АРХИТЕКТУРА ПРОГРАММНОГО РЕШЕНИЯ

Заказчик:	ИП МАЖАЕВ ВЯЧЕСЛАВ СЕРГЕЕВИЧ, Мажаев Вячеслав Сергеевич
Название проекта:	SpeechMate
Исполнители	Куликова Татьяна Дмитриевна, Милорадова Ксения Сергеевна, Насыхова Анастасия Артемовна, Щербакова Елизавета Александровна

1) ОПИСАНИЕ И НАЗНАЧЕНИЕ ПРОГРАММЫ

Цель: разработать сайт и телеграм-бот, которые дают возможность пользователям масштабировать контент путем перевода речи на другой язык, а также упростить коммуникацию путем озвучивания и генерации краткого пересказа чатов.

Сайт должен предоставлять пользователю следующие функции:

1. Создавать транскрипт файла (видео или аудио);
2. Переводить содержимое файла (видео или аудио) на заданный язык;
3. Выбирать один из трех вариантов синтезирования речи в выходном файле: с сохранением пола говорящего, клонирование голоса, выбор из списка доступных.

Телеграм-бот должен предоставлять пользователю дополнительные функции для обработки не только файла, но и чата, такие как:

1. Создавать транскрипт переписки за выбранный период времени;
2. Озвучивать сообщения в групповом чате за выбранный период времени с одной из трех опций синтезирования речи;
3. Генерировать краткий пересказ переписки за выбранный период времени в виде текста или аудио.

2) Описание модели Жизненного Цикла проекта SpeechMate

Для реализации данного проекта можно использовать различные модели Жизненного Цикла (ЖЦ), но наиболее подходящей является инкрементная модель. Она предполагает поэтапную разработку продукта, в ходе которой каждая итерация (инкремент) включает в себя полный цикл работ от проектирования до тестирования и выпуска.

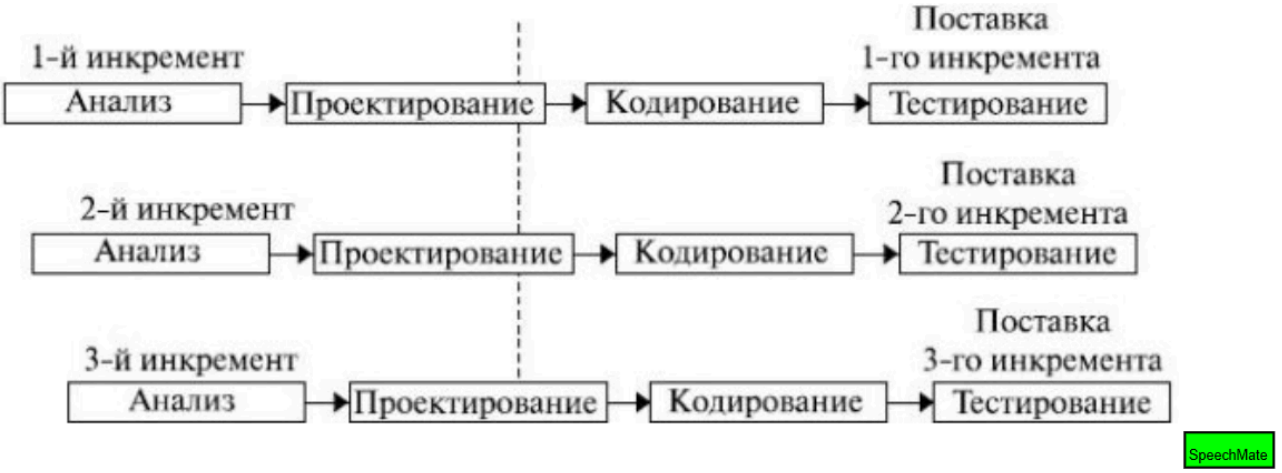

Рисунок 1: Инкрементная модель жизненного цикла разработки программного проекта

Преимущества инкрементной модели для проекта SpeechMate

- Быстрое выведение продукта на рынок. Инкрементная модель предполагает поэтапную реализацию проекта, начиная с основных функций и постепенно добавления новых. Это позволяет сократить время разработки и вывода продукта на рынок.
- Повышенная гибкость. Инкрементная модель позволяет легко вносить изменения в проект в соответствии с потребностями пользователей. Это делает проект более адаптивным к изменениям рынка.
- Простота управления. Данная модель жизненного цикла позволяет разбить проект на небольшие, управляемые итерации, что упрощает планирование и контроль проекта.

