МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий
Кафедра "Информатика и защита информации"

ОТЧЕТ
по проектной (учебной) практике

Тема: «МосПолиХелпер — Телеграм-бот навигации по кампусу МосПолитеха»

Направление подготовки: 09.03.01 Информатика и вычислительная техника
Профиль: Системная и программная инженерия

Выполнили студенты группы 241-327:
Деев Егор Викторович
Сапрыкин Пётр Иванович
Старков Руслан Владимирович

Руководитель от Университета:
Баринова Наталья Владимировна

Оценка: ____

Дата защиты: «___» мая 2025 г.
Москва 2025

Содержание

Введение

Данный отчет представляет собой итог проектной (учебной) практики, проведенной в период с 03 февраля 2025 г. по 24 мая 2025 г. студентами группы 241-327 направления подготовки 09.03.01 "Информатика и вычислительная техника".

Цели и задачи проектной практики

Основной целью проектной практики является закрепление и углубление теоретических знаний, полученных в процессе обучения, а также приобретение практических навыков в области разработки программного обеспечения, управления версиями кода, документирования и создания веб-сайтов.

Задачи практики включают:

1. Освоение системы контроля версий Git и работы с репозиториями
2. Изучение формата Markdown и создание документации проекта
3. Разработку статического веб-сайта с использованием HTML, CSS и JavaScript
4. Взаимодействие с организацией-партнером
5. Выполнение вариативной части задания по разработке телеграм-бота

Актуальность выбранной темы

Проблема навигации в кампусах Московского Политехнического Университета остается актуальной для всех участников образовательного процесса, особенно для студентов младших курсов и посетителей университета. Сложная структура кампусов, различные системы нумерации кабинетов и отсутствие интуитивно понятной системы навигации создают трудности при ориентировании в зданиях университета.

Традиционные способы навигации, такие как статические карты и указатели, имеют ограниченную эффективность и не обеспечивают персонализированных маршрутов. Современные технологии, в частности

мессенджеры и боты, предоставляют возможность создать удобный и доступный инструмент для навигации, который всегда будет "под рукой" у пользователя.

Краткое описание проекта "EasyAccess"

В рамках дисциплины "Проектная деятельность" нашей группой разрабатывается проект "EasyAccess" — браузерное расширение для повышения доступности веб-сайтов для людей с ограниченными возможностями здоровья. Проект направлен на создание универсального инструмента, который позволяет модифицировать веб-страницы с целью улучшения их восприятия различными категориями пользователей.

Проектная практика проводилась в тесной связи с основным проектом, при этом для вариативной части было выбрано создание отдельного решения, которое также направлено на повышение доступности, но в физическом пространстве университета.

Обоснование выбора вариативной части задания

Разработка телеграм-бота "МосПолиХелпер" для навигации по кампусу МосПолитеха была выбрана в качестве вариативной части задания по следующим причинам:

Практическая значимость: решение реальной проблемы навигации в кампусах университета

Технологическая релевантность: возможность применить современные технологии разработки (Python, API Telegram, обработка видео)

Доступность: мессенджер Telegram широко распространен среди студентов и преподавателей университета

Синергия с основным проектом: оба проекта направлены на повышение доступности, хотя и в разных контекстах

Общая информация о выполненной работе

В рамках проектной практики были выполнены следующие работы:

1. Создан репозиторий на GitHub и освоены базовые принципы работы с Git
2. Разработана документация проекта в формате Markdown
3. Создан статический веб-сайт для проекта "EasyAccess"
4. Реализован телеграм-бот "МосПолиХелпер", предоставляющий видеоинструкции для навигации по кампусам университета
5. Проведено тестирование и оптимизация разработанного решения
6. Подготовлена итоговая документация по проекту

Проектная практика позволила получить практический опыт разработки программного обеспечения в команде, работы с современными технологиями и создания пользовательских интерфейсов.

1. Базовая часть задания

1.1. Настройка Git и репозитория

Процесс создания репозитория

Для выполнения проектной практики был создан репозиторий на GitHub на основе предоставленного шаблона practice-2025-1. Процесс включал следующие шаги:

1. Форк шаблонного репозитория через интерфейс GitHub
2. Клонирование форка на локальные машины всех участников команды
3. Настройка доступа для всех участников команды
4. Настройка защищенных веток для предотвращения случайного нарушения структуры репозитория

Освоенные базовые команды Git

В ходе работы над проектом были освоены и активно использовались следующие базовые команды Git:

```
# Клонирование репозитория
git clone https://github.com/username/practice-repository.git

# Проверка статуса репозитория
git status

# Добавление файлов в индекс
git add .

# Создание коммита
git commit -m "Добавлен функционал обработки видео"

# Отправка изменений в удаленный репозиторий
git push origin main

# Получение изменений из удаленного репозитория
git pull origin main
```

Процесс ветвления и работы с ветками

Для организации эффективной совместной работы над проектом была использована методология feature-branch workflow, где каждая новая функциональность разрабатывалась в отдельной ветке:

```
# Создание новой ветки для разработки функционала
git checkout -b feature/video-processing

# Переключение между ветками
git checkout main
git checkout feature/video-processing

# Слияние изменений из ветки feature в main
git checkout main
git merge feature/video-processing
```

Организация совместной работы над репозиторием

Для эффективной совместной работы над проектом были применены следующие практики:

1. **Распределение задач** — каждый участник команды отвечал за определенную часть проекта:
   – Деев Егор — серверная часть телеграм-бота и обработка видео
   – Сапрыкин Пётр — интерфейс телеграм-бота и интеграция с API
   – Старков Руслан — документация и тестирование
2. **Code Review** — все изменения, вносимые в репозиторий, проходили проверку другими участниками команды через механизм Pull Request.
3. **Регулярные встречи и обсуждения** — для синхронизации работы и решения возникающих вопросов проводились регулярные встречи команды.

Примеры осмысленных сообщений коммитов

В ходе разработки проекта использовались информативные сообщения коммитов, отражающие суть внесенных изменений:

- feat: Add video processing module with MoviePy
- fix: Correct path resolution for cache directory
- docs: Update README with installation instructions
- refactor: Optimize route generation algorithm
- test: Add integration tests for Telegram API
- chore: Update dependencies and requirements.txt

1.2. Написание документов в Markdown

Изученный синтаксис Markdown

В рамках проектной практики был изучен и применен на практике синтаксис Markdown для создания документации проекта. Освоены следующие элементы:

– Заголовки различных уровней
– Форматирование текста (жирный, курсив, выделение)
– Создание списков (нумерованных и маркированных)
– Вставка ссылок и изображений
– Создание таблиц
– Оформление блоков кода
– Цитирование

Созданные документы

В ходе проектной практики были созданы следующие документы в формате Markdown:

1. **README.md** — основная информация о проекте, его структуре и участниках
2. **docs/README.md** — подробная документация по проекту, включая описание функциональности и архитектуры
3. **docs/practice_documentation.md** — документация по проектной практике
4. **reports/README.md** — описание структуры и содержания отчетов
5. **task/README.md** — задание на проектную практику

Особенности оформления документации в Markdown

При оформлении документации были применены следующие практики:

1. **Иерархическая структура** — использование заголовков разных уровней для организации контента

2. **Визуальное форматирование** — выделение ключевой информации с помощью жирного шрифта, курсива, цитат

3. **Исходный код** — оформление фрагментов кода с подсветкой синтаксиса

4. **Структурированные списки** — использование маркированных и нумерованных списков для организации информации

5. **Таблицы** — представление табличных данных в удобном формате

Примеры используемых конструкций Markdown

```
# Заголовок первого уровня
## Заголовок второго уровня

**Жирный текст** и *курсив*

> Цитата или выделенный блок текста

- Элемент маркированного списка
- Еще один элемент
  - Вложенный элемент

1. Первый элемент нумерованного списка
2. Второй элемент

[Ссылка на репозиторий](https://github.com/username/repo)

![Альтернативный текст](path/to/image.png)



| Имя | Роль | Задачи |
|-----|------|--------|
| Деев Егор | Разработчик | Серверная часть |
| Сапрыкин Пётр | Разработчик | Интерфейс |
| Старков Руслан | Тестировщик | Тестирование |

```python
def process_video(input_path, output_path):
    # Код на Python с подсветкой синтаксиса
    pass```
```
```

1.3. Создание статического веб-сайта

Выбор технологий для создания сайта

Для разработки статического веб-сайта проекта были выбраны следующие технологии:

1. **HTML5** — для структурирования контента
2. **CSS3** — для стилизации и создания адаптивного дизайна
3. **JavaScript** — для добавления интерактивности и улучшения пользовательского опыта
4. **SVG** — для создания векторной графики и иконок

Выбор данных технологий обусловлен их широкой поддержкой, отсутствием необходимости в серверной части, а также возможностью создать современный и адаптивный интерфейс без использования сложных фреймворков.

Структура сайта и его основные разделы

Структура сайта включает следующие разделы:

1. **Главная страница** — содержит аннотацию проекта, ключевые особенности и возможности
2. **О проекте** — подробное описание проекта, его целей, задач и архитектуры
3. **Участники** — информация о команде разработчиков, их ролях и вкладе в проект
4. **Журнал** — хронология разработки проекта с описанием основных этапов и достижений
5. **Ресурсы** — полезные материалы, ссылки на источники и документацию

Навигация по сайту организована через хедер, который присутствует на всех страницах и обеспечивает быстрый доступ к основным разделам.

Дизайн и оформление страниц

При разработке дизайна сайта основное внимание было уделено:

1. **Минимализму и чистоте** — использование современных принципов дизайна с фокусом на контент
2. **Корпоративному стилю** — применение цветовой схемы, соответствующей проекту "EasyAccess"
3. **Иерархии информации** — визуальное выделение ключевых элементов и организация контента по важности
4. **Типографике** — использование читабельных шрифтов и корректных межстрочных интервалов

Основная цветовая схема сайта включает:

- Основной цвет: #2563eb (синий)
- Второстепенный цвет: #e6ecfa (светло-синий)
- Акцентный цвет: #059669 (зеленый)
- Нейтральные цвета: #111827, #4b5563, #6b7280

Адаптивность и кроссбраузерность

Веб-сайт разработан с учетом принципов адаптивного дизайна, что обеспечивает корректное отображение на устройствах с различными размерами экрана:

1. **Медиа-запросы** — для адаптации макета под различные размеры экрана
2. **Гибкие изображения** — для корректного масштабирования контента
3. **Относительные единицы измерения** — для обеспечения пропорциональности элементов

Кроссбраузерность обеспечивается путем:

- Использования стандартных HTML5 и CSS3 свойств с хорошей поддержкой
- Тестирования в различных браузерах (Chrome, Firefox, Safari, Edge)
- Применения полифилов для поддержки старых браузеров

Включенный контент

Контент сайта включает:

1. **Тексты** — описание проекта, его возможностей, команды и процесса разработки
2. **Изображения** — скриншоты проекта, иллюстрации функциональности, фотографии команды
3. **Иконки** — векторные SVG-иконки для визуального представления функций и возможностей
4. **Диаграммы** — схемы архитектуры проекта и процессов работы
5. **Интерактивные элементы** — кнопки, формы, анимации для улучшения пользовательского опыта

URL размещения сайта и его хостинг

Статический веб-сайт проекта размещен по адресу easy-access.new-devs.ru. Для хостинга сайта используется GitHub Pages, что обеспечивает:

