:

if current\_user.role is RoleEnum.USER:

raise MethodNotAllowedError(current\_user.id)

user = await self.\_user\_repository.get\_by\_id(user\_id)

if user:

if (

current\_user.role is RoleEnum.MODERATOR

and user.group == current\_user.group

):

return user

elif current\_user.role is RoleEnum.ADMIN:

return user

else:

raise UserNotFoundError(user\_id)

class ListUsersUseCase:

def \_\_init\_\_(self, user\_repository: UserRepository):

self.\_user\_repository = user\_repository

async def \_\_call\_\_(self, current\_user: User, filters: dict) -> list[User]:

if current\_user.role is RoleEnum.USER:

raise MethodNotAllowedError(current\_user.id)

if current\_user.role is RoleEnum.ADMIN:

return await self.\_user\_repository.list(\*\*filters)

if current\_user.role is RoleEnum.MODERATOR:

return await self.\_user\_repository.list\_from\_same\_group(

str(current\_user.group), \*\*filters

)

class FetchUsernamesUseCase:

def \_\_init\_\_(self, user\_repository: UserRepository):

self.\_user\_repository = user\_repository

async def \_\_call\_\_(

self, current\_user: User, user\_ids: [UserId]

) -> list[UserFastInfo]:

converted\_ids = [user.id.hex for user in user\_ids]

usernames = await self.\_user\_repository.fetch\_usernames(converted\_ids)

return usernames

class UploadImageUseCase:

def \_\_init\_\_(

self, images\_repository: ImagesRepository, user\_repository: UserRepository

):

self.\_images\_repository = images\_repository

self.\_user\_repository = user\_repository

async def \_\_call\_\_(self, current\_user: User, content: bytes | None) -> str:

s3\_filename = await self.\_images\_repository.upload\_image(current\_user, content)

to\_update = {'image': s3\_filename}

updated\_item = replace(current\_user, \*\*to\_update)

await self.\_user\_repository.partial\_update(updated\_item)

return s3\_filename

from typing import Annotated

from fastapi import (

APIRouter,

status,

Depends,

Form,

UploadFile,

File,

Header,

)

from fastapi.security import OAuth2PasswordRequestForm

from pydantic import EmailStr, ValidationError

from pydantic\_extra\_types.phone\_numbers import PhoneNumber

from src.drivers.rest.routers.exceptions import ModelValidationError

from src.drivers.rest.routers.schema import (

UserIn,

TokenSchema,

ResetPasswordRequest,

ResponseUser,

ResetPasswordResponse,

)

from src.drivers.rest.routers.config import logger

from src.domain.entities.user import LoginUser

from src.drivers.rest.dependencies import (

get\_sign\_up\_use\_case,

get\_login\_use\_case,

get\_refresh\_token\_use\_case,

get\_reset\_password\_use\_case,

)

from src.use\_cases.auth\_use\_cases import (

CreateUserUseCase,

LoginUseCase,

RefreshTokenUseCase,

ResetPasswordUseCase,

)

router = APIRouter()

@router.post(

'/login',

status\_code=status.HTTP\_200\_OK,

response\_model=TokenSchema,

summary='Login',

)

async def login(

use\_case: Annotated[LoginUseCase, Depends(get\_login\_use\_case)],

form\_data: OAuth2PasswordRequestForm = Depends(),

):

user = LoginUser(login=form\_data.username, password=form\_data.password)

return await use\_case(user)

@router.post(

'/signup',

status\_code=status.HTTP\_201\_CREATED,

response\_model=ResponseUser,

summary='Sign Up',

)

async def signup(

use\_case: Annotated[CreateUserUseCase, Depends(get\_sign\_up\_use\_case)],

username: Annotated[str, Form()],

phone: Annotated[PhoneNumber, Form()],

email: Annotated[EmailStr, Form()],

password: Annotated[str, Form()],

image: UploadFile = File(None),

):

try:

user\_in = UserIn(email=email, phone=phone, password=password, username=username)

except ValidationError as err:

logger.error(f'failed to create model from sign up input with err: {err}')

raise ModelValidationError(repr(err.errors()[0]['msg']))

file\_bytes = None if image is None else await image.read()

return await use\_case(user\_in.to\_entity(), file\_bytes)

@router.post(

'/refresh-token',

status\_code=status.HTTP\_200\_OK,

response\_model=TokenSchema,

summary='Refresh Both Tokens',

)

async def refresh\_token(

use\_case: Annotated[RefreshTokenUseCase, Depends(get\_refresh\_token\_use\_case)],

refresh\_tkn: Annotated[str, Header()],

):

return await use\_case(refresh\_tkn)

@router.post(

'/reset-password',

status\_code=status.HTTP\_200\_OK,

response\_model=ResetPasswordResponse,

summary='Reset Your Password',

)

async def reset\_password(

request: ResetPasswordRequest,

use\_case: Annotated[ResetPasswordUseCase, Depends(get\_reset\_password\_use\_case)],

