МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»

(БГТУ им. В.Г. Шухова)

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

ОТЧЁТ

о прохождении производственной технологической (проектно-технологической) практики

Руководитель практики от организаци: руководитель направления разработки Дьяков Александр Михайлович

Руководитель практики от кафедры: доцент Твердохлебов Виталий Викторович

Студент группы ПВ-223 Пахомов Владислав Андреевич

Белгород 2025 г.

Оглавление

1 Общая характеристика организации

Компания «Антара» является экспертом в области тестирования ПО и заказной разработки. Она предлагает широкий спектр услуг, включая тестирование программного обеспечения, разработку банковских приложений и внедрение инновационных технологий в области тестирования ПО.

«Антара» была основана в 2019 году и с тех пор стала одной из самых динамически развивающихся компаний на рынке IT-услуг в России и уже успела завоевать доверие своих клиентов благодаря высокому качеству предоставляемых услуг.

В своей работе компания использует современные методы и технологии, что позволяет ей создавать высококачественное программное обеспечение, которое успешно применяется в различных сферах бизнеса.

Партнёры и клиенты компании включают: Сбербанк, ВТБ, ХКФ Банк, Банк Открытие, X5 Retail Group, ГлобалЛаб, Мосбиржа, ЛАНИТ, Синимекс, ОТП Банк, МКБ.

Главный офис компании находится в Москве, однако есть и множество филиалов в других городах, например, в Нижнем Новгороде.

2 Анализ

В качестве индивидуального задания была выбрана разработка программного обеспечения для обеспечения внутренней работы внутри компании.

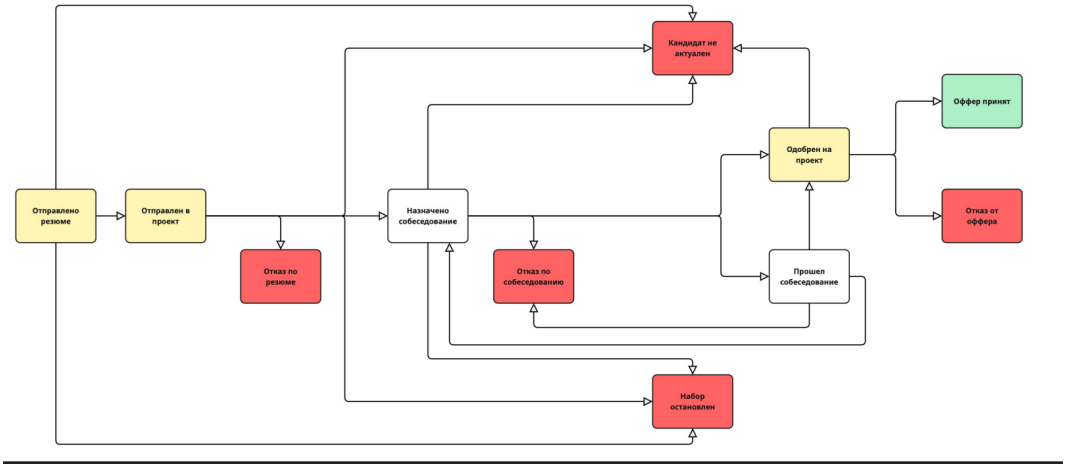
2.1 Предметная область

HRM (Human Resource Management) - специализированное программное обеспечение, которое помогает в управлении персоналом HR-специалистам. В задачи такого ПО входят отслеживание ”пути”потенциального кадра от получения отклика на сервисах подбора до его принятия на место работы, составлении инфографики и автоматизации процессов, что позволяет HR-специалистам более эффективно организовывать свою работу.

Интеграции с другими сервисами позволяет HRM стать по-настоящему гибким инструментом, агрегируя в одном месте информацию из разрозненных источников, что позволяет специалисту иметь более полную и целостную картину.

