**Содержание**

[Введение 3](#_Toc165826347)

[1 Постановка задачи 4](#_Toc165826348)

[1.1 Обзор аналогичных решений 4](#_Toc165826349)

[1.2 Спецификация требований 6](#_Toc165826350)

[2 Проектирование web-приложения 7](#_Toc165826351)

[2.1 Проектирование вариантов использования 7](#_Toc165826352)

[2.2 Структура web-приложения 8](#_Toc165826353)

[2.3 Проектирование базы данных 8](#_Toc165826354)

[2.4 Проектирование сервера web-приложения 13](#_Toc165826355)

[3 Разработка web-приложения 19](#_Toc165826356)

[3.1 Разработка бэкэнда 19](#_Toc165826357)

[3.2 Разработка фронтэнда 22](#_Toc165826358)

[4 Тестирование web-приложения 25](#_Toc165826359)

[5 Руководство пользователя 28](#_Toc165826360)

[Заключение 34](#_Toc165826361)

[Список используемых источников 35](#_Toc165826362)

[Приложение А 36](#_Toc165826363)

Введение

Прохождение практики в период с 17.06.2024 по 12.07.2024 происходило в компании ЗАСО «Белнефтестрах».

Целью производственной практики является закрепление на практике полученных в ВУЗе знаний и умений. Также немаловажным является приобретение новых знаний и навыков, которые в будущем будут применяться в трудовой деятельности, и приобретение практических навыков самостоятельной работы, тренировка их применения при решении конкретных задач и проблем.

В качестве темы проекта производственной практики было получено «Веб-приложение для обмена документами в рамках отдела». Для реализации данного программного продукта был выбран язык Python.

В качестве системы управления базой данных была использована СУБД PostgreSQL.

1 Постановка задачи

1.1 Обзор аналогичных решений

В качестве первого аналога была рассмотрена электронная почта от компании «Яндекс». Она позволяет осуществлять поиск по контактам, прикреплять файлы и форматировать сообщения. Это удобные и привычные функции для пользователей электронной почты. Пример страницы Fiverr представлен на рисунке 1.1.

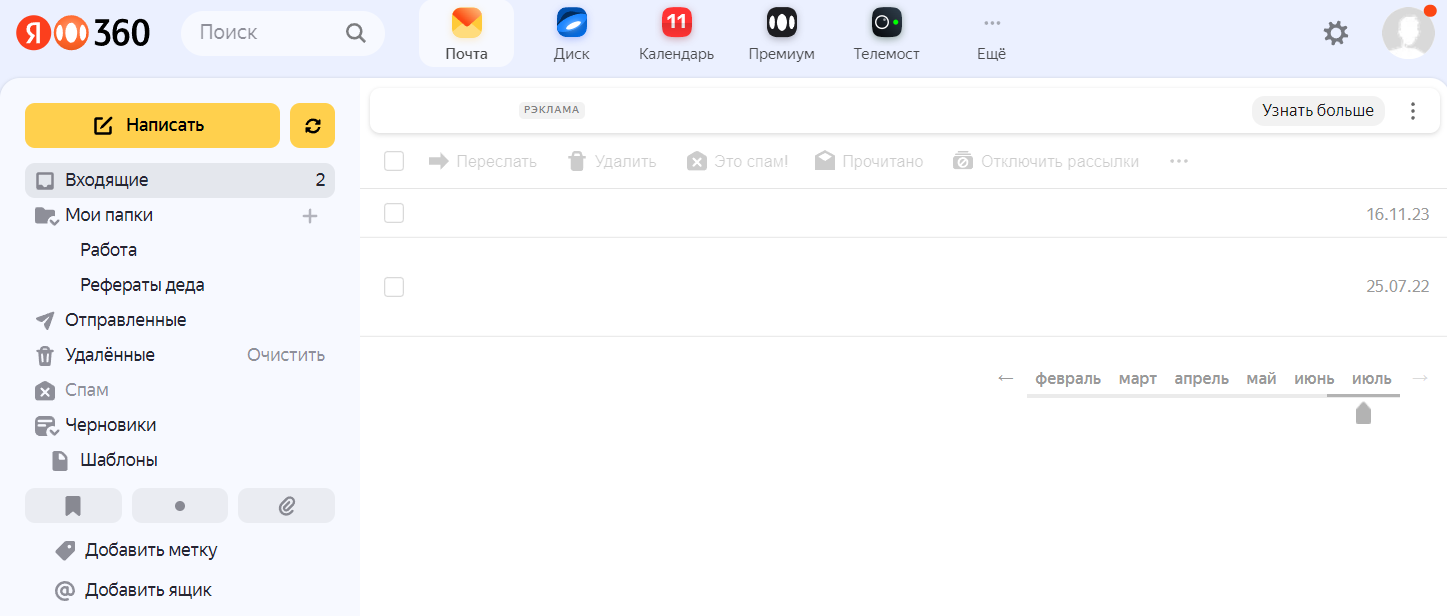


Рисунок 1.1 – Страница «Яндекс Почты»

К недостаткам можно отнести чрезвычайно большое количество доступных функций, занимающих место на странице, которыми пользователи, скорее всего, не будут пользоваться. Это может перегружать интерфейс и отвлекать от основных задач.