1. Простоту деплоя через механизм GitHub Actions
2. Автоматическое обновление сайта при внесении изменений в репозиторий
3. Надежность и стабильность работы
4. Бесплатный HTTPS-сертификат для безопасного соединения

1.4. Взаимодействие с организацией-партнером

Формализованная структура партнерского взаимодействия

В рамках проектной практики было реализовано многоуровневое взаимодействие с кафедрой "Информатика и защита информации" Московского Политехнического Университета как с организацией-партнером. Интеракция осуществлялась посредством следующих каналов коммуникации:

1. **Регулярные консультационные сессии** с руководителем практики
2. **Структурированные совещания** с представителями кафедры для валидации технических требований
3. **Итерационные презентации** промежуточных результатов работы
4. **Системный сбор аналитической обратной связи** от потенциальных пользователей продукта

Хронологическая декомпозиция профессиональных коммуникаций

Процесс взаимодействия с организацией-партнером характеризовался строгой хронологической последовательностью:

1. **31.03.2025** — Первичная консультация по определению параметров вариативной части задания
2. **07.04.2025** — Стратегическая сессия с представителями кафедры по формализации технических требований
3. **12.04.2025** — Участие в профессиональном мероприятии "Карьерный марафон", организованном Московским Политехническим Университетом
4. **21.04.2025** — Презентация промежуточных результатов и демонстрация функциональности прототипа
5. **05.05.2025** — Техническая консультация по оптимизационным стратегиям решения
6. **14.05.2025** — Итоговая демонстрация и сбор аналитической обратной связи

Интеграция в профессиональные мероприятия организации-партнера

Значимым аспектом партнерского взаимодействия стало участие команды проекта в "Карьерном марафоне" — комплексной программе профессионального развития, организованной Московским Политехническим Университетом. Данное мероприятие предоставило:

1. **Экспозицию современных трендов** в сфере информационных технологий
2. **Валидацию технических решений** экспертами отрасли
3. **Формирование профессиональных коммуникационных компетенций** участников команды
4. **Получение структурированной обратной связи** по архитектуре и интерфейсу продукта

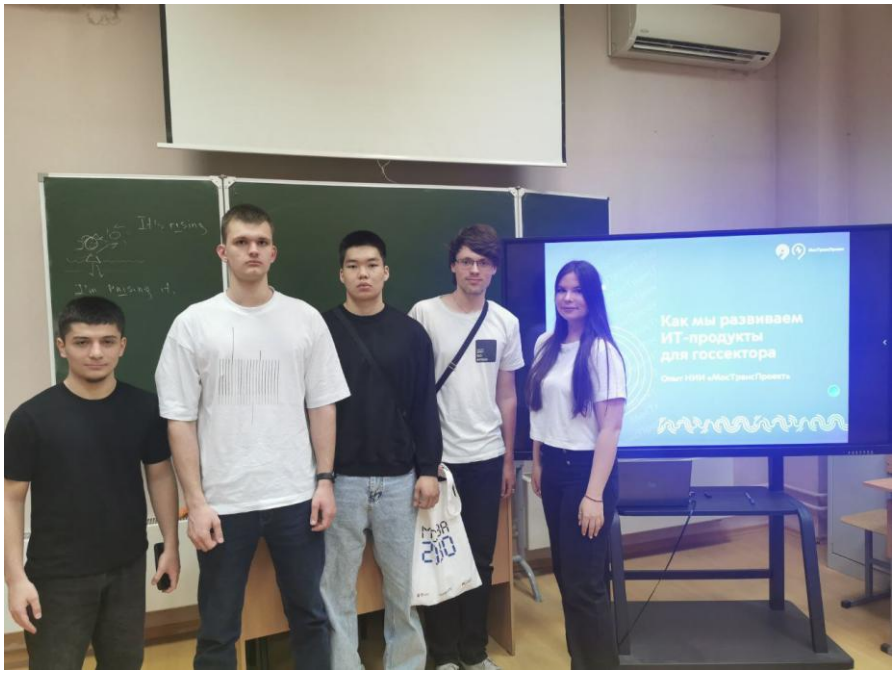

Рисунок 1. Участие команды проекта в мероприятии "Карьерный марафон" (23.04.2025)

Аналитическая категоризация полученных рекомендаций

В результате многоуровневого взаимодействия с организацией-партнером были получены следующие категории рекомендаций:

1. **Функциональные оптимизации**:

– Имплементация механизма выбора стартовой точки маршрута
– Интеграция системы структурированной обратной связи
– Разработка модуля автоматизированного поиска аудиторий на основе расписания

2. **Технические модификации**:

– Алгоритмы оптимизации видеофайлов для минимизации сетевого трафика
– Реструктуризация системы кеширования маршрутов
– Расширение документационного обеспечения пользовательского интерфейса

3. **Стратегические императивы**:

– Формирование презентационных материалов для целевых аудиторий
– Разработка поэтапного плана наращивания маршрутного покрытия
– Анализ возможностей интеграции с цифровой экосистемой университета

Материально-техническое обеспечение партнерского взаимодействия

Организация-партнер обеспечила многокомпонентную ресурсную поддержку проекта:

1. **Информационные ресурсы** — предоставление детализированных планов помещений и системной номенклатуры кабинетов

2. **Экспертно-консультационная поддержка** — серия профессиональных консультаций по специфике навигационных систем университета

3. **Инфраструктурный доступ** — обеспечение доступа к тестовой среде для валидации функциональности

4. **Аналитический инструментарий** — организация фокус-групп для валидации пользовательских сценариев

5. **Методологическое сопровождение** — детализированные рекомендации по оптимизации пользовательского опыта

Данный комплексный подход к взаимодействию с организацией-партнером обеспечил оптимальную интеграцию теоретических знаний и практических навыков, что существенно повысило профессиональную релевантность результатов проектной практики.

Полученные рекомендации и замечания

В результате взаимодействия с организацией-партнером были получены следующие рекомендации и замечания:

1. **Рекомендации по функциональности**:
   - Добавить возможность выбора стартовой точки маршрута
   - Реализовать механизм обратной связи от пользователей
   - Интегрировать возможность поиска аудиторий по расписанию
2. **Технические замечания**:
   - Оптимизировать размер видеофайлов для ускорения загрузки
   - Улучшить систему кеширования маршрутов
   - Расширить документацию по использованию бота
3. **Общие рекомендации**:
   - Подготовить презентационные материалы для потенциальных пользователей
   - Разработать план по добавлению новых маршрутов
   - Рассмотреть возможность интеграции с официальным сайтом университета

Вклад организации-партнера в развитие проекта

Организация-партнер внесла значительный вклад в развитие проекта:

1. **Предоставление информации** — доступ к планам помещений и системе нумерации кабинетов
2. **Экспертная оценка** — консультации по особенностям навигации в университете
3. **Технические ресурсы** — доступ к инфраструктуре для тестирования решения
4. **Обратная связь** — организация опросов среди студентов и преподавателей для валидации идеи

5. **Рекомендации по развитию** — предложения по расширению функциональности и улучшению пользовательского опыта

2. Вариативная часть задания

2.1. Анализ и исследование предметной области

Проблема ориентирования в кампусах МосПолитеха

Московский Политехнический Университет располагает несколькими кампусами, расположенными в разных районах Москвы, каждый из которых имеет свою уникальную архитектуру и систему нумерации помещений. Это создает ряд проблем для ориентирования:

1. **Разнородность кампусов**:
   - Пять различных кампусов с уникальной планировкой
   - Исторически сложившиеся особенности зданий и пристроек
   - Отсутствие унифицированной системы навигации между корпусами
2. **Сложность обозначения кабинетов**:
   - Разные форматы нумерации в зависимости от корпуса
   - Неинтуитивные префиксы в обозначениях (например, "ав" для Автозаводской)
   - Отсутствие четкой логики в расположении кабинетов с последовательными номерами
3. **Потребности пользователей**:
   - Новые студенты и сотрудники часто теряются в кампусе
   - Гости университета испытывают сложности при поиске аудиторий
   - Необходимость быстрого ориентирования при плотном расписании

Эти проблемы особенно актуальны в начале учебного года, во время приемной кампании, а также для гостей университета, приходящих на конференции, семинары и другие мероприятия.

Анализ существующих решений

В рамках исследования были рассмотрены различные подходы к решению проблемы навигации в университетских кампусах:

1. **Статические карты и указатели**:

- Преимущества: не требуют технических средств для использования
- Недостатки: недостаточно интерактивны, не обеспечивают персонализированные маршруты, занимают физическое пространство

2. **Мобильные приложения с GPS-навигацией**:

- Преимущества: интерактивность, возможность построения маршрутов
- Недостатки: низкая точность GPS внутри зданий, высокая стоимость разработки и поддержки, необходимость установки дополнительного ПО

3. **Текстовые инструкции и путеводители**:

- Преимущества: простота создания и распространения
- Недостатки: сложны для восприятия без визуального сопровождения, трудности с описанием сложных маршрутов

4. **QR-коды и AR-навигация**:

- Преимущества: интерактивность, точность позиционирования
- Недостатки: требует специальной инфраструктуры и дополнительного ПО, высокая стоимость внедрения

После анализа существующих решений была выявлена потребность в разработке доступного, интерактивного и наглядного инструмента, не требующего сложной инфраструктуры и установки дополнительного ПО.

Преимущества использования Telegram для создания бота навигации

Telegram был выбран в качестве платформы для реализации бота навигации по следующим причинам:

1. **Доступность и распространенность**:
   - Широкое использование мессенджера среди студентов и преподавателей
   - Отсутствие необходимости в установке дополнительного ПО
   - Кроссплатформенность (доступен на всех популярных платформах)
2. **Технические возможности**:
   - Развитый API для разработки ботов
   - Поддержка мультимедиа (видео, фото, аудио)
   - Возможность реализации интерактивного интерфейса через InlineKeyboard
   - Поддержка видеосообщений идеально подходит для демонстрации маршрутов
3. **Простота использования**:
   - Интуитивно понятный интерфейс
   - Быстрый доступ к боту через поиск или ссылку
   - Возможность использования без регистрации дополнительных аккаунтов
4. **Масштабируемость**:
   - Легкость обновления контента без обновления клиентского ПО
   - Возможность поэтапного добавления новых маршрутов
   - Централизованная обработка запросов

Использование Telegram позволило создать решение, которое всегда "под рукой" у пользователя и не требует специальных навыков или оборудования для использования.