- Снижение рисков. Инкрементная модель предполагает тестирование проекта на каждой итерации, что позволяет выявить и устранить ошибки на ранних этапах и снизить риски провала проекта.

Обоснование выбора инкрементной модели ЖЦ

В данном проекте целью является быстрое выведение продукта на рынок. При этом совместно с заказчиком уже разработано техническое задание с фиксированным функционалом, который легко разбивается на инкременты. После реализации в случае успеха проекта, планируется его дальнейшее развитие за счет добавления новых функции. Инкрементная модель жизненного цикла идеально подходит под все эти аспекты, за счет чего и была выбрана.

Спиральная модель ЖЦ также могла быть использована для реализации данного проекта. Однако она предполагает более длительный и затратный процесс разработки за счет детальной проработки рисков, что не соответствует цели проекта.

Вывод

Инкрементная модель является хорошим выбором для проекта SpeechMate, так как позволяет быстро получить работающий продукт с возможностью внесения изменений в требования и функциональность продукта на протяжении всего жизненного цикла.

3) Описание используемых паттернов проектирования (Design Patterns)

Паттерн Singleton

Назначение и решаемая паттерном задача

Паттерн Singleton обеспечивает, чтобы у класса был только один экземпляр. В данном проекте паттерн Singleton используется для обеспечения того, чтобы у каждого пользователя был только один объект класса. Это необходимо для того, чтобы пользователи могли сохранять свои настройки и данные в приложении.

Общее описание паттерна и структура паттерна в виде UML-диаграммы классов проекта

UML-диаграмма классов для паттерна Singleton представлена на рисунке 2.

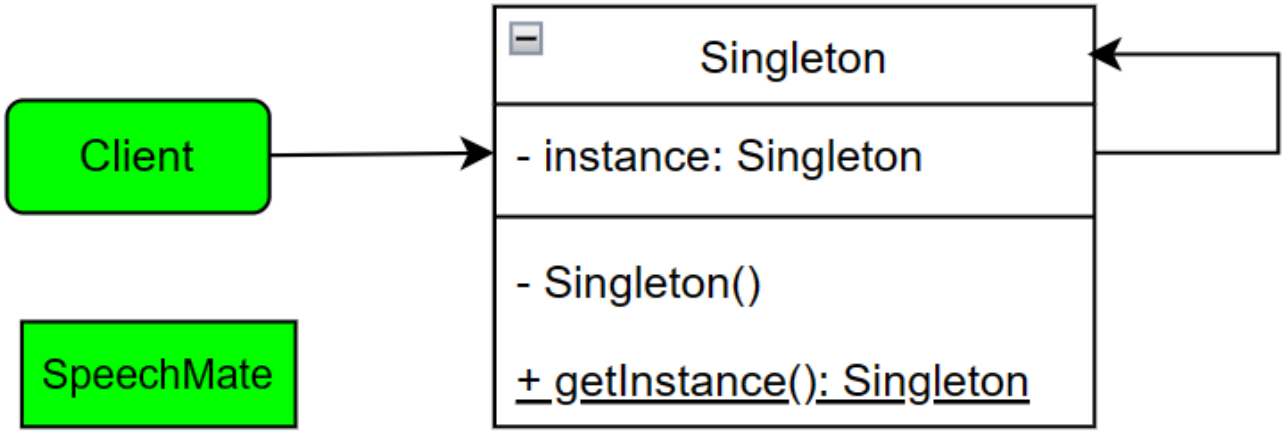

Рисунок 2: UML-диаграмма классов для паттерна Singleton

Последствия применения паттерна

Применение паттерна Singleton имеет следующие последствия:

- У класса может быть только один экземпляр;
- Доступ к экземпляру класса осуществляется через статический метод getInstance();
- Класс не может быть наследован.