Кроме того, невозможность хранения данных локально является недостатком, так как пользователи вынуждены доверять компании «Яндекс» и полагаться на их серверы для хранения своей переписки и файлов. Это ставит под вопрос конфиденциальность и безопасность данных.

В качестве второго аналога был рассмотрен сервис электронной почты iRedMail. Он позволяет развернуть собственный локальный сервер электронной почты. Это дает пользователям больший контроль над своими данными, так как вся переписка и файлы хранятся на их собственном сервере, а не на серверах сторонних компаний.

Данный сервер обладает следующими преимуществами:

– открытый исходный код: обеспечивает прозрачность и возможность настройки под нужды организации;

– масштабируемость: может обслуживать от нескольких до сотен пользователей;

– безопасность и конфиденциальность;

– самостоятельное размещение сервера.

Пример страницы платформы представлен на рисунке 1.2.

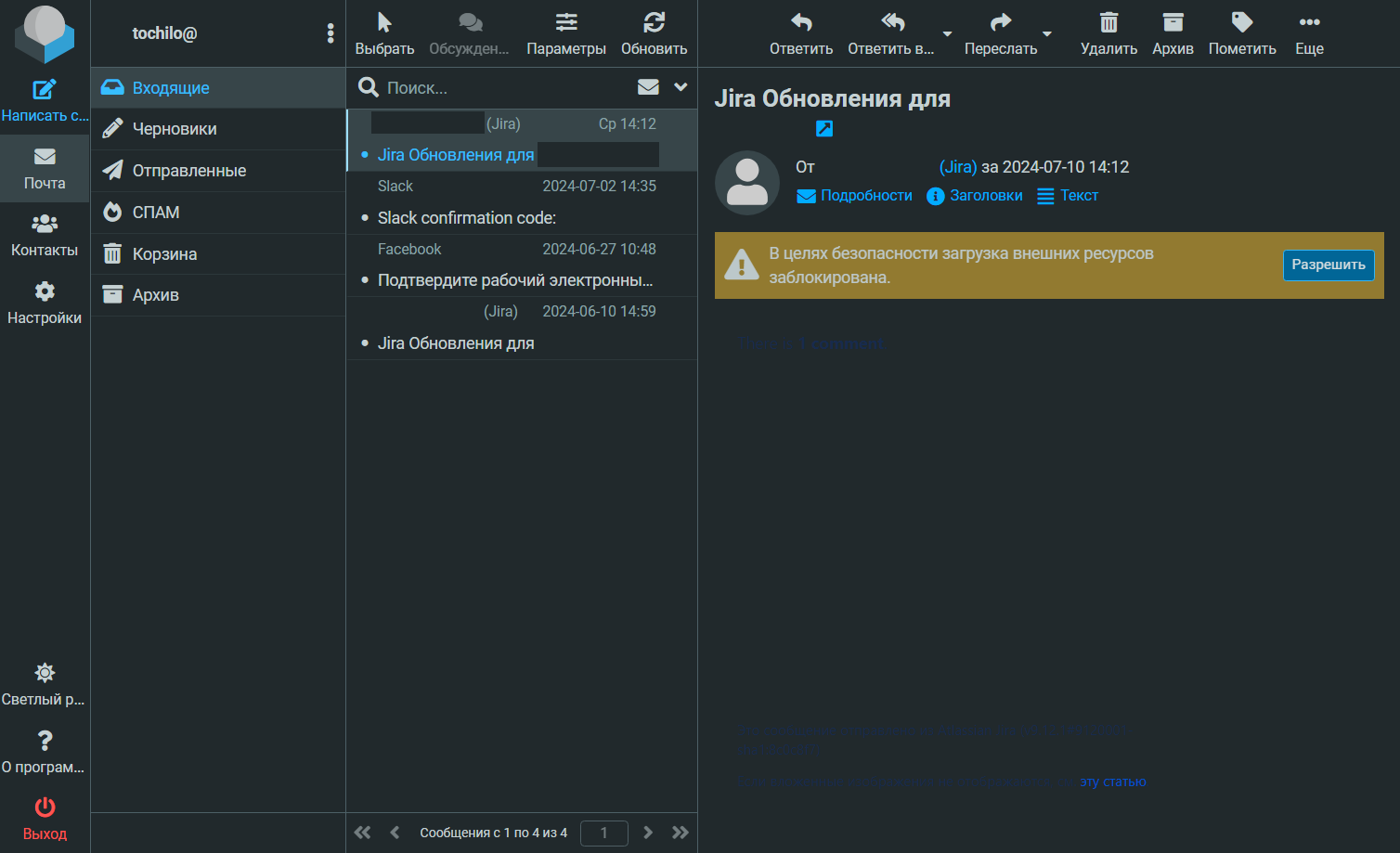


Рисунок 1.2 – Страница Upwork

Данный сервер предлагает меньший по сравнению с почтой от «Яндекса» набор функций, который тем не менее охватывает большинство пользовательских потребностей. Это более простой и лаконичный интерфейс, что может быть предпочтительно для тех, кто не нуждается в большом количестве дополнительных возможностей.