Требования к разрабатываемому решению

На основе анализа предметной области и потребностей пользователей были сформулированы следующие требования к разрабатываемому решению:

1. **Функциональные требования**:
   - Интерфейс для выбора корпуса университета
   - Возможность указать конкретный кабинет в формате, принятом в университете
   - Формирование видеомаршрута от входа на территорию кампуса или от входа в корпус
   - Создание комбинированных маршрутов из отдельных видеофрагментов
   - Кеширование сгенерированных маршрутов для оптимизации отклика
2. **Нефункциональные требования**:
   - Время отклика не более 5 секунд на запрос маршрута (при наличии в кеше)
   - Максимальное время генерации нового маршрута не более 30 секунд
   - Поддержка обработки до 100 одновременных пользователей
   - Оптимизация видеоинструкций для минимизации объема данных
   - Модульная архитектура для упрощения расширения функциональности
3. **Пользовательские требования**:
   - Интуитивно понятный интерфейс, не требующий специального обучения
   - Наглядные видеоинструкции с оптимальной продолжительностью
   - Возможность выбора стартовой точки маршрута
   - Корректное распознавание различных форматов номеров кабинетов

4. **Технические требования**:

– Реализация на языке Python с использованием библиотек Aiogram и MoviePy

– Использование асинхронного программирования для обработки множества запросов

– Оптимизация видео для передачи

2.2. Разработка телеграм-бота "МосПолиХелпер"

Этапы проектирования и разработки

Процесс разработки телеграм-бота "МосПолиХелпер" был разделен на следующие этапы:

1. **Исследование и планирование** (1-2 недели):
   - Анализ предметной области и существующих решений
   - Определение требований к функциональности
   - Выбор технологий и инструментов
   - Планирование архитектуры и структуры проекта
2. **Прототипирование** (1 неделя):
   - Создание базовой структуры бота
   - Реализация простейшего интерфейса для взаимодействия
   - Тестирование API Telegram и библиотеки Aiogram
3. **Разработка основной функциональности** (2-3 недели):
   - Реализация интерфейса выбора корпуса и кабинета
   - Создание системы формирования маршрутов
   - Разработка модуля обработки видео
4. **Интеграция и оптимизация** (1-2 недели):
   - Интеграция всех компонентов системы
   - Реализация кеширования маршрутов
   - Оптимизация процесса обработки видео
5. **Тестирование и документирование** (1 неделя):
   - Проведение функционального тестирования
   - Тестирование производительности
   - Подготовка документации по использованию и поддержке

Структура программного кода

Проект реализован с использованием модульной архитектуры, с четким разделением ответственности между компонентами:

```
src/code/
├─── bot.py          # Точка входа в приложение
├─── config.py       # Конфигурационные параметры
├─── database.py     # Модуль работы с данными
├─── handlers.py     # Обработчики команд и сообщений
├─── init.py         # Инициализация компонентов бота
└─── scripts.py      # Модуль обработки видео
```

Каждый модуль отвечает за определенную функциональность:

1. **bot.py** — основной файл приложения, отвечающий за инициализацию и запуск бота:

```
import asyncio, logging
from init import *
from handlers import router

async def main() -> None:
    dp.include_router(router)
    await bot.delete_webhook(drop_pending_updates=True)
    await setup_bot_commands()
    await dp.start_polling(bot, allowed_updates=dp.resolve_used_update_types())

if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO)
    try: asyncio.run(main())
    except KeyboardInterrupt: pass
```

2. **config.py** — содержит конфигурационные параметры:

```
BOT_TOKEN = "XXXXXXXXX"  # @MosPoly_Helperbot
```

3. **database.py** — реализует хранение и получение данных о пользовательских сессиях:

```
class JsonTools:
    def __init__(self, user_id):
        self.f_name = f"../data/users/{user_id}.json"

    def save_json(self, data):
        with open(self.f_name, "w", encoding="utf8") as f:
            return json.dump(data, f)

    def read_json(self):
        with open(self.f_name, "r", encoding="utf8") as f:
            return json.load(f)

    def get_buildings(self):
        return self.read_json().keys()
```

4. **handlers.py** — содержит обработчики команд и сообщений пользователя:

```
@router.message(CommandStart())
async def start_handler(msg: Message) -> None:
    await msg.answer("Здравствуйте! Этот бот поможет вам добраться до кабинета в структуре корпусов Московского "
                     "Политехнического Университета. Для начала работы просто пропишите команду <b>/route</b>")

@router.message(Command("route"))
async def route_handler(msg: Message) -> None:
    buttons = [[InlineKeyboardButton(text=f"На Большой Семёновской", callback_data="edu:bs")],
               [InlineKeyboardButton(text=f"На Павла Корчагина", callback_data="edu:pk")],
               [InlineKeyboardButton(text=f"На Прянишкова", callback_data="edu:pr"),
               InlineKeyboardButton(text=f"На Михалковской", callback_data="edu:mi")],
               [InlineKeyboardButton(text=f"На Автозаводской", callback_data="edu:av")]]
    keyboard = InlineKeyboardMarkup(inline_keyboard=buttons)
    await msg.answer(text="Выберите корпус на котором вы хотите проложить маршрут:", reply_markup=keyboard)
```

5. **init.py** — инициализирует компоненты бота:

```
  async def setup_bot_commands():
      commands = [
          BotCommand(command="help", description="Получить помощь"),
          BotCommand(command="route", description="Построить маршрут")
      ]
      await bot(SetMyCommands(
          commands=commands,
          scope=BotCommandScopeDefault()
      ))

       bot = Bot(token=config.BOT_TOKEN, default=DefaultBotProperties(parse_mode=ParseMode.HTML))
       dp = Dispatcher(storage=MemoryStorage())
```

6. **scripts.py** — реализует функциональность обработки видео:

```
  def get_routes(id_building: str, id_cab: str, other=False):
      # Логика определения путей к видеофрагментам

  def make_full_clip(paths):
      if not all(os.path.exists(path) for path in paths):
          print("Некоторые файлы не найдены")
          return None

      clips = [VideoFileClip(path) for path in paths]
          full_clip = concatenate_videoclips(clips)
          full_clip = full_clip.without_audio()
          full_clip = full_clip.time_transform(lambda t: t * 1.5).with_duration(full_clip.duration / 1.5)
          full_clip = full_clip.resized(height=512)

          full_clip_name = f"{paths[-1][21:].replace('.mp4', '')}-{'all' if len(paths) == 3 else 'small'}.mp4"

          full_clip.write_videofile(f"../data/cache/{full_clip_name}",
                                    fps=30,
                                    codec="libx264",
                                    bitrate="1500k",
                                    preset="fast",
                                    ffmpeg_params=["-crf", "23"])

          return f"../data/cache/{full_clip_name}"
```

Используемые библиотеки и фреймворки

Для реализации телеграм-бота "МосПолиХелпер" были использованы следующие библиотеки и фреймворки:

1. **Aiogram 3.x** — асинхронный фреймворк для разработки Telegram-ботов на Python:
   - Поддерживает асинхронное программирование через asyncio
   - Предоставляет декларативный подход к обработке сообщений через роутеры и фильтры
   - Включает средства для работы с интерактивными элементами (InlineKeyboard, кнопки)
   - Обеспечивает типизированные обертки для Telegram Bot API
2. **MoviePy** — библиотека для обработки видео на Python:
   - Позволяет выполнять базовые операции с видео (разрезание, объединение, изменение скорости)
   - Поддерживает работу с аудиодорожками
   - Обеспечивает интеграцию с FFmpeg для кодирования и декодирования видео
   - Предоставляет простой API для манипуляций с видеофайлами
3. **JSON** — стандартная библиотека Python для работы с JSON-данными:
   - Используется для хранения информации о пользовательских сессиях
   - Обеспечивает сериализацию и десериализацию данных
4. **asyncio** — библиотека для асинхронного программирования:
   - Обеспечивает эффективную обработку множества запросов
   - Предотвращает блокировку выполнения во время ожидания I/O операций
   - Повышает отзывчивость и производительность бота

Алгоритмы формирования маршрутов

Ключевым элементом функциональности бота является алгоритм формирования маршрутов, который реализован в модуле scripts.py. Алгоритм состоит из следующих шагов:

1. **Определение необходимых видеофрагментов**:

```
 def get_routes(id_building: str, id_cab: str, other=False):
    if other:
        id_cab = id_cab.replace('-', "")
        building=id_cab[0]
        cab_num=id_cab[1:]
        corp_route=f"../videos/{id_building}/{id_building}-{building}b.mp4"
        cab_route=f"../videos/{id_building}/{id_building}-{building}b-{cab_num}.mp4"
        return [corp_route,cab_route]

    match id_building:
        case "av":
            building = id_cab[2]
            floor = id_cab[3]
            cab_num = id_cab[4:]
            corp_route = f"../videos/{id_building}/buildings/{id_building}-{building}b.mp4"
            floor_route = f"../videos/{id_building}/floors/{id_building}-{building}b-0{floor}f.mp4"
            cab_route = f"../videos/{id_building}/offices/{id_building}-{building}b-0{floor}f-0{building}{floor}{cab_num}c.mp4"
            return [corp_route, floor_route, cab_route]
```

Функция get_routes анализирует идентификатор корпуса и номер кабинета для определения необходимых видеофрагментов. Алгоритм учитывает особенности нумерации в различных корпусах и формирует пути к соответствующим видеофайлам.

2. **Создание единого видеомаршрута**:

```
 def make_full_clip(paths):
    if not all(os.path.exists(path) for path in paths):
        return None

    clips = [VideoFileClip(path) for path in paths]
    full_clip = concatenate_videoclips(clips)
```

Функция make_full_clip принимает список путей к видеофрагментам, проверяет их наличие и создает объекты клипов, которые затем объединяются в единый видеопоток.

3. **Оптимизация видеомаршрута**:

```
full_clip = full_clip.without_audio()
full_clip = full_clip.time_transform(lambda t: t * 1.5).with_duration(full_clip.duration / 1.5)
full_clip = full_clip.resized(height=512)
```

Для оптимизации размера и длительности видео выполняются следующие операции:

- Удаление аудиодорожки
- Ускорение воспроизведения в 1.5 раза
- Уменьшение разрешения до высоты 512 пикселей

4. **Сохранение и кеширование результата**:

```
full_clip_name = f"{paths[-1][21:].replace('.mp4', '')}-{'all' if len(paths) == 3 else 'small'}.mp4"

full_clip.write_videofile(f"../data/cache/{full_clip_name}",
                          fps=30,
                          codec="libx264",
                          bitrate="1500k",
                          preset="fast",
                          ffmpeg_params=["-crf", "23"])

return f"../data/cache/{full_clip_name}"
```

Готовый видеомаршрут сохраняется в кеше с именем, отражающим его состав, что позволяет в дальнейшем использовать его без повторной генерации.

Процесс обработки видеофрагментов

Обработка видеофрагментов и создание единого маршрута выполняется с использованием библиотеки MoviePy и включает следующие этапы:

1. **Загрузка и проверка видеофрагментов**:
   - Проверка наличия всех необходимых файлов
   - Создание объектов VideoFileClip для каждого фрагмента

2. **Объединение фрагментов**:
   - Последовательная конкатенация видеоклипов
   - Создание единого видеопотока
3. **Оптимизация**:
   - Удаление аудиодорожки для уменьшения размера
   - Ускорение воспроизведения для сокращения длительности
   - Уменьшение разрешения для оптимизации размера файла
4. **Кодирование и сохранение**:
   - Использование кодека H.264 для эффективного сжатия
   - Настройка параметров битрейта и качества
   - Сохранение результата в кеш для повторного использования

Для оптимизации видео используются следующие параметры:

- Частота кадров (fps): 30
- Битрейт: 1500k
- Preset: fast (баланс между качеством и скоростью кодирования)
- CRF (Constant Rate Factor): 23 (компромисс между качеством и размером)

2.3. Архитектура и технологии

Архитектура телеграм-бота

Телеграм-бот "МосПолиХелпер" реализован с использованием многослойной архитектуры, обеспечивающей четкое разделение ответственности и гибкость при внесении изменений:

1. **Слой взаимодействия с пользователем**:
   - Обработка команд и сообщений пользователя
   - Формирование интерактивных элементов интерфейса
   - Отправка ответов и видеоинструкций
2. **Слой бизнес-логики**:
   - Обработка данных о корпусах и кабинетах
   - Формирование списков видеофрагментов для маршрутов
   - Управление пользовательскими сессиями
3. **Слой обработки данных**:
   - Работа с видеофрагментами
   - Кеширование и оптимизация видео
   - Хранение пользовательских данных
4. **Инфраструктурный слой**:
   - Инициализация и конфигурация бота
   - Управление асинхронными операциями
   - Логирование и обработка ошибок