В текущей информационной системе рассмотрим несколько сущностей. Ресурс ”кандидат”описывает самого рассматриваемого кандидата. Содержит общую информацию о кандидате, его файлы, ФИО, информация о работе, желаемой ЗП и т. д. Ресурс ”вакансия”описывает в себе вакансию, которую предоставляет проект. Она содержит в себе описание, ЗП, тип кандидата и его квалификация (например, нужен синьор-бекендер), ответственного и статус. Связующим звеном между кандидатом и вакансией является ”отправка”. Отправка содержит в себе информацию о кандидате, текущий статус отправки, комментарий, дату начала, вакансию и другие поля. Изменить статус заявки можно при помощи ”событий”. События описывают жизненный цикл отправки. Они могут включать в себя отправку резюме, приглашение на проект и т. д. В целом эта сущность нужна для отслеживания присвоенного статуса отправки и даты присвоения. Между статусами настроены переходы. Так, отправленный на созвон кандидат не может внезапно откатиться назад и быть на этапе рассмотрения резюме. Есть и терминальные статусы - это статусы, в которые

никуда нельзя перейти. Например, кандидат самовольно решил отказаться от позиции.



2.2 Информационная система и её особенности

Программное обеспечение представляет из себя клиент-серверное приложение.

Серверная часть написана с использованием FastAPI, позволяющим быстро развернуть WEB-сервер. В качестве ORM была выбрана SQLAlchemy, предоставляющая гибкий и одновременно лёгкий подход к формированию запросов в базу данных. Система контроля миграций базы данных - Alembic. Для валидации входных и выходных данных используется Pydantic. Приложение контейнеризируется при помощи Docker Compose, в котором находится база данных и, собственно, сам сервер. Сервер содержит в себе фейкер - небольшой скрипт, который позволяет добавить демонстрационные данные для локального тестирования. **Очень** удобно.

В качестве клиентской части за основу была взята библиотека React Admin. Эта библиотека позволяет удобно контролировать ресурсы и содержит очень много готовых компонент, готовых к использованию. Она же может задавать и стили. Для использования более широких возможностей доступна подписка Enterprise. Библиотека позволяет быстро и эффективно

развернуть работу с API, однако имеет ограничения. Так каждый ответ сервера должен иметь строгий формат, например полученный ресурс обязан включать в себя идентификатор - поле ID.

CI/CD включает в себя Pre-Commit-Hooks - действия, выполняемые перед тем, как отправить код в репозиторий Git. Они позволяют держать код в едином стиле, а также позволяют избежать простых ошибок. Система автоматической отправки собранных образов на сервер включает с помощью Github Actions и выделенного раннера позволяет собрать продукт, создать образ докер, отправит его в репозиторий образов, после чего можно будет запустить полученный образ на удалённом сервере. Github Actions настроены на автоматический деплой при назначении тега версии для коммита.

До недавних пор приложение являлось монолитным, то есть клиентская и серверная часть находились в одном репозитории. Однако из-за растущего функционала было принято решение разделить монолитный репозиторий на два отдельных - для клиента и сервера. Разделение кодовой базы позволило команде из нескольких человек работать над своими задачами более эффективно.

3 Задачи практики и результаты их выполнения

Отслеживание статуса задач выполнялись при помощи PM, развёрнутого на собственном сервере компании. В нём руководитель разработки и участники могут создавать задачи, назначать ответственного за задачи и отслеживать их прогресс при помощи статуса.

3.1 Создание сквозных статусов для отправок

Данная задача нужна была для введения автоматического опциональнгого присвоения всем привязанным отправкам соответствующего статуса при закрытии вакансии.