К недостаткам данного решения можно отнести следующее:

– сложность установки и настройки: развертывание собственного почтового сервера требует определенных технических знаний и навыков администрирования;

– требуется ручная настройка многих компонентов, таких как веб-интерфейс, SMTP, IMAP, базы данных и т.д.;

– iRedMail, как открытое программное обеспечение, не предоставляет официальной технической поддержки пользователям;

– интеграция с другими приложениями и сервисами может быть более сложной;

– интеграция с мобильными устройствами и приложениями может требовать дополнительной настройки.

1.2 Спецификация требований

На основе рассмотренных аналогичных решений были сформированы следующие требования к программному продукту:

– возможность создавать, изменять, удалять и получать сообщения с текстом и файлами;

– разделение ролей на обычных пользователей, администраторов и другие;

– изменение статусов сообщений;

– возможность скачивания файлов, сжатых в архив;

– возможность создания, изменения и удаления пользователей администратором.

Всем пользователям должна быть доступна возможность просматривать входящие сообщения и исходящие сообщения, скачивать вложенные файлы и изменять сообщения. Также всем пользователям должна быть доступна возможность создания сообщений и загрузки своих файлов. Пользователи должны иметь возможность обновить статус своих сообщений на принятый либо отклонённый.

Администратору должна быть доступна возможность создания, изменения и удаления пользователей.

Обычные пользователи не должны иметь возможность зарегистрироваться. Для этого они должны обратиться к администратору.

2 Проектирование web-приложения

2.1 Проектирование вариантов использования

Согласно сформулированным требованиям, была создана диаграмма вариантов использования, представленная на рисунке 2.1.

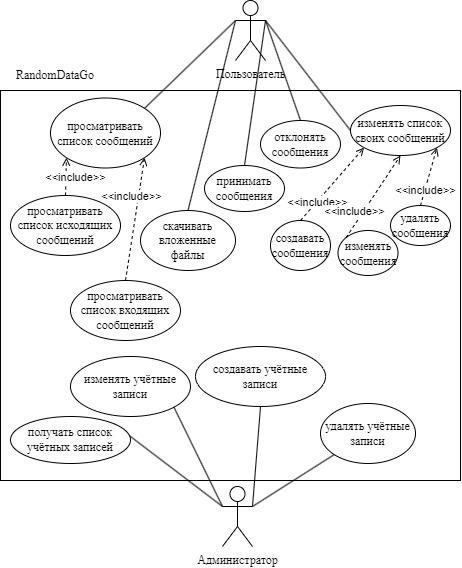


Рисунок 2.1 – Диаграмма вариантов использования

Разным категориям пользователей доступны разные действия. Так, обычные пользователи могут просматривать входящие сообщения и исходящие сообщения, скачивать вложенные файлы и изменять сообщения. Также всем пользователям доступна возможность создания сообщений с прикреплёнными файлами. Пользователи имеют возможность обновить статус своих сообщений на принятый либо отклонённый.

Администраторам доступно управление учётными записями: получение списка всех учётных записей, их создание, изменение и удаление.

2.2 Структура web-приложения

Приложение состоит из четырёх частей: базы данных с СУБД PostgreSQL, web-сервера nginx, сервера приложения, созданного при помощи фреймворка FastAPI и фронтэнд-приложения, созданного при помощи фреймворка Vue.js. База данных, web-сервер и сервер приложения находятся в Docker-контейнерах. За обработку запросов к серверу приложения и раздачу статических файлов отвечает сервер nginx.

Обобщённая структура web-приложения представлена на рисунке 2.2.

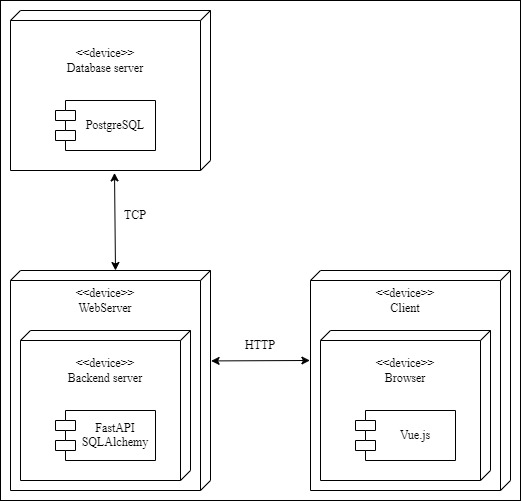


Рисунок 2.2 – Диаграмма развёртывания web-приложения

Согласно данной схеме, клиент и сервер находятся на разных устройствах. Клиент использует браузер для отправки запросов к web-серверу. В свою очередь web-сервер решает, перенаправить запрос на сервер приложения или вернуть статический файл.