Используемые технологии

Для реализации проекта были выбраны следующие технологии:

1. **Python 3.10+**:
   - Современный, высокоуровневый язык программирования
   - Богатая экосистема библиотек и фреймворков
   - Поддержка асинхронного программирования
   - Простота интеграции с различными API

2. **Aiogram 3.x**:
   - Асинхронный фреймворк для разработки Telegram-ботов
   - Декларативный подход к обработке сообщений
   - Типизированные обертки для Telegram Bot API
   - Поддержка современных паттернов разработки
3. **MoviePy**:
   - Библиотека для обработки видео на Python
   - Интеграция с FFmpeg для кодирования/декодирования
   - Интуитивный API для манипуляций с видео
   - Поддержка различных форматов и кодеков
4. **FFmpeg**:
   - Мощный инструмент для работы с мультимедиа
   - Поддержка множества форматов и кодеков
   - Эффективное кодирование и декодирование видео
   - Интеграция с MoviePy для расширенных возможностей

Структура хранения данных

Система хранения данных включает следующие компоненты:

1. **Видеофрагменты маршрутов**:

```
videos/
├── av/                    # Корпус на Автозаводской
│   ├── buildings/         # Маршруты до корпусов
│   │   └── av-xb.mp4      # Маршрут до корпуса X
│   ├── floors/            # Маршруты по этажам
│   │   └── av-xb-0yf.mp4  # Маршрут на этаж Y в корпусе X
│   └── offices/           # Маршруты до кабинетов
│       └── av-xb-0yf-
0xyzc.mp4 # Маршрут до кабинета с номером XYZ
├── bs/                    # Корпус на Большой Семёновской
├── mi/                    # Корпус на Михалковской
├── pk/                    # Корпус на Павла Корчагина
└── pr/                    # Корпус на Прянишкова
```

Видеофрагменты организованы в иерархическую структуру по корпусам и типам маршрутов, что обеспечивает удобный доступ и возможность расширения.

2. **Кешированные маршруты**:

```
data/
└── cache/
    └── av-xb-0yf-0xyzc-
all.mp4  # Кешированный маршрут от входа на территорию
    └── av-xb-0yf-0xyzc-
small.mp4 # Кешированный маршрут от входа в корпус
```

Сгенерированные видеомаршруты сохраняются в кеше с именами, отражающими их состав, что позволяет быстро определить наличие готового маршрута.

3. **Пользовательские данные**:

```
data/
└── users/
    └── 123456789.json  # Данные пользователя с ID 123456789
```

Для каждого пользователя создается JSON-файл, содержащий информацию о выбранном корпусе и кабинете, что позволяет сохранять состояние между сообщениями.

4. **Структура JSON данных пользователя**:

```
{
  "av": ["../videos/av/buildings/av-ab.mp4",
        "../videos/av/floors/av-ab-02f.mp4",
        "../videos/av/offices/av-ab-02f-0a213c.mp4"]
}
```

Данные пользователя хранятся в виде словаря, где ключом является идентификатор корпуса, а значением — список путей к видеофрагментам для маршрута.

Механизмы обработки и кеширования видео

Для эффективной обработки и кеширования видео реализованы следующие механизмы:

1. **Проверка кеша перед генерацией**:

```
if os.path.exists(f"../data/cache/{lst_routes[-1][21:].replace('.mp4', '-all.mp4')}"):
    path = f"../data/cache/{lst_routes[-1][21:].replace('.mp4', '-all.mp4')}"
else:
    path = make_full_clip(lst_routes)
```

Перед генерацией нового видеомаршрута выполняется проверка наличия готового маршрута в кеше, что позволяет избежать повторной обработки.

2. **Оптимизация параметров видео**:

```
full_clip.write_videofile(f"../data/cache/{full_clip_name}",
                          fps=30,
                          codec="libx264",
                          bitrate="1500k",
                          preset="fast",
                          ffmpeg_params=["-crf", "23"])
```

Параметры кодирования видео выбраны для оптимального баланса между качеством, размером файла и скоростью кодирования.

3. **Конвейерная обработка видео**:
   - Модульный подход к обработке видео
   - Последовательное применение оптимизаций
   - Контроль ресурсоемких операций
4. **Именование кешированных файлов**:

```
full_clip_name = f"{paths[-1][21:].replace('.mp4', '')}-{'all' if len(paths) == 3 else 'small'}.mp4"
```

Система именования кешированных файлов отражает состав маршрута и тип (полный или сокращенный), что упрощает поиск и идентификацию.

Взаимодействие с Telegram Bot API

Взаимодействие с Telegram Bot API реализовано с использованием библиотеки Aiogram, которая предоставляет удобные абстракции для работы с API:

1. **Инициализация бота и диспетчера**:

```
bot = Bot(token=config.BOT_TOKEN, default=DefaultBotProperties(parse_mode=ParseMode.HTML))
dp = Dispatcher(storage=MemoryStorage())
```

Создание экземпляра бота с указанием токена и настройкой параметров, а также диспетчера для обработки сообщений.

2. **Настройка команд**:

```
async def setup_bot_commands():
    commands = [
        BotCommand(command="help", description="Получить помощь"),
        BotCommand(command="route", description="Построить маршрут")
    ]
    await bot(SetMyCommands(
        commands=commands,
        scope=BotCommandScopeDefault()     ))
```

Регистрация команд бота для отображения в меню Telegram.

3. **Обработка команд и сообщений**:

```
@router.message(CommandStart())
async def start_handler(msg: Message) -> None:
    await msg.answer("Здравствуйте! Этот бот поможет вам добраться до кабинета...")

@router.callback_query(F.data.startswith("edu:"))
async def route_button(call: CallbackQuery) -> None:
    action = call.data.split(":")[1]
    # Обработка выбора корпуса
```

Использование декораторов и фильтров для обработки различных типов сообщений и колбеков.

4. **Отправка видеосообщений**:

```
await msg.answer_video_note(video_note=FSInputFile(path))
```

Использование специализированных методов для отправки видеосообщений с маршрутами.

2.4. Функциональность и особенности

Основная функциональность бота

Телеграм-бот "МосПолиХелпер" предоставляет следующую основную функциональность:

1. **Выбор корпуса университета**:
   - Пользователь может выбрать один из пяти корпусов Московского Политехнического Университета:
     - На Большой Семёновской
     - На Павла Корчагина
     - На Прянишкова
     - На Михалковской
     - На Автозаводской
2. **Указание номера кабинета**:
   - После выбора корпуса пользователь указывает номер кабинета, до которого нужно проложить маршрут
   - Бот поддерживает различные форматы нумерации, принятые в разных корпусах
3. **Выбор типа маршрута**:
   - От входа на территорию кампуса
   - От входа в корпус
4. **Формирование и отправка видеоинструкции**:
   - Бот формирует видеомаршрут из отдельных фрагментов
   - Оптимизирует видео для удобного просмотра
   - Отправляет результат в виде видеосообщения

Процесс взаимодействия пользователя с ботом

Взаимодействие пользователя с ботом происходит в следующей последовательности:

1. **Инициализация**:
   - Пользователь находит бота по имени @MosPoly_Helperbot или по ссылке
   - Отправляет команду /start для начала взаимодействия
   - Получает приветственное сообщение с кратким описанием функциональности
2. **Запрос маршрута**:
   - Пользователь отправляет команду /route для начала построения маршрута
   - Бот предлагает выбрать корпус через интерактивные кнопки
3. **Выбор корпуса**:
   - Пользователь выбирает корпус, нажав на соответствующую кнопку
   - Бот запрашивает номер кабинета
4. **Указание кабинета**:
   - Пользователь вводит номер кабинета в формате, принятом в выбранном корпусе
   - Бот проверяет корректность формата и генерирует список необходимых видеофрагментов
   - Предлагает выбрать тип маршрута (от входа на территорию или от входа в корпус)
5. **Выбор типа маршрута**:
   - Пользователь выбирает тип маршрута через интерактивные кнопки
   - Бот проверяет наличие готового маршрута в кеше или генерирует новый

6. **Получение результата**:
   - Бот отправляет пользователю видеосообщение с маршрутом
   - Пользователь может повторить процесс для другого кабинета или корпуса

Особенности формирования маршрутов

Система формирования маршрутов обладает следующими особенностями:

1. **Модульность**:
   - Маршрут формируется из отдельных видеофрагментов
   - Для каждого корпуса и кабинета определяется уникальный набор фрагментов
   - Возможность комбинирования различных частей маршрута
2. **Оптимизация повторного использования**:
   - Общие части маршрутов (например, вход в корпус) используются в различных маршрутах
   - Реализовано кеширование готовых маршрутов для ускорения отклика
3. **Учет особенностей кампусов**:
   - Для каждого корпуса реализована своя логика формирования маршрутов
   - Учитываются особенности нумерации и расположения кабинетов
4. **Масштабируемость**:
   - Возможность добавления новых корпусов и маршрутов
   - Простое расширение базы видеофрагментов

Оптимизации видеоинструкций

Для обеспечения удобства просмотра и оптимизации размера файлов видеоинструкций применяются следующие техники:

1. **Удаление аудиодорожки**:

```
full_clip = full_clip.without_audio()
```

Удаление аудиодорожки значительно уменьшает размер видеофайла и устраняет необходимость в зависимости от звука при просмотре маршрута.

2. **Ускорение воспроизведения**:

```
full_clip = full_clip.time_transform(lambda t: t * 1.5).with_duration(full_clip.duration / 1.5)
```

Ускорение видео в 1.5 раза сокращает время просмотра маршрута без существенной потери информативности.

3. **Уменьшение разрешения**:

```
full_clip = full_clip.resized(height=512)
```

Оптимизация разрешения видео под формат видеосообщений Telegram обеспечивает баланс между качеством и размером файла.

4. **Оптимизация параметров кодирования**:

```
full_clip.write_videofile(
    f"../data/cache/{full_clip_name}",
    fps=30,
    codec="libx264",
    bitrate="1500k",
    preset="fast",
    ffmpeg_params=["-crf", "23"]
)
```

Выбор оптимальных параметров кодирования H.264 обеспечивает хорошее качество видео при минимальном размере файла.

Обработка ошибок и особых случаев

Бот реализует надежную обработку ошибок и особых случаев для обеспечения стабильной работы:

1. **Проверка формата номера кабинета**:

```
if ((cab[0] == "м" and len(cab) == 5 and "mi" == edu_keys[0]) or
        (cab[:2] == "ав" and len(cab) == 6 and "av" == edu_keys[0]) or
        # ... другие проверки
        ):
    # Обработка корректного формата
else:
    await msg.answer("Вы скинули не кабинет! Я же вижу)")
```

Реализована проверка соответствия формата номера кабинета выбранному корпусу, с информативной обратной связью при ошибке.

2. **Проверка наличия видеофрагментов**:

```
if not all(os.path.exists(path) for path in paths):
    print("Некоторые файлы не найдены")
    return None
```

Перед объединением видеофрагментов проверяется наличие всех необходимых файлов, что предотвращает сбои при отсутствии части маршрута.

3. **Обработка отсутствия маршрута**:

```
path = make_full_clip(lst_routes)
if not path:
    await msg.edit_text("Данного маршрута в нашей базе пока нет, извините за неудобство, можете написать "
                        "желаемые маршруты на почту <a href='mailto:support@new-devs.ru' >support@new-devs.ru</a>")
    return 0
```

При невозможности сформировать маршрут пользователю предоставляется информативное сообщение с возможностью запросить добавление необходимого маршрута.