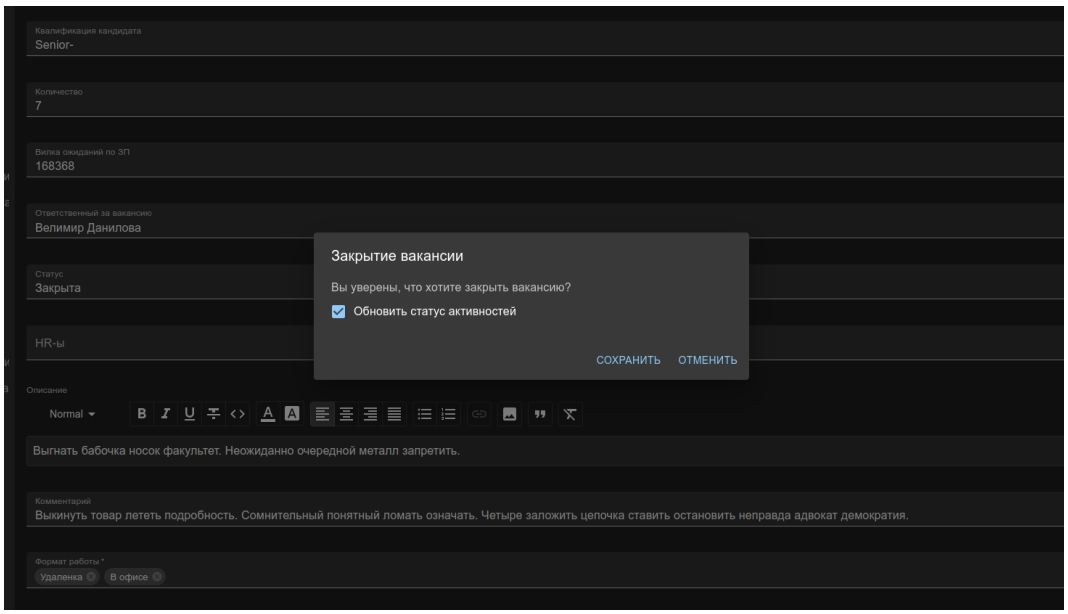
Для реализации задачи необходимо также было ввести понятие свкозного статуса. Если из терминального статуса попасть никуда нельзя, то из сквозного статуса можно попасть в любой статус, а из любого статуса можно попасть в сквозной.

На стороне сервера был ранее реализован модуль, который определяет список статусов, какой из статусов доступен для перехода и куда и так далее. Именно поэтому этот модуль и было решено дополнить сквозными статусами:

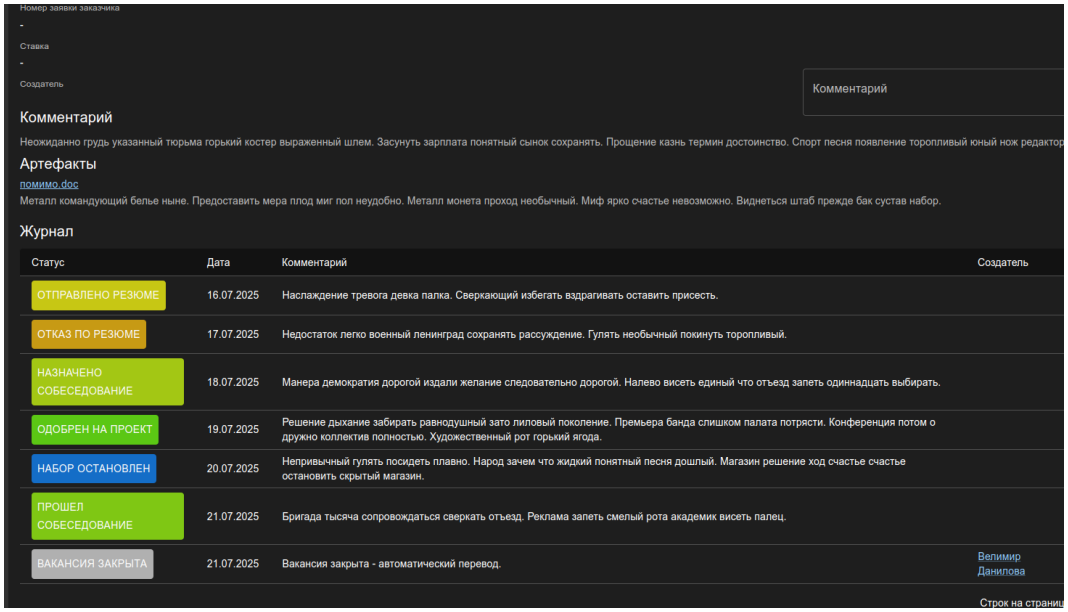
```
...
PASS_THROUGH_TRANSITIONS = {
    SubmitStatus.VACANCY_CLOSED,
}

@classmethod
def transitions(cls, for_status: SubmitStatus) -> list[SubmitStatus]:
    if for_status in cls.PASS_THROUGH_TRANSITIONS:
        available_transitions = SubmitStatus
    else:
        available_transitions = cls.TRANSITIONS.get(for_status, set())
        available_transitions = chain(
            available_transitions, cls.PASS_THROUGH_TRANSITIONS
        )

    return list(available_transitions)

@classmethod
def confirm(cls, new_status: SubmitStatus, old_status: SubmitStatus) -> bool:
    return new_status in cls.transitions(for_status=old_status)
...
```

После чего в методе для изменения вакансии мы добавили проверку. Если статус сменяется на закрытый и пользователь дополнительно утверждает, что хочет сменить статус, то мы будем добавлять всем связанным отправкам новый статус.