):

return await use\_case(request.email)

from typing import Annotated, List

from fastapi import APIRouter, Depends, status, UploadFile, Query

from src.domain.entities.user import User

from src.drivers.rest.dependencies import (

get\_patch\_me\_use\_case,

get\_delete\_me\_use\_case,

get\_get\_user\_by\_id\_use\_case,

get\_patch\_user\_by\_id\_use\_case,

get\_list\_users\_use\_case,

get\_upload\_file\_use\_case,

get\_delete\_by\_id\_use\_case,

get\_auth\_with\_jwt\_use\_case,

get\_fetch\_username\_use\_case,

)

from src.use\_cases.auth\_use\_cases import AuthUserWithJWTUseCase

from src.use\_cases.users\_use\_cases import (

PatchMeUseCase,

DeleteMeUseCase,

GetUserByIdUseCase,

PatchUserByIdUseCase,

ListUsersUseCase,

UploadImageUseCase,

DeleteUserByIdUseCase,

FetchUsernamesUseCase,

)

from src.drivers.rest.routers.schema import (

UserPatch,

UserBase,

AdminPatch,

OrderByEnum,

UserFastInfo,

UserId,

)

from src.common.security import oauth2\_scheme

router = APIRouter()

async def get\_current\_user(

access\_token: Annotated[str, Depends(oauth2\_scheme)],

use\_case: Annotated[AuthUserWithJWTUseCase, Depends(get\_auth\_with\_jwt\_use\_case)],

) -> User:

return await use\_case(access\_token)

@router.get(

'/me',

status\_code=status.HTTP\_200\_OK,

response\_model=UserBase,

summary='Get Yourself',

)

async def read\_users\_me(cur\_user: Annotated[User, Depends(get\_current\_user)]):

return cur\_user

@router.post(

'/batch',

status\_code=status.HTTP\_200\_OK,

response\_model=List[UserFastInfo],

summary='Fetch usernames',

)

async def batch\_usernames(

cur\_user: Annotated[User, Depends(get\_current\_user)],

user\_ids: List[UserId],

use\_case: Annotated[FetchUsernamesUseCase, Depends(get\_fetch\_username\_use\_case)],

):

return await use\_case(cur\_user, user\_ids)

@router.patch(

'/me',

status\_code=status.HTTP\_200\_OK,

response\_model=UserBase,

summary='Patch Yourself',

)

async def update\_user\_me(

updated\_user: UserPatch,

cur\_user: Annotated[User, Depends(get\_current\_user)],

use\_case: Annotated[PatchMeUseCase, Depends(get\_patch\_me\_use\_case)],

):

updated\_dict = updated\_user.model\_dump(exclude\_unset=True)

return await use\_case(updated\_dict, cur\_user)

@router.delete('/me', status\_code=status.HTTP\_200\_OK, summary='Delete Me')

async def delete\_me(

cur\_user: Annotated[User, Depends(get\_current\_user)],

use\_case: Annotated[DeleteMeUseCase, Depends(get\_delete\_me\_use\_case)],

):

await use\_case(cur\_user)

@router.get(

'/{user\_id}',

status\_code=status.HTTP\_200\_OK,

response\_model=UserBase,

summary='Get User By Id',

)

async def get\_user\_by\_id(

cur\_user: Annotated[User, Depends(get\_current\_user)],

use\_case: Annotated[GetUserByIdUseCase, Depends(get\_get\_user\_by\_id\_use\_case)],

user\_id: str,

):

return await use\_case(cur\_user, user\_id)

@router.patch(

'/{user\_id}',

status\_code=status.HTTP\_200\_OK,

response\_model=UserBase,

summary='Patch User As Admin',

)

async def patch\_user(

updated\_user: AdminPatch,

user\_id: str,

cur\_user: Annotated[User, Depends(get\_current\_user)],

use\_case: Annotated[PatchUserByIdUseCase, Depends(get\_patch\_user\_by\_id\_use\_case)],

):

updated\_dict = updated\_user.model\_dump(exclude\_unset=True)

return await use\_case(cur\_user, user\_id, updated\_dict)

@router.get(

'',

status\_code=status.HTTP\_200\_OK,

response\_model=list[UserBase],

summary='List Users',

)

async def list\_users(

cur\_user: Annotated[User, Depends(get\_current\_user)],

use\_case: Annotated[ListUsersUseCase, Depends(get\_list\_users\_use\_case)],

page: Annotated[int, Query(gt=0)] = 1,

limit: Annotated[int, Query(gt=-1, lt=101)] = 30,

filter\_by\_name: str = '',

sort\_by: str = 'username',

order\_by: OrderByEnum = OrderByEnum.DESC,

):

filters = {

'page': page,

'limit': limit,

'filter\_by\_name': filter\_by\_name,

'sort\_by': sort\_by,

'order\_by': order\_by,

}

return await use\_case(cur\_user, filters)

@router.post('/me/upload\_image', status\_code=status.HTTP\_200\_OK, summary='Upload Image')

async def upload\_avatar\_image(

cur\_user: Annotated[User, Depends(get\_current\_user)],

use\_case: Annotated[UploadImageUseCase, Depends(get\_upload\_file\_use\_case)],

file: UploadFile,

):

file\_bytes = await file.read()

s3\_filename = await use\_case(cur\_user, file\_bytes)

return {'image': s3\_filename, 'status': 'uploaded'}

@router.delete('/{user\_id}', status\_code=status.HTTP\_200\_OK, summary='Delete User')

async def delete\_user\_as\_admin(

cur\_user: Annotated[User, Depends(get\_current\_user)],

user\_id: str,

use\_case: Annotated[DeleteUserByIdUseCase, Depends(get\_delete\_by\_id\_use\_case)],

):

await use\_case(cur\_user, user\_id)

return {'message': f'user with id {user\_id} deleted'}