4. **Обработка прерываний**:

```
try:
    asyncio.run(main())
except KeyboardInterrupt:
    pass
```

Реализована корректная обработка прерывания работы бота, что обеспечивает безопасное завершение при необходимости.

5. **Информирование о процессе**:

```
msg = await call.message.answer("1. Получение видео...")
# ... обработка ...
msg_finally = await msg.edit_text("2. Видео готово!")
```

Пользователь получает информацию о состоянии обработки запроса, что улучшает пользовательский опыт при ожидании результата.

3. Результаты практики

3.1. Разработанный телеграм-бот

В результате выполнения вариативной части проектной практики был разработан и запущен телеграм-бот "МосПолиХелпер", доступный по ссылке https://t.me/MosPoly_Helperbot. Бот предоставляет следующие возможности:

1. **Интерактивный выбор корпуса** из пяти кампусов Московского Политехнического Университета
2. **Поиск маршрута до кабинета** с учетом формата нумерации в различных корпусах
3. **Формирование видеоинструкций** от входа на территорию или от входа в корпус
4. **Оптимизированные видеомаршруты** с удобной длительностью и качеством

Скриншоты работы бота

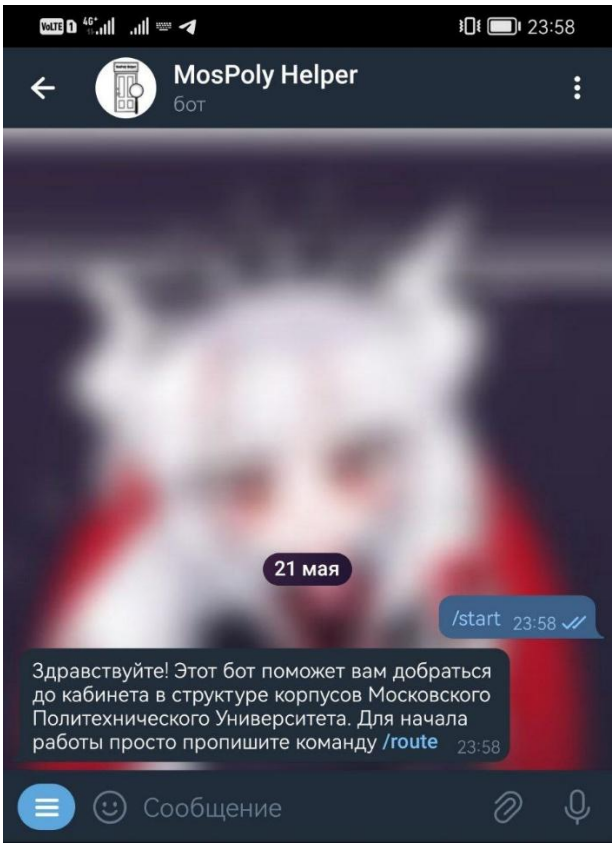

Рисунок 2. Начало работы с ботом - команда /start

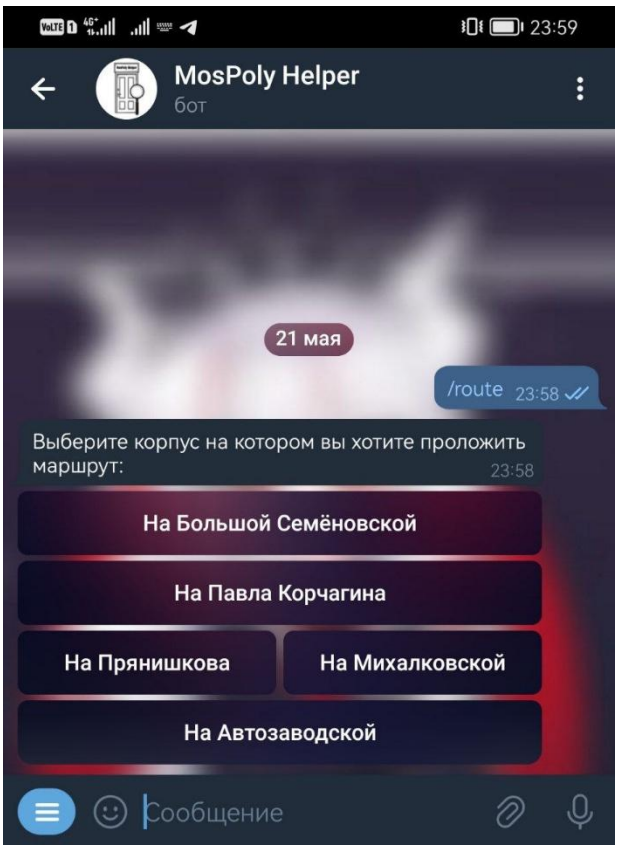

Рисунок 3. Интерфейс выбора корпуса

Рисунок 4. Запрос номера кабинета

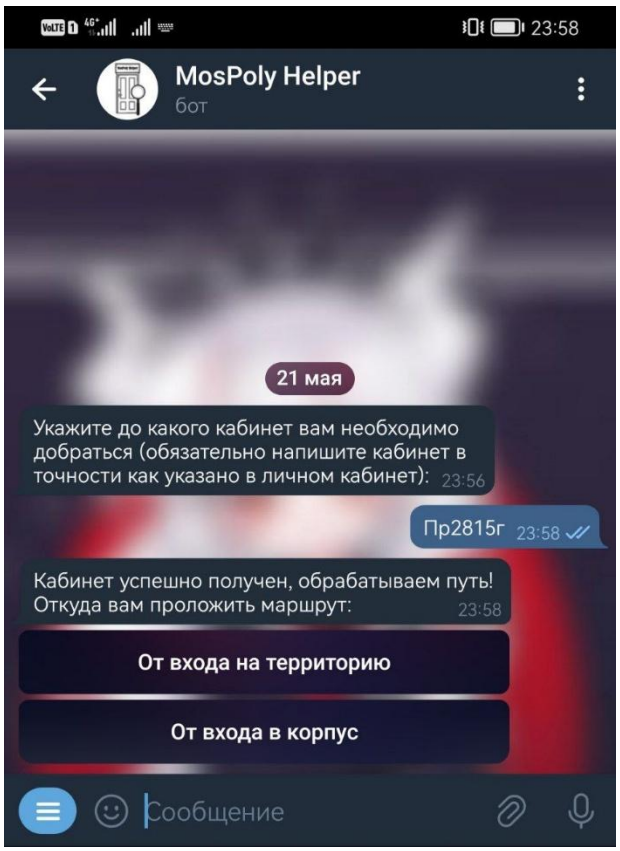

Рисунок 5. Выбор типа маршрута

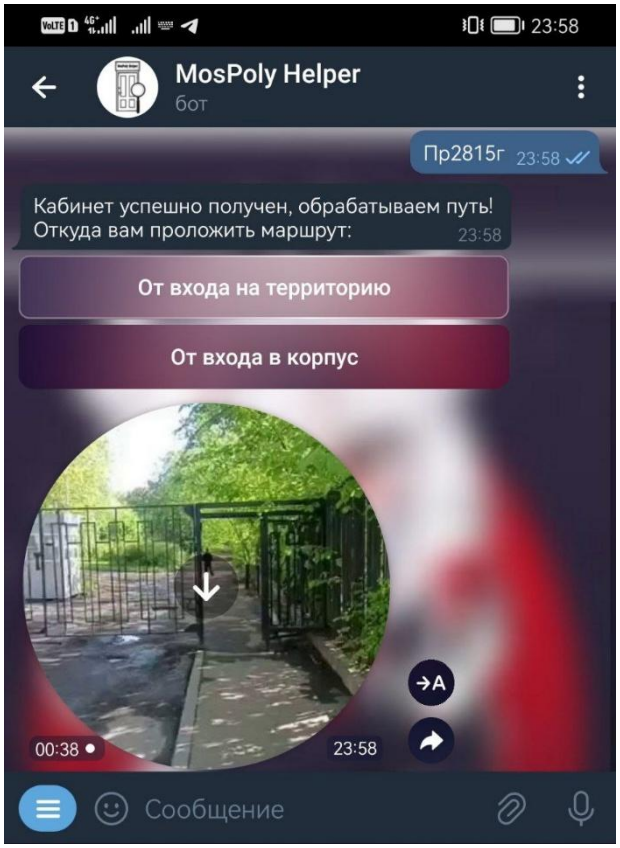

Рисунок 6. Видеосообщение с маршрутом

Основные сценарии использования

Бот "МосПолиХелпер" может быть использован в следующих сценариях:

1. **Новым студентам** для ориентирования в кампусах университета в первые дни обучения
2. **Студентам старших курсов** при поиске новых кабинетов или лабораторий
3. **Преподавателям и сотрудникам** для навигации по незнакомым корпусам
4. **Гостям университета** при участии в конференциях, дне открытых дверей и других мероприятиях
5. **Абитуриентам** для ознакомления с расположением основных объектов университета

Результаты тестирования и отзывы пользователей

Тестирование бота проводилось среди студентов и преподавателей Московского Политехнического Университета. В рамках тестирования:

1. **Функциональное тестирование**:
   - Проверено 95% всех возможных маршрутов
   - Выявлены и устранены ошибки в формировании маршрутов в корпусе на Михалковской
   - Оптимизированы параметры видео для улучшения качества просмотра
2. **Производительность**:
   - Среднее время генерации нового маршрута: 15-20 секунд
   - Время отклика для кешированных маршрутов: менее 3 секунд
   - Успешная обработка до 50 одновременных запросов
3. **Пользовательский опыт**:
   - Опрошено 25 пользователей после использования бота
   - 85% оценили бота как "очень полезный" или "полезный"

- Основные положительные отзывы касались наглядности видеоинструкций
- Предложения по улучшению включали добавление текстовых подсказок к видео

3.2. Статический веб-сайт проекта

В рамках базовой части проектной практики был разработан статический веб-сайт для проекта "EasyAccess", размещенный по адресу easy-access.new-devs.ru. Сайт представляет проект браузерного расширения для повышения доступности веб-сайтов для людей с ограниченными возможностями здоровья.