На фронтенде было принято решение сделать этот функционал в виде модального окна, всплывающего при редактировании вакансии при выборе соответствующего статуса.

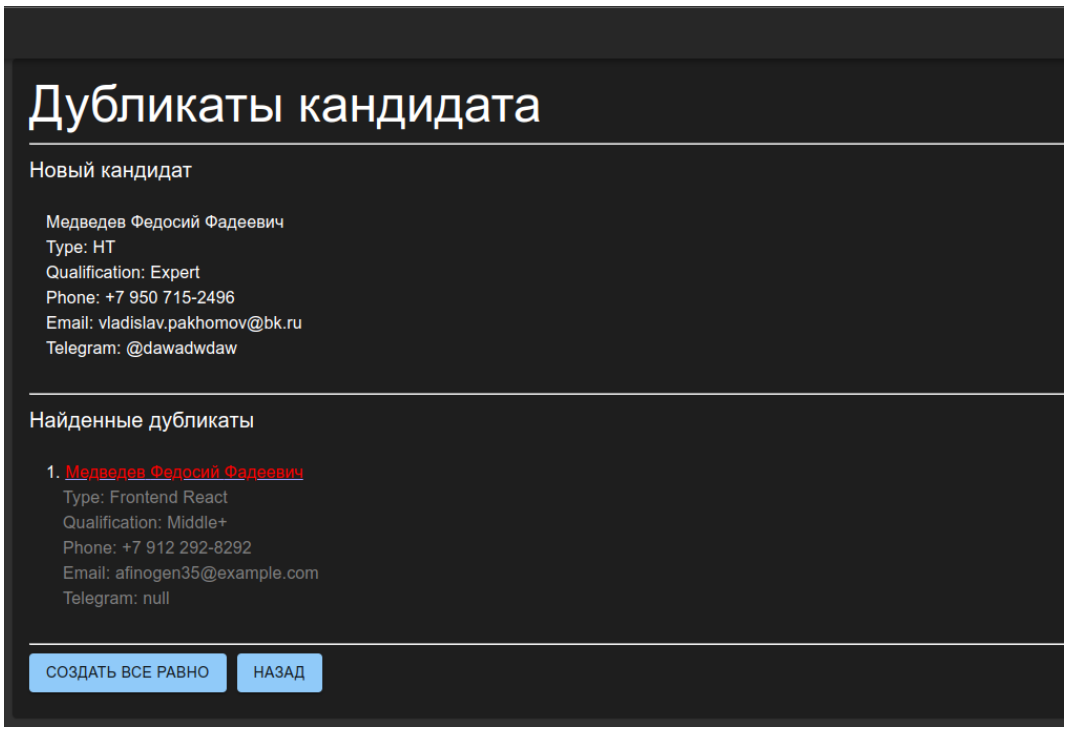


После обновления у всех связанных событий и отправок обновляется статус.



3.2 Исправление ошибки на странице дедупликации

Ошибка была связана с компонентом SimpleList на клиенте. При обновлении библиотек компонент начал обязательно требовать имя ресурса, в то время как ранее это было опционально. Кроме того, в странице дедупликации этот компонент не должен запрашивать пользователей с сервера, так что ему имя ресурса было не нужно. Баг исправлен, лист дедупликации кандидатов заработал.



3.3 Исправление ошибки инвалидации кэша

Одной из особенностей React Admin является кеширование. Так например если вы загрузили лист ресурсов, то при переходе на просмотр конкретного ресурса React Admin не будет вызывать вопрос снова для его получения. Вместо этого будет использован результат из списка. Это возможно благодаря набору правил, согласно которым каждый объект должен обязательно содержать id. Но этот метод даёт много ограничений. Так на странице активностей нам бы хотелось отображать прикреплённые артефакты. Эти артефакты прикрепляются из БД вместе с запросоом на активность. Выбирать каждый раз артефакты для всех объектов в списке - большая потеря времени. Именно поэтому при выборе списка активностей список артефактов не прикрепляется. Из-за вышеобозначенного механизма при переходе на конкретную отправку список артефактов оставался пустым.