Web-сервер обрабатывает запросы при помощи сервера, созданного при помощи фреймворка FastAPI. В случае необходимости сервер отправляет запросы к базе данных, находящейся в контейнере Docker и находящейся под управлением СУБД PostgreSQL.

Для выполнения запросов к базе данных используется ORM SQLAlchemy.

2.3 Проектирование базы данных

Согласно схеме вариантов использования была создана база данных. Её логическая схема представлена на рисунке 2.3. Описание моделей представлено в приложении А.

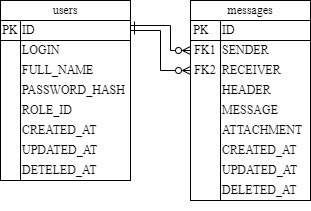


Рисунок 2.3 – Логическая схема базы данных

База данных содержит две таблицы, хранящих информацию о пользователях и сообщениях. Типы данных были выбраны согласно документации [1].

Таблица users хранит информацию о соискателях. Описание её столбцов представлено в таблице 2.1.

Таблица 2.1 – Описание таблицы users

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | UUID | идентификатор пользователя, первичный ключ |
| login | varchar(30) | логин пользователя |
| full\_name | varchar(100) | имя пользователя |
| password\_hash | varchar(70) | хеш пароля пользователя |
| role\_id | integer | идентификатор роли пользователя |
| created\_at | timestamp | время создания пользователя |
| updated\_at | timestamp | время изменения пользователя |
| deleted\_at | timestamp | время удаления пользователя |

Описание столбцов таблицы messages представлено в таблице 2.2.

Таблица 2.2 – Описание таблицы messages

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | UUID | идентификатор сообщения, первичный ключ |
| header | varchar | заголовок сообщения |
| message | varchar | текст сообщения |
| attachment | BYTEA | прикреплённый файл архива |
| sender | UUID | идентификатор отправителя |

Продолжение таблицы 2.2

|  |  |  |
| --- | --- | --- |
| receiver | UUID | идентификатор получателя, внешний клюс |
| status | integer | идентификатор статуса сообщения |
| created\_at | timestamp | время создания пользователя |
| updated\_at | timestamp | время изменения пользователя |
| deleted\_at | timestamp | время удаления пользователя |

Данная таблица хранит информацию о сообщениях, такую как текст, прикреплённые файлы, идентификатор отправителя и другую.

2.4 Проектирование сервера web-приложения

Для обработки запросов применяется три роутера, каждый из которых обрабатывает запросы к определённым адресам. Так, auth\_router обрабатывает все запросы, связанные с аутентификацией, message\_router обрабатывает запросы на CRUD-операции с сообщениями, а user\_router – на CRUD-операции с пользователями. Список представляемых обработчиков роутера auth\_router представлен в таблице 2.3.

Таблица 2.3 – Cписок представляемых обработчиков роутера masterRouter

|  |  |  |
| --- | --- | --- |
| Адрес | Метод | Описание |
| /register-admin | GET | Проверяет правильность переданного секрета и создаёт нового пользователя-администратора с заданными параметрами |
| /login | GET | Проверяет данные аутентификации и возвращает маркеры доступа пользователя |
| /refresh | GET | Проверяет валидность маркера обновления и возвращает новую пару маркеров |
| /logout | PUT | Удаляет cookie-файлы пользователя |

Также в конвейере обработки запросов используется CORSMiddleware, предназначенное для разрешения запросов, приходящих с доменов, отличающихся от домена сервера приложения, что чрезвычайно полезно при тестировании приложения.

3 Разработка web-приложения

3.1 Разработка бэкэнда

Для разработки бэкэнда был использован фреймворк FastAPI. Согласно [2], в нём для обработки запросов могут применяться простые обработчики запросов. Обработчик запросов, позволяющий скачать вложенный файл сообщения по его идентификатору, представлен в листинге 3.1.

|  |
| --- |
| @message\_router.get('/all/')  async def get\_all\_messages\_list(  jwt\_payload: Annotated[dict, str] = Depends(  JWTCookieBearer(role='admin')  ),  page\_params: PageParams = Depends(),  db\_session: AsyncSession = Depends(create\_session)  ):  messages, total\_elements = await MessageService.get\_all(  params=page\_params,  db\_session=db\_session  )  return PaginationListScheme(  total\_elements=total\_elements,  elements=list(map(  lambda x: MessageScheme(x),  messages  ))  ) |

Листинг 3.1 – Простой обработчик запросов

Также для обработки запросов могут использоваться более продвинутые роутеры. Фрагмент роутера message\_router представлен в листинге 3.2.

|  |
| --- |
| message\_router = APIRouter(  prefix='/message',  tags=[  'Messages'  ]  ) |
|  |

Листинг 3.2 – Объявление роутера message\_router

Объекты запроса и ответа могут быть получены путём добавления параметров в функцию-обработчик запросов. Объект ответа также может быть , над которым можно производить различные операции, такие как получение и установка cookie, чтение тела запроса, установка тела ответа при помощи оператора return и так далее.