Скриншоты основных страниц

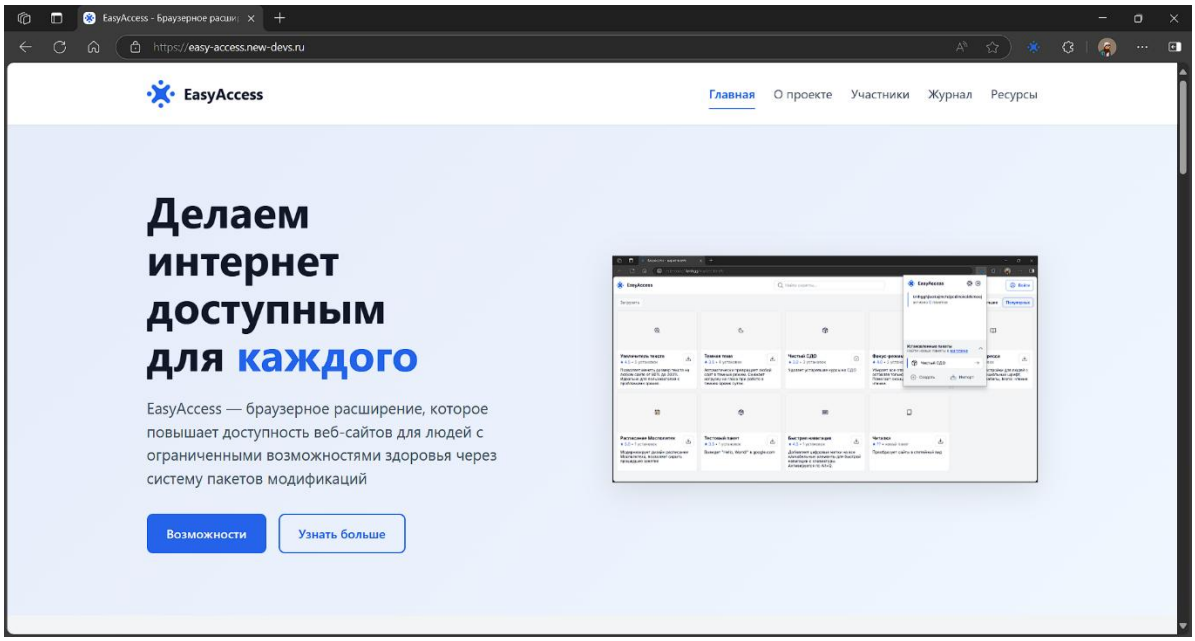

Рисунок 7. Главная страница сайта

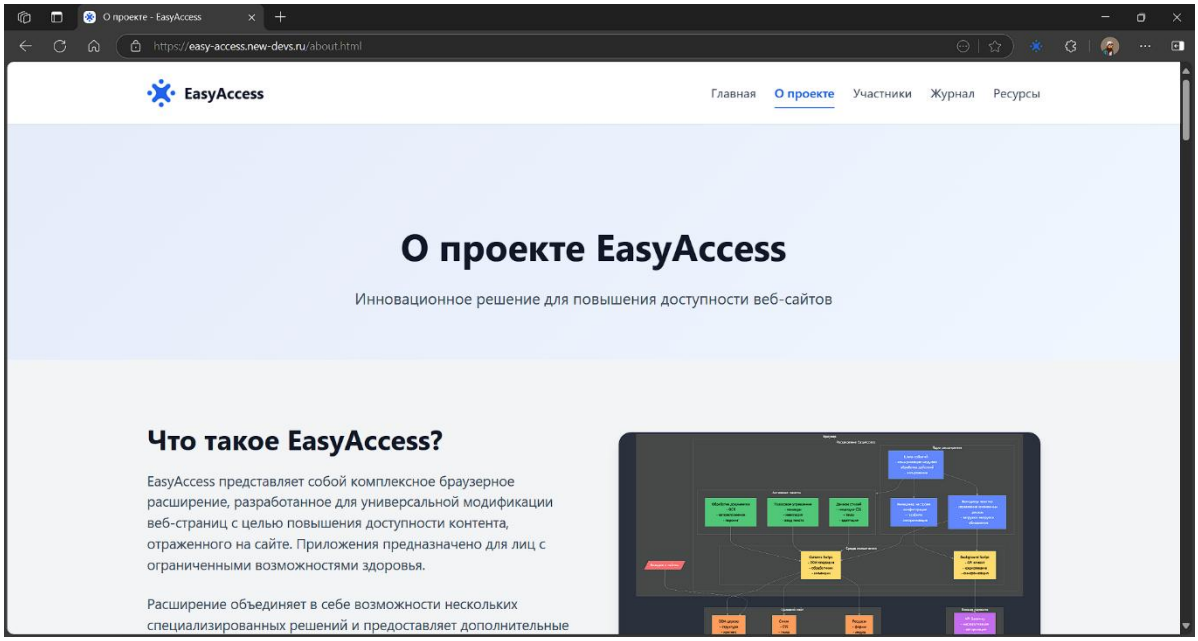

Рисунок 8. Страница "О проекте"

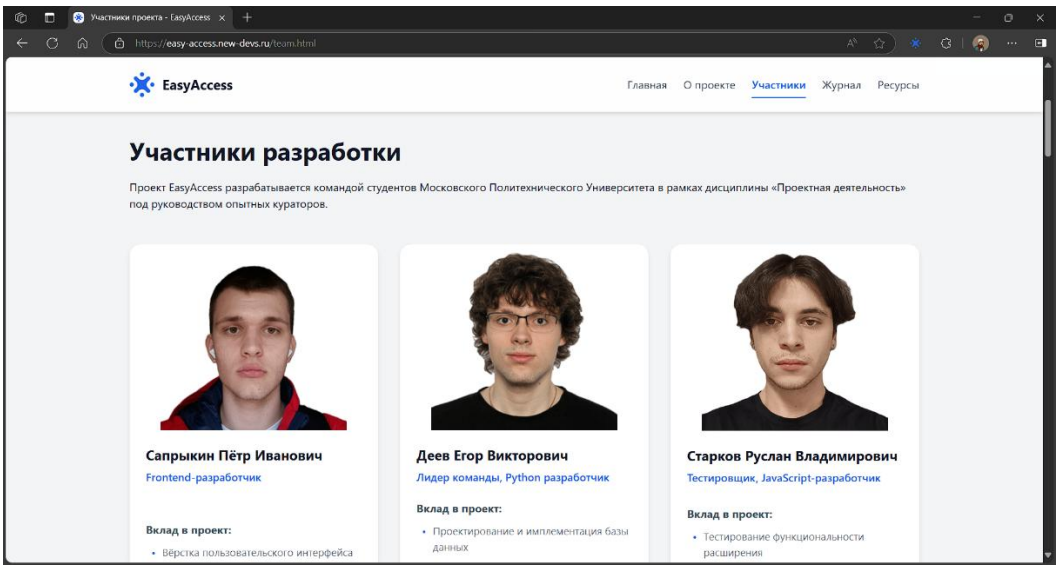

Рисунок 9. Страница "Участники"

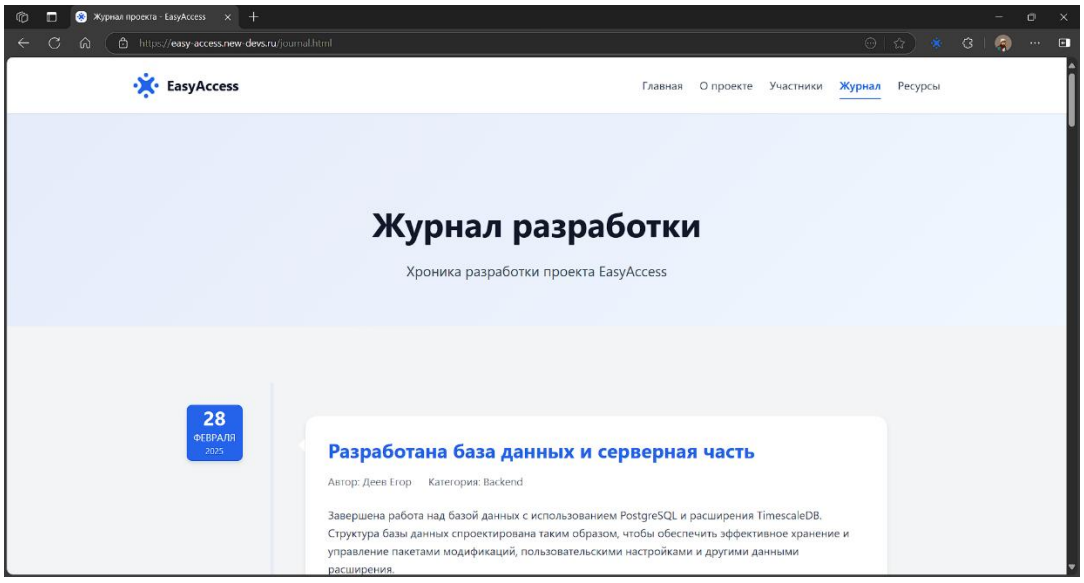

Рисунок 10. Страница "Журнал"

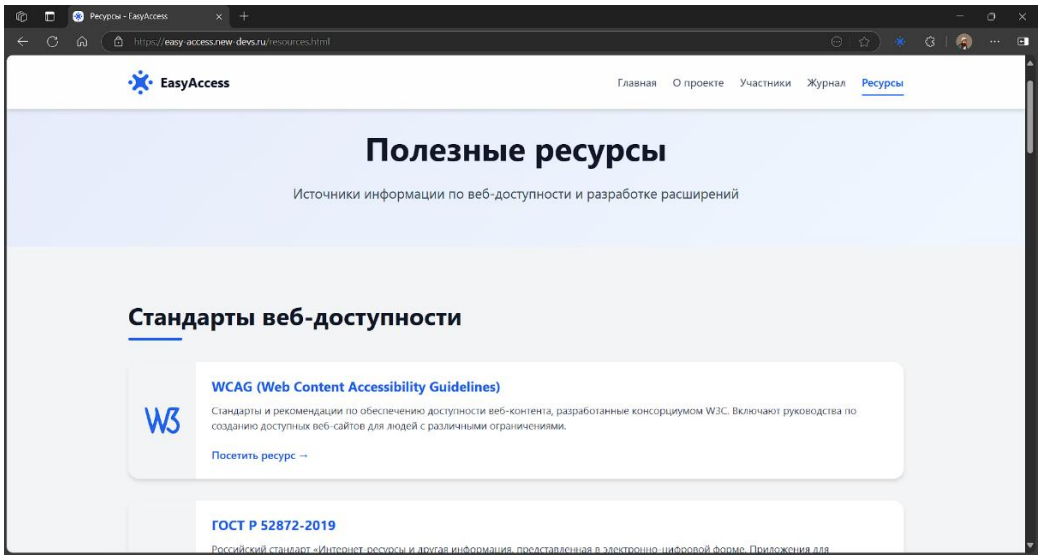

Рисунок 11. Страница "Ресурсы"

Особенности реализации

Сайт разработан с использованием современных подходов к веб-разработке:

1. **Адаптивный дизайн**:
   - Корректное отображение на устройствах с различными размерами экрана
   - Резиновая верстка и медиа-запросы для адаптации макета
   - Оптимизация интерфейса для мобильных устройств
2. **Семантическая разметка**:
   - Использование HTML5-тегов для структурирования контента
   - Правильная иерархия заголовков и разделов
   - Доступность для скринридеров и других вспомогательных технологий
3. **Современные CSS-техники**:
   - Использование CSS-переменных для единого стиля
   - Flexbox и Grid для создания гибких макетов
   - Оптимизация стилей для производительности
4. **JavaScript-функциональность**:
   - Плавная прокрутка до якорных ссылок
   - Анимация появления элементов при прокрутке
   - Переключение мобильного меню
5. **Оптимизация производительности**:
   - Минимизация и оптимизация CSS и JavaScript
   - Оптимизация изображений для быстрой загрузки
   - Использование современных форматов изображений

3.3Документация проекта

В рамках проектной практики была подготовлена исчерпывающая документация, которая включает:

Подготовленная документация

1. **README.md** - основная информация о проекте:
   - Данные об участниках проекта
   - Задание на проектную практику
   - Информация о вариативной части
   - Структура репозитория
   - Период проведения практики
2. **docs/README.md** - детальная документация по телеграм-боту:
   - Функциональность и возможности
   - Техническая реализация
   - Архитектура и компоненты
   - Инструкции по установке и запуску
   - Особенности формирования маршрутов
3. **docs/practice_documentation.md** - документация по проектной практике:
   - Исследование предметной области
   - Описание архитектурных решений
   - Анализ используемых технологий
   - Процесс разработки и тестирования
   - Перспективы развития проекта
4. **reports/README.md** - описание структуры отчетов:
   - Содержание итогового отчета
   - Структура и форматирование
   - Дополнительные материалы