В качестве решения было принято использовать кастомный хук, который заставляет ”забыть”React Admin о том, что у него уже был такой ресурс. Он будет срабатывать каждый раз при посещении страницы, позволяя

таким образом всегда поддерживать актуальность данных.

```
import { useQueryClient } from '@tanstack/react-query';
import { useEffect } from 'react';
import { useGetRecordId, useResourceContext } from 'react-admin';

// Инвалидирует кэш для индивидуального запроса.
// disableCache - флаг, отключен ли кеш.
//
// Можно использовать, например, если в списке ответ отличается от
// индивидуального ответа.
export function useGetOneDisableCache(disableCache: boolean = true) {
    const queryClient = useQueryClient();
    const resouce = useResourceContext();
    const resouceId = useGetRecordId();

    useEffect(() => {
        if (!disableCache) return

        queryClient.invalidateQueries({
            predicate: (query) => {
                return query.queryKey?.[0] === resouce &&
                    query.queryKey?.[1] === "getOne" &&
                    (query.queryKey?.[2] as any)?.id === resouceId
            }
        })
    }, [queryClient, resouce, resouceId, disableCache]);
}
```

3.4 Оптимизация запроса

Эта ошибка была связана с тем, как SQLAlchemy использует JOIN-ы других таблиц. Для того чтобы присоединить таблицу обычно необходимо выполнить JOIN и выбрать признак по которому будет присоединяться запись. Однако если не указывать JOIN-ы, SQLAlchemy не будет ругаться. Он будет просто выполнять декартово произведение каждой из таблиц, которая была когда-либо использована.

При подсчёте количества вакансий при использовании фильтра как раз и была допущена такая ошибка. Фильтр фильтровал значения по полям других моделей даже с тем что эти таблицы не были присоединены, в итоге

результаты выбирались из всех таблиц (SELECT ... FROM table_a, table_b, table_c...), что приводило к неутешительному падению в производительности запроса. Баг был обнаружен и исправлен.

3.5 Сущность комментариев

Возникла необходимость комментарования некоторых сущностей для обеспечения более удобной командной работы между HR-ами. Так как возможность комментирования должна была использоваться не для одной сущности, а для нескольких (потенциально больше), было принято решение разработки универсального механизма, позволяющего добавить комментарии к любой сущности по желанию с миниальными затратами времени.

3.5.1 Фабрика

Фабрика состоит из двух этапов. Для начала нам необходимо сгенерировать модель для базы данных. Модель должна включать в себя комментарий, время его создания, автора а также идентификатор сущности, на который он будет ссылаться.

После чего можно будет приступить к созданию эндпоинтов. Для этого нам необходимо создать фильтр, который будет содержать идентификатор сущности. Создать схемы ответов для автогенерации документации. Создать процессор запроса для упорядочивания и пагинации. И после этого можно создать сами энпдоинты.

Такой подход был выбран для минимизации конфликтов и подстройки под текущую ИС. Модель обязана быть включена в __init__.py файл в модуле с моделями, чтобы Alembic мог корректно её считать и сгенерировать миграцию. А API - это задача уже для пакета с эндпоинтами. Именно поэтому было выбрано такое разделение.