Для запуска сервера необходимо создать объект приложения, определить обработчики запросов и роутеры, присоединить роутеры к приложению и при необходимости определить промежуточные обработчики на конвейере обработки запросов. Код запуска сервера представлен в листинге 3.3.

|  |
| --- |
| app = FastAPI()  app.exception\_handler(HTTPException)(http\_exception\_handler)  app.exception\_handler(500)(internal\_exception\_handler)  app.include\_router(auth\_router)  app.include\_router(message\_router)  app.include\_router(users\_router)  if AppConfig.debug:  origins = [  f"{AppConfig.host}",  f"{AppConfig.host}:5173",  ]  print('ORIGINS', origins)  app.add\_middleware(  CORSMiddleware,  allow\_origins=origins,  allow\_credentials=True,  allow\_methods=["\*"],  allow\_headers=["\*"],  ) |

Листинг 3.3 – Код запуска сервера

Для получения данных из базы данных использовалась ORM SQLAlchemy. Применялся подход model-first, при котором в первую очередь разрабатывались модели данных, а затем в соответствии с ними создавались таблицы и обработчики запросов. Были определены модели для пользователей и сообщений. Объявление модели пользователей представлено в листинге 3.4.

|  |
| --- |
| class UserModel(TimestampMixin, DeletedMixin, Base):  id: Mapped[str] = mapped\_column(  UUID(as\_uuid=True), primary\_key=True, default=uuid.uuid4  )  full\_name: Mapped[str] = mapped\_column(  String(100), nullable=False  )  login: Mapped[str] = mapped\_column(  String(30), unique=True, nullable=False  )  password\_hash: Mapped[str] = mapped\_column(  String(70), nullable=False  )  role\_id: Mapped[int] = mapped\_column(  Integer(),  nullable=False  ) |

Листинг 3.4 – Объявление модели пользователя

Для поиска, создания, обновления и удаления записей в базе данных применялись функции SQLAlchemy, такие как select(UserModel), update(UserModel), методы filter и filter\_by и другие.

Секретные строки для auth-токена и refresh-токена, а также прочие конфигурационные данные хранятся в файле settings.py. Соль для хеширования паролей хранится в файле .env и устанавливается при построении Docker-контейнера.

3.2 Разработка фронтэнда

Для разработки сайта использовался фреймворк Vue.js в качестве базы, пакет vue-router для создания нескольких страниц с разными URI и библиотека vue-paginate для упрощения добавления пагинации данных [3]. Фрагмент роутера фронтэнд-приложения представлен в листинге 3.5.

|  |
| --- |
| import Home from '../pages/Home.vue';  import UsersList from '../pages/UsersList.vue';  import EditUser from '../pages/EditUser.vue';  const routes = [  {  path: '/',  component: GeneralWrapper,  children: [  {  path: '',  name: 'Home',  component: Home  },  {  path: 'messages/:messageMode/:filterOption',  name: 'MessagesList',  component: MessagesList  }, |

Листинг 3.5 – Фрагмент роутера фронтэнд-приложения

За каждую страницу отвечает свой компонент, который запрашивает данные с сервера и отображает их на странице. Для запроса данных с сервера была разработана специальная функция-декоратор fetchWithCredentials, которая запрашивает данные с определённого адреса и в зависимости от возвращённого статуса ответа либо возвращает данные, либо перенаправляет пользователя на главную страницу или на страницу выхода из учётной записи в случае ошибки. В случае, если у пользователя истёк auth-токен, но не истёк refresh-токен, данная функция запрашивает у сервера новую пару токенов и повторяет запрос. Также была разработана функция fetchWithNotification, использующая пакет vue3-notification, выводяая сообщение об успехе и использующая функцию fetchWithCredentials. Код функции fetchWithNotification представлен в листинге 3.6.

|  |
| --- |
| export async function fetchWithCredentials(url: string, options: Object | undefined = undefined) {  const result = await fetch(url, {  ...options,  credentials: 'include'  });  if (result.ok) {  return result.json();  }  else if(result.status === 401) {  const refreshResult = await fetch(`${AppConfig.api\_host}/auth/refresh`, {  credentials: 'include'  });  else {  location.href = '/login';  return null;  }  } else {  throw await getErrorBody(result);  }  } err = await err;  console.log(err);  if (err.code === 401) {  location.href = '/signout';  } else if (err.code === 403) {  location.href = '/';  }  })  .then(d => {  return d;  });} |

Листинг 3.6 – Функция запроса данных с сервера с уведомлением об успехе

Также для отображения номеров страниц в случаях, когда содержимое не помещается на одну страницу, применяется подключаемое расширение vue-paginate. Оно позволяет запрашивать у сервера данные в нужном количестве и с нужным смещением относительно начала. Объём данных и смещение задаются в файле settings.ts фронтэнд-приложения.