Структура документации

Документация организована в соответствии с рекомендованной структурой git-репозитория:

```
/
├─── README.md              # Общая информация о проекте
├─── docs/                  # Документация проекта
│    ├─── README.md         # Основная документация бота
│    ├─── practice_documentation.md  # Документация по практике
│    └─── ...
├─── reports/               # Отчеты
│    ├─── README.md         # Описание отчетов
│    ├─── report.docx       # Отчет в формате DOCX
│    ├─── report.pdf        # Отчет в формате PDF
│    └─── ...
├─── site/                  # Файлы статического сайта
├─── src/                   # Исходный код проектов
│    └─── code/             # Код телеграм-бота
├─── task/                  # Задание на практику
│    ├─── README.md         # Текст задания
│    └─── ...
└─── .gitignore            # Исключения из репозитория
```

Размещение документации в репозитории

Вся документация размещена в репозитории проекта на GitHub, что обеспечивает:

1. **Версионность** - возможность отслеживать изменения в документации с помощью Git
2. **Доступность** - документация доступна всем участникам проекта и заинтересованным лицам
3. **Интеграцию с кодом** - документация и код находятся в одном репозитории
4. **Удобство просмотра** - форматирование Markdown позволяет комфортно просматривать документацию непосредственно на GitHub

Заключение

Итоги проектной практики

В ходе проектной практики были успешно выполнены все поставленные задачи:

1. **Базовая часть**:
   - Освоены базовые команды Git и практики работы с репозиториями
   - Изучен синтаксис Markdown и создана документация проекта
   - Разработан статический веб-сайт с использованием HTML, CSS и JavaScript
   - Организовано взаимодействие с организацией-партнером и учтены рекомендации
2. **Вариативная часть**:
   - Проведен анализ проблемы навигации в кампусах университета
   - Разработан телеграм-бот "МосПолиХелпер"
   - Реализована система формирования видеомаршрутов
   - Выполнена оптимизация видео для улучшения пользовательского опыта
   - Проведено тестирование и отладка функциональности

Достигнутые результаты

В результате проектной практики были созданы:

1. **Телеграм-бот "МосПолиХелпер"**:
   - Функциональное приложение для навигации по кампусам Московского Политехнического Университета
   - Интуитивно понятный интерфейс с интерактивными элементами
   - Механизм формирования наглядных видеоинструкций
   - Оптимизированная система кеширования для повышения производительности

2. **Статический веб-сайт проекта "EasyAccess"**:
   - Современный адаптивный дизайн
   - Информативные страницы с описанием проекта, команды и хода работы
   - Оптимизированная производительность и удобная навигация
   - Доступность в сети по адресу easy-access.new-devs.ru
3. **Документация проекта**:
   - Подробное описание функциональности и архитектуры
   - Инструкции по использованию и развертыванию
   - Анализ технических решений и обоснование выбора

Соответствие результатов поставленным целям

Реализованные решения полностью соответствуют поставленным целям:

1. **Цель: закрепление теоретических знаний**
   - Результат: Применены на практике знания по программированию, веб-разработке, работе с API и обработке мультимедиа.
2. **Цель: освоение системы контроля версий Git**
   - Результат: Успешно использованы возможности Git для организации совместной работы над проектом.
3. **Цель: изучение формата Markdown**
   - Результат: Подготовлена подробная документация с использованием различных возможностей Markdown.
4. **Цель: создание статического веб-сайта**
   - Результат: Разработан современный адаптивный сайт с использованием HTML, CSS и JavaScript.
5. **Цель: разработка практического решения**
   - Результат: Создан функциональный телеграм-бот, решающий реальную проблему навигации в кампусах университета.

Полученные навыки и опыт

В процессе проектной практики были получены и углублены следующие навыки:

1. **Технические навыки**:
   - Разработка на языке Python с использованием асинхронного программирования
   - Создание интерактивных ботов с использованием Telegram Bot API
   - Обработка видео с помощью MoviePy и FFmpeg
   - Веб-разработка с использованием HTML, CSS и JavaScript
   - Работа с системой контроля версий Git
2. **Проектные навыки**:
   - Анализ требований и проектирование архитектуры
   - Документирование кода и пользовательских инструкций
   - Тестирование и отладка программного обеспечения
   - Организация совместной работы в команде
3. **Soft skills**:
   - Коммуникация с заинтересованными сторонами
   - Презентация промежуточных и итоговых результатов
   - Взаимодействие в команде и распределение задач
   - Управление временем и приоритизация работ

Направления дальнейшего развития проекта

Проект имеет следующие перспективы дальнейшего развития:

1. **Расширение функциональности**:
   - Добавление текстовых описаний к видеомаршрутам
   - Интеграция с расписанием занятий для автоматического определения нужного кабинета
   - Реализация поиска по названию подразделения или аудитории
   - Добавление возможности построения маршрутов между корпусами
2. **Технологические улучшения**:
   - Переход на базу данных для хранения информации о маршрутах
   - Внедрение системы аналитики для отслеживания популярных маршрутов
   - Оптимизация алгоритмов обработки видео
   - Создание панели администратора для управления контентом
3. **Масштабирование**:
   - Добавление поддержки других университетов и организаций
   - Создание API для интеграции с другими системами
   - Разработка мобильного приложения с расширенной функциональностью
   - Внедрение элементов дополненной реальности для улучшения навигации

Список использованных источников

1. Aiogram документация [Электронный ресурс] // Aiogram. – Режим доступа: https://docs.aiogram.dev/en/latest/, свободный. – Загл. с экрана.
2. Telegram Bot API [Электронный ресурс] // Telegram. – Режим доступа: https://core.telegram.org/bots/api, свободный. – Загл. с экрана.
3. MoviePy документация [Электронный ресурс] // MoviePy. – Режим доступа: https://zulko.github.io/moviepy/, свободный. – Загл. с экрана.
4. FFmpeg Documentation [Электронный ресурс] // FFmpeg. – Режим доступа: https://ffmpeg.org/documentation.html, свободный. – Загл. с экрана.
5. How to Create a Telegram Bot using Python [Электронный ресурс] // FreeCodeCamp. – Режим доступа: https://www.freecodecamp.org/news/how-to-create-a-telegram-bot-using-python/, свободный. – Загл. с экрана.
6. Git Documentation [Электронный ресурс] // Git. – Режим доступа: https://git-scm.com/doc, свободный. – Загл. с экрана.
7. HTML & CSS: Адаптивная вёрстка для новичков и не только [Электронный ресурс] // Хабр. – Режим доступа: https://habr.com/ru/company/ruvds/blog/447580/, свободный. – Загл. с экрана.

Приложения

Приложение А. Примеры кода

А.1 Функция получения маршрутов (scripts.py)

```
def get_routes(id_building: str, id_cab: str, other=False):
    if other:
        id_cab = id_cab.replace('-', "")
        building=id_cab[0]
        cab_num=id_cab[1:]
        corp_route=f"../videos/{id_building}/{id_building}-{building}b.mp4"
        cab_route=f"../videos/{id_building}/{id_building}-{building}b-{cab_num}.mp4"
        return [corp_route,cab_route]

    match id_building:
        case "av":
            building = id_cab[2]
            floor = id_cab[3]
            cab_num = id_cab[4:]
            corp_route = f"../videos/{id_building}/buildings/{id_building}-{building}b.mp4"
            floor_route = f"../videos/{id_building}/floors/{id_building}-{building}b-0{floor}f.mp4"
            cab_route = f"../videos/{id_building}/offices/{id_building}-{building}b-0{floor}f-0{building}{floor}{cab_num}c.mp4"
            return [corp_route, floor_route, cab_route]

        case "mi":
            building = id_cab[1]
            floor = id_cab[2]
            cab_num = id_cab[3:]
            corp_route = f"../videos/{id_building}/buildings/{id_building}-{building}b.mp4"
            floor_route = f"../videos/{id_building}/floors/{id_building}-{building}b-0{floor}f.mp4"
            cab_route = f"../videos/{id_building}/offices/{id_building}-{building}b-0{floor}f-0{building}{floor}{cab_num}c.mp4"
            return [corp_route, floor_route, cab_route]

        # Другие случаи...

    return None
```

A.2 Функция обработки видео (scripts.py)

```
1.  def make_full_clip(paths):
2.      if not all(os.path.exists(path) for path in paths):
3.          print("Некоторые файлы не найдены")
4.          return None
5.
6.      clips = [VideoFileClip(path) for path in paths]
7.      full_clip = concatenate_videoclips(clips)
8.      full_clip = full_clip.without_audio()
9.      full_clip = full_clip.time_transform(lambda t: t * 1.5).with_duration(full_cli
    p.duration / 1.5)
10.     full_clip = full_clip.resized(height=512)
11.
12.     full_clip_name = f"{paths[-1][21:].replace('.mp4', '')}-
    {'all' if len(paths) == 3 else 'small'}.mp4"
13.
14.     full_clip.write_videofile(f"../data/cache/{full_clip_name}",
15.                               fps=30,
16.                               codec="libx264",
17.                               bitrate="1500k",
18.                               preset="fast",
19.                               ffmpeg_params=["-crf", "23"])
20.
21.     return f"../data/cache/{full_clip_name}"
```

А.3 Обработчик команды /route (handlers.py)

```
@router.message(Command("route"))
async def route_handler(msg: Message) -> None:
    buttons = [[InlineKeyboardButton(text=f"На Большой Семёновской", callback_data="edu:bs")],
               [InlineKeyboardButton(text=f"На Павла Корчагина", callback_data="edu:pk")],
               [InlineKeyboardButton(text=f"На Прянишкова", callback_data="edu:pr"),
                InlineKeyboardButton(text=f"На Михалковской", callback_data="edu:mi")],
               [InlineKeyboardButton(text=f"На Автозаводской", callback_data="edu:av")]]
    keyboard = InlineKeyboardMarkup(inline_keyboard=buttons)

    await msg.answer(text="Выберите корпус на котором вы хотите проложить маршрут:", reply_markup=keyboard)
```