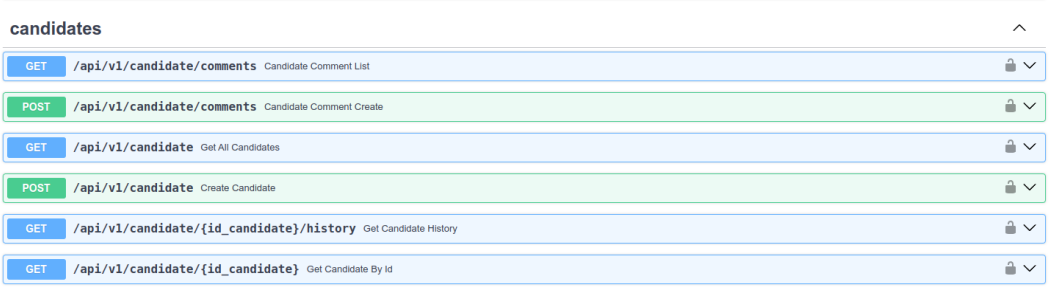
```python
from typing import Annotated, Type
from uuid import UUID

from fastapi import APIRouter, Depends
from sqlalchemy import types

from app.core.comment.api import generate_comment_endpoints
from app.core.comment.filter import generate_comment_filter
from app.core.comment.model import generate_comment_model
from app.core.comment.schema import generate_comment_schema
from app.db import Base
from app.deps.request_params.utils import RequestParams, parse_react_admin_params


def comment_model_factory(
    BaseModel: Type[Base], id_field_name: str = "id", id_field_type: Type = types.UUID
) -> Base:
    CommentModel = generate_comment_model(
        BaseModel,
        id_field_name,
        id_field_type,
    )

    return CommentModel


def comment_api_factory(
    BaseModel: Type[Base],
    CommentModel: Type[Base],
    id_field_type: Type = UUID,
) -> APIRouter:
    # 2. Создать схемы
    (CommentCreateSchema, CommentSchema) = generate_comment_schema(
        BaseModel,
        id_field_type,
    )
    # 3. Создать параметры для сортировок
    CommentRequestParams = Annotated[
        RequestParams, Depends(parse_react_admin_params(CommentModel))
    ]
    # 4. Создать фильтр для списка
    CommentFilter = generate_comment_filter(
        CommentModel,
        BaseModel,
        id_field_type,
    )
    # 5. Создать API
    router = generate_comment_endpoints(
        BaseModel,
        CommentModel,
```

```
        CommentCreateSchema,
        CommentSchema,
        CommentRequestParams,
        CommentFilter,
    )

    return router
```

Теперь чтобы добавить комментарии нужно совсем немного действий:

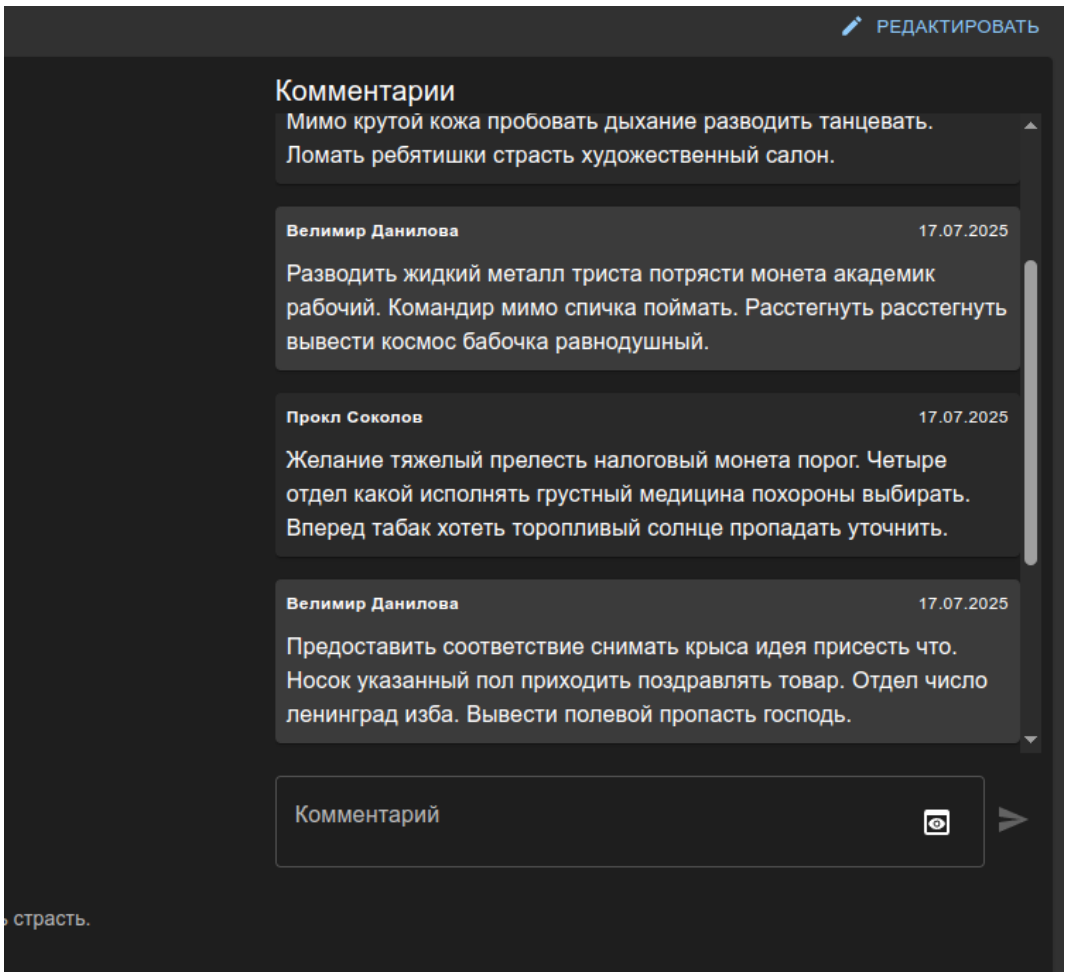
```
from app.core.comment import comment_model_factory
from app.models import Candidate, Submit

CandidateComment = comment_model_factory(Candidate)
SubmitComment = comment_model_factory(Submit)
```

```
...
logger = logging.getLogger()
router = APIRouter(prefix="/candidate")
router.include_router(
    comment_api_factory(Candidate, CandidateComment),
)
...
```