4 Тестирование web-приложения

Для тестирования web-приложения использовалось ручное тестирование. Обработчики запросов были проверены на возможность неавторизованного и неаутентифицированного доступа и поведение при передаче неожиданных параметров при помощи автоматически сгенерированной swagger-документации. Страница документации представлена на рисунке 4.1.

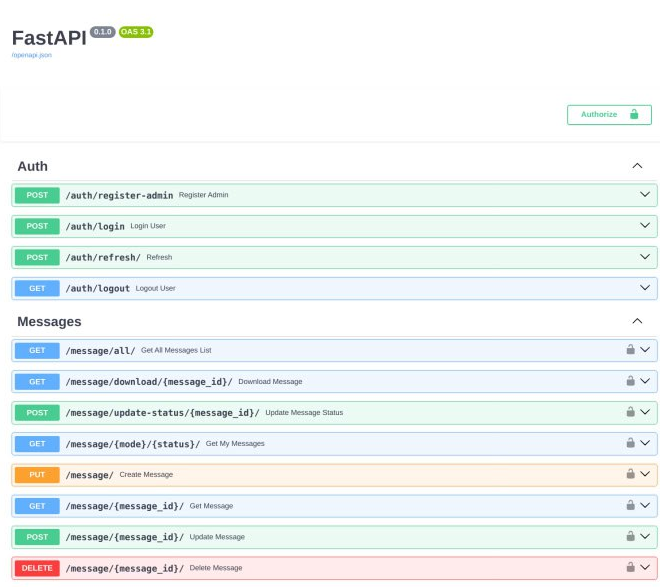


Рисунок 4.1 – Страница документации

Данная страница документации позволяет тестировать конечные точки сервера с разными параметрами и предоставляет удобный интерфейс, возможность аутентификации при помощи различных способов (маркер доступа в cookie, маркер доступа в заголовке Authentication: Bearer, OAuth и многие другие).

Также документация предоставляет готовую curl-команду с параметрами, чтобы нужную конечную точку можно было протестировать в терминале.

Так, для установки cookie-файлов может использоваться встроенное окно Authorize, представленное на рисунке 4.2.

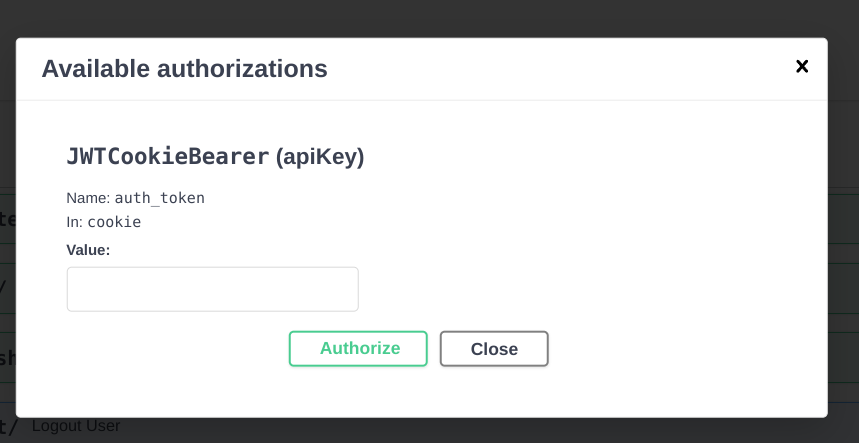


Рисунок 4.2 – Окно аутентификации

Для проверки каждой конечной точки предоставлено раскрывающееся меню, в которое можно ввести нужные параметры, отправить запрос и просмотреть результат в удобно читаемой человеком форме. Пример такого меню представлен на рисунке 4.3.