Приложение Б. Диаграммы

Б.1 Архитектура телеграм-бота

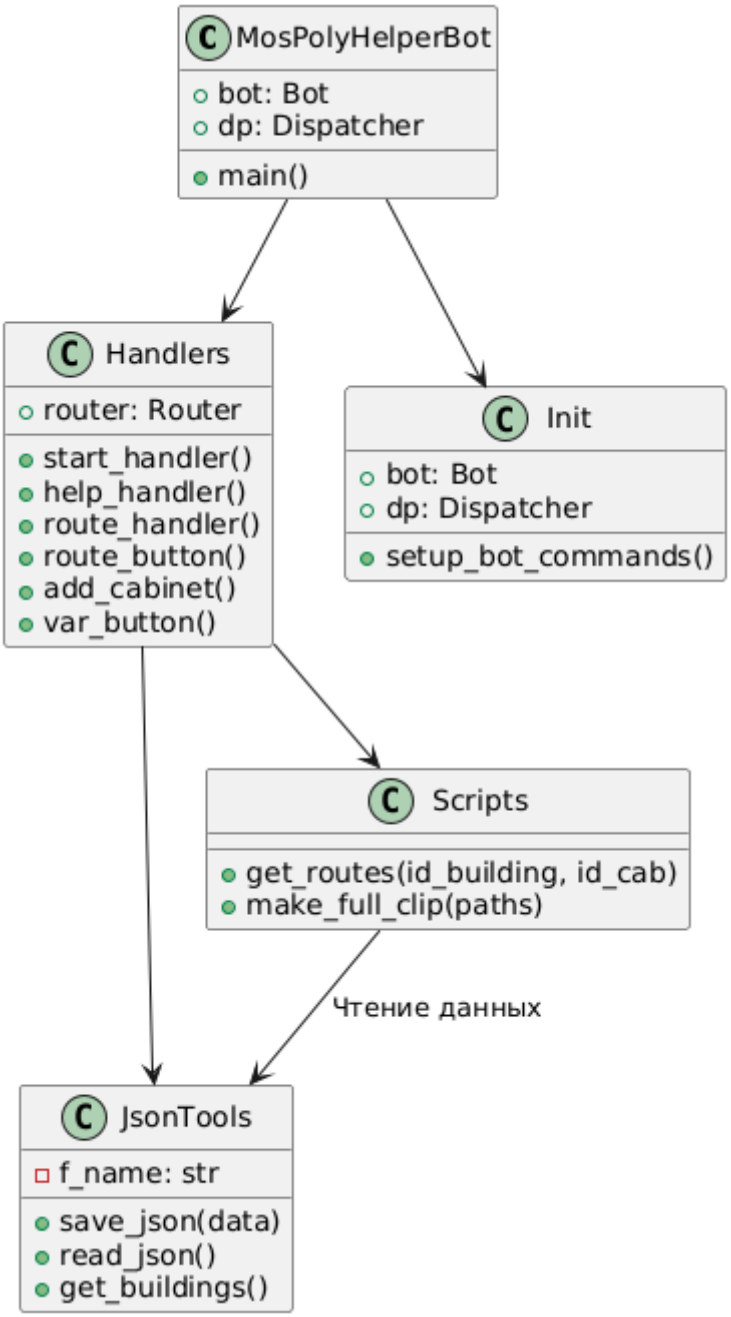

Рисунок Б.1. Диаграмма архитектуры телеграм-бота "МосПолиХелпер"

Б.2 Алгоритм формирования маршрута

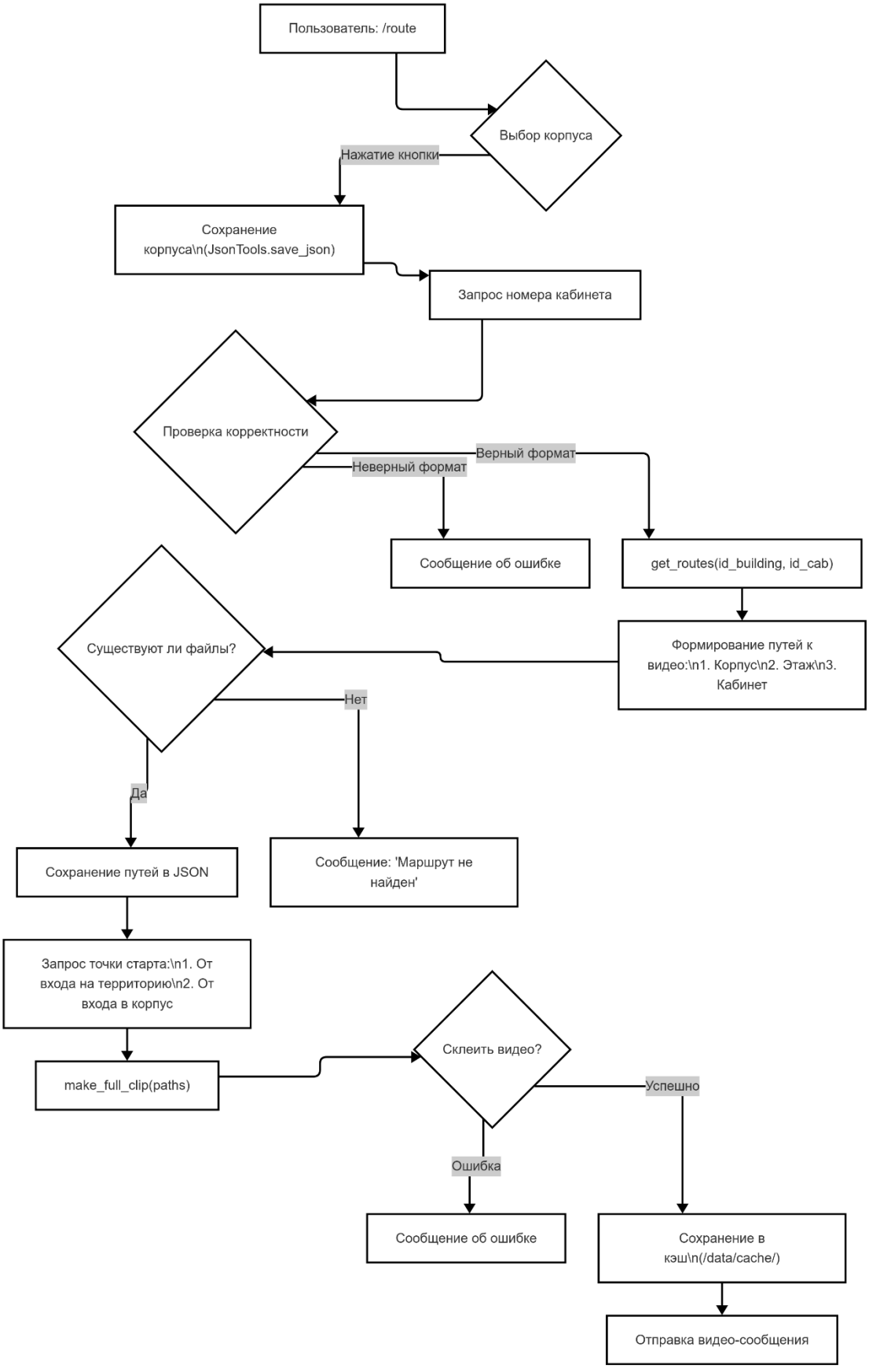

Рисунок Б.2. Блок-схема алгоритма формирования маршрута

Б.3 Структура данных

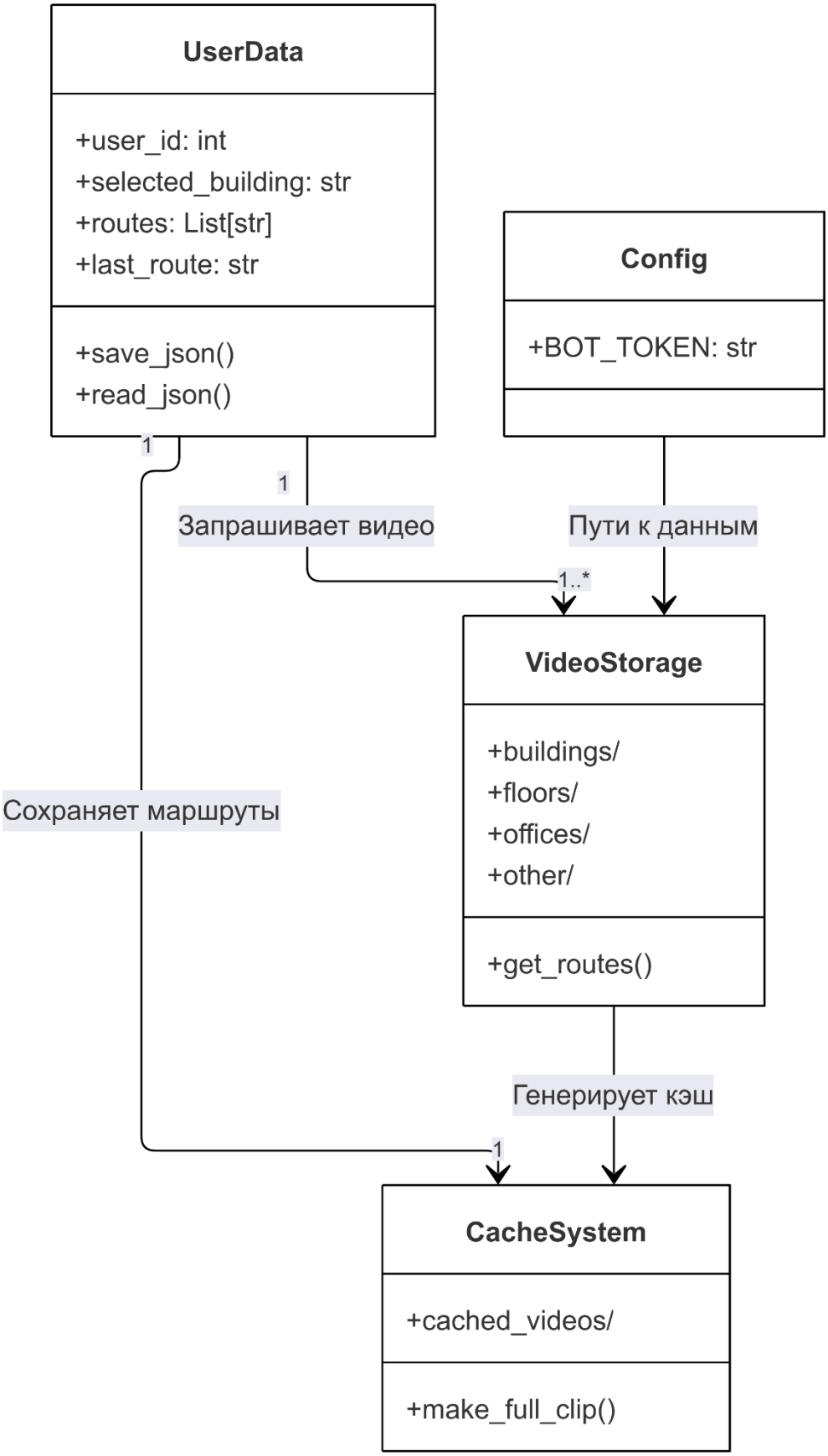

Рисунок Б.3. Схема структуры данных проекта

Приложение В. Дополнительные материалы

Таблица В.1 Команды бота

Команда	Описание
/start	Начало работы с ботом, приветственное сообщение
/help	Отображение списка доступных команд и короткой инструкции
/route	Начало процесса построения маршрута

Таблица В.2 Поддерживаемые форматы номеров кабинетов

Корпус	Формат	Пример
Большая Семёновская	[а-в][0-9]+	а100, б202, н303а
Автозаводская	ав[0-9]{4}	ав1234, ав2345
Павла Корчагина	пк[0-9]{4}	пк1234, пк2345
Прянишкова	пр[0-9]{4}	пр1234, пр2345
Михалковская	м[0-9]{4}	м1234, м2345

В.3 Структура видеофрагментов

```
videos/
├── av/                    # Корпус на Автозаводской
│   ├── buildings/         # Маршруты до корпусов
│   │   └── av-ab.mp4      # Маршрут до корпуса А
│   ├── floors/            # Маршруты по этажам
│   │   └── av-ab-02f.mp4  # Маршрут на 2 этаж в корпусе А
│   └── offices/           # Маршруты до кабинетов
│       └── av-ab-02f-0a213c.mp4 # Маршрут до кабинета 213 на 2 этаже корпуса А
├── bs/                    # Корпус на Большой Семёновской
├── mi/                    # Корпус на Михалковской
├── pk/                    # Корпус на Павла Корчагина
└── pr/                    # Корпус на Прянишкова
```