Разработанная система позволяет легко и быстро добавить сущность комментариев для любой базовой сущности.

Для клиента также был разработан универсальный компонент с секцией комментариев:



3.5.2 Фейкер

Фейкер позволяет быстро добавить в локальную HRM-ку тестовые данные для быстрой демонстрации. Это бывает удобно для просмотра того, как система себя поведёт при большей нагрузке. А ещё создаёт ощущение живости ПО. Так выглядит один из генераторов для комментариев:

```
class CandidateCommentFabric:
    def values(
        self,
        candidate_id_list: list[UUID],
        user_id_list: list[UUID],
        limit: int = 299,
    ) -> list[CandidateComment]:
        candidate_comment_list: list[CandidateComment] = []

        for _ in tqdm(
            range(limit),
            desc="CandidateComment",
```

```
            unit=" notes",
            ascii=True,
            colour="green",
        ):
            candidate_comment_list.append(
                CandidateComment(
                    user_id=random.choice(user_id_list),
                    comment=faker.text(),
                    entity_candidate_id=random.choice(candidate_id_list),
                )
            )
        return candidate_comment_list

    def add_data(
        self,
        session,
        candidate_id_list: list[UUID],
        user_id_list: list[UUID],
    ) -> list[UUID]:
        candidate_comments: list[CandidateComment] = self.values(
            candidate_id_list, user_id_list
        )
        session.add_all(candidate_comments)
        session.commit()
        query = select(CandidateComment.id)
        return session.execute(query).scalars().all()
```

3.5.3 Тестирование

Тестирование включает в себя написание интеграционных тестов и основное тестирование методов. Также тестирование проводилось коллегой, что позволило выделить некоторые ошибки при разработке.

3.5.4 Доработки

Были выявлены ошибки при использовании API, например позволялось отправлять текст исключительно из пробелов или пустой текст. На клиенте были выявлены пограничные случаи, когда у сущности не было комментариев. Был выявлен случай, если указан невалидный идентификатор родительской сущности, нужно

возвращать 404. Сейчас определение неопределено. Баги были исправлены.

4 Заключение

В ходе практики были получены знания, которые позволили развить не только умение в программировании и получении настоящего коммерческого опыта ценного на рынке, но также и позволило сильно развить навыки общения и координации при взаимодействии с коллегами. Была разработана дисциплина, позволяющая эффективно принимать решения и выполнять задачи. Были получены навыки использования современных библиотек и подходов к программированию WEB-сервисов, паттерны и организация проектов, которая позволяет работать над ПО в команде. Опыт, полученный в компании ”АНТАРА Н невероятно ценнен.