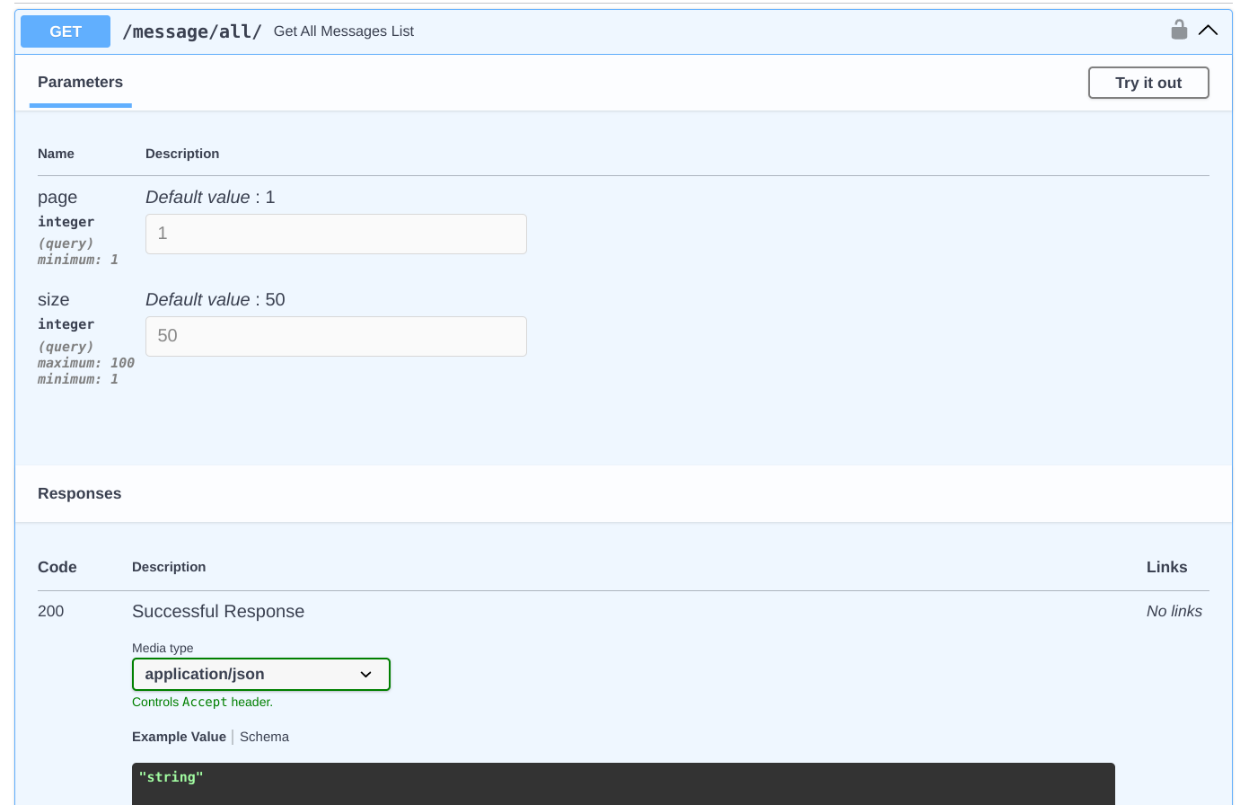


Рисунок 4.3 – Меню проверки конечной точки

Также данное меню предоставляет дополнительную информацию о конечной точке, такую как описание, статус (устаревшая или нет), примеры возвращаемых ответов для разных кодов статуса и прочее, что делает отладку удобнее.

5 Руководство пользователя

При первом открытии сайта пользователь видит страницу входа в учётную запись, представленную на рисунке 5.1.



Рисунок 5.1 – Страница входа в учётную запись

Пользователь должен ввести данные аутентификации и нажать на кнопку «Войти», и его перенаправит на страницу исходящих сообщений, представленную на рисунке 5.2. На данной странице представлены исходящие сообщения, каждое из которых содержит заголовок, текст, имена отправителя и получателя, а также время создания и изменения сообщения.



Рисунок 5.2 – Страница исходящих сообщений

Для изменения сообщения пользователь должен нажать на кнопку «Обновить» в карточке нужного сообщения, и его перенаправит на страницу, представленную на рисунке 5.3

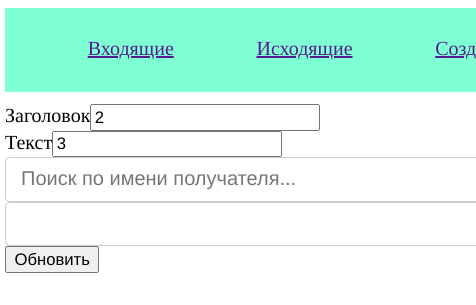


Рисунок 5.3 – Страница изменения сообщения

Пользователь должен ввести заголовок и текст сообщения, выбрать нужные файлы, а также выбрать получателя из выпадающего списка. Для повышения удобства поиска получателя реализован поиск по имени: при вводе текста в строку поиска в выпадающем списке останутся только те пользователи, чьё имя содержит введённую подстроку.

Для создания нового пользователя пользователь должен иметь роль администратора и перейти по ссылке «Создать пользователя». Страница создания пользователя представлена на рисунке 5.4.

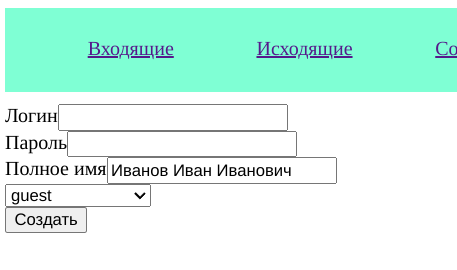


Рисунок 5.4 – Страница создания пользователя

Администратор должен ввести логин, пароль и имя пользователя, а также выбрать его роль из выпадающего списка. Для данного выпадающего списка поиск по названию не предусмотрен.

Также администратор может получать список всех зарегистрированных пользователей. Для этого он должен перейти по ссылке «Пользователи». Страница пользователей представлена на рисунке 5.5.



Рисунок 5.5 – Страница пользователей

На данной странице представлены все пользователи. Если пользователей больше, чем помещается на одну страницу, администратор может переключаться между страницами при помощи кнопок в нижней части страницы.