Паттерн Facade

Назначение и решаемая паттерном задача

Паттерн Facade предоставляет простой интерфейс для доступа к более сложным компонентам системы. В данном проекте паттерн Facade используется для предоставления простого интерфейса для доступа к функциям транскрибирования, перевода и озвучивания.

Общее описание паттерна и структура паттерна в виде UML-диаграммы классов проекта

UML-диаграмма классов для паттерна Facade представлена на рисунке 3.

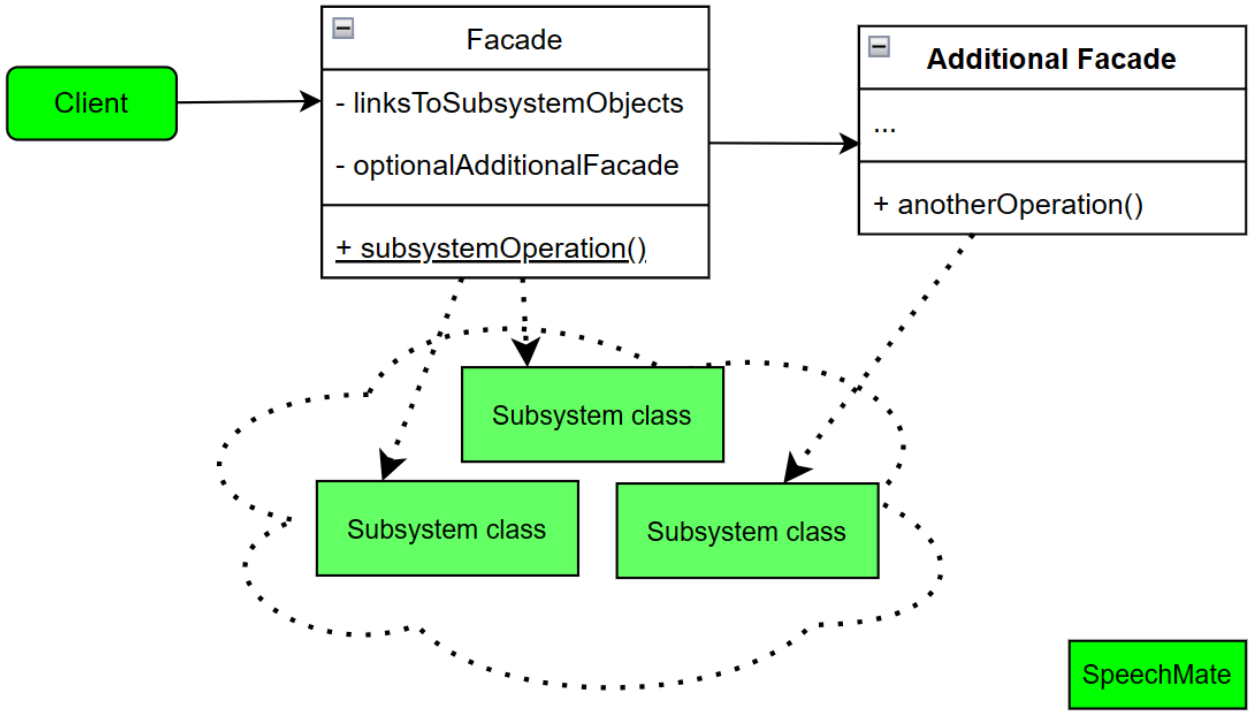

Рисунок 3: UML-диаграмма классов для паттерна Facade

Последствия применения паттерна

Применение паттерна Facade имеет следующие последствия:

- Упрощается использование более сложных компонентов системы;
- Скрывает детали реализации от пользователя;
- Обеспечивает единое представление для доступа к различным компонентам системы.

Паттерн Orchestrator

Назначение и решаемая паттерном задача

Паттерн