Для выхода из учётной записи пользователь должен перейти по ссылке «Выход». Сервер удалит пользовательские cookie-файлы, и при заходе на любую страницу сайта пользователь будет автоматически перенаправлен на страницу входа в учётную запись.

Заключение

При выполнении курсового проекта было создано приложение поиска и предложения работы. Сервер был создан при помощи платформы Node.js, языка программирования JavaScript и фреймворка h3. Web-сайт был реализован при помощи библиотеки React. База данных была реализована в СУБД PostgreSQL. Были реализованы все функциональные требования, а именно:

– обеспечение возможности регистрации и авторизации;

– поддержка ролей гостя, соискателя, работодателя и администратора;

– обеспечение возможности изменять образование, опыт работы и список ключевых навыков соискателя;

– предоставление возможности работодателю принимать и отклонять отклики;

− обеспечение возможности оставлять отзывы об исполнителе и работодателе;

− предоставление возможности отслеживать статус предложения о работе в реальном времени;

− обеспечение возможности соискателю откликаться на предложения о работе;

− предоставление возможности создавать и удалять учётные записи;

− обеспечение возможности фильтровать предложения по критериям (тип, необходимый опыт, оплата и так далее);

− предоставление возможности размещать и удалять вакансии и заказы на выполнение работ.

Также были реализованы WebSocket-сервер и поддержка протокола HTTPS. Приложение было протестировано на наличие ошибок с использованием ручного и автоматического тестирования. Для ручного тестирования использовался браузер и Postman. Для автоматического тестирования использовалось вспомогательное приложение, использующее функцию fetch.

По итогам тестирования были исправлены следующие ошибки в приложении: неверный формат отправляемых данных, отсутствие необходимых данных в ответе, ошибки в проверке авторизации.

Также было создано иллюстрированное руководство пользователя, в котором были описаны способы взаимодействия с приложением и наглядно продемонстрированы web-страницы приложения.

Список используемых источников

1. PostgreSQL Documentation [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.postgresql.org/docs/>.
2. FastAPI [Электронный ресурс]. – Электронные данные. – Режим доступа: https://fastapi.tiangolo.com.
3. Vue-paginate [Электронный ресурс]. – Электронные данные. – Режим доступа: https://svifty7.github.io/vue-paginate/.

Приложение А

Листинг определения моделей для ORM SQLAlchemy

|  |
| --- |
| import datetime import uuid from database import Base, Session from settings import AppConfig from sqlalchemy import (  LargeBinary,  ForeignKey,  Integer,  String,  UUID, ) from sqlalchemy.dialects.postgresql import ENUM from sqlalchemy.ext.mutable import MutableDict from sqlalchemy.orm import (  Mapped,  mapped\_column,  relationship )   class TimestampMixin:  created\_at: Mapped[datetime.datetime] = mapped\_column(  default=datetime.datetime.utcnow()  )  updated\_at: Mapped[datetime.datetime] = mapped\_column(  default=datetime.datetime.utcnow()  )   class DeletedMixin:  deleted\_at: Mapped[datetime.datetime] = mapped\_column(nullable=True)  deleted\_by: Mapped[str] = mapped\_column(  ForeignKey(f"users.id", ondelete="RESTRICT"),  nullable=True  )   class UserModel(TimestampMixin, DeletedMixin, Base):  \_\_table\_args\_\_ = {'extend\_existing': True}  \_\_tablename\_\_ = f"users"   id: Mapped[str] = mapped\_column(  UUID(as\_uuid=True),  primary\_key=True,  default=uuid.uuid4  )  full\_name: Mapped[str] = mapped\_column(  String(100),  nullable=False  )  login: Mapped[str] = mapped\_column(  String(30),  unique=True,  nullable=False  )  password\_hash: Mapped[str] = mapped\_column(  String(70),  nullable=False  )  role\_id: Mapped[int] = mapped\_column(  Integer(),  nullable=False  ) class MessageModel(TimestampMixin, DeletedMixin, Base):  \_\_table\_args\_\_ = {'extend\_existing': True}  \_\_tablename\_\_ = f"messages"   id: Mapped[str] = mapped\_column(  UUID(as\_uuid=True),  primary\_key=True,  default=uuid.uuid4  )  header: Mapped[str] = mapped\_column(  String(), nullable=False  )  message: Mapped[str] = mapped\_column(  String(), nullable=False  )  attachment: Mapped[bytes] = mapped\_column(  LargeBinary())  status: Mapped[str] = mapped\_column(  Integer(),  default=AppConfig.message\_statuses.sent.value,  nullable=False  )  sender: Mapped[str] = mapped\_column(  ForeignKey(f"users.id", ondelete="CASCADE"),  nullable=False  )  receiver: Mapped[str] = mapped\_column(  ForeignKey(f"users.id", ondelete="CASCADE"),  nullable=False  )   sender\_user: Mapped["UserModel"] = relationship(  "UserModel", foreign\_keys=[sender], backref="sent\_messages"  )   receiver\_user: Mapped["UserModel"] = relationship(  "UserModel", foreign\_keys=[receiver], backref="received\_messages"  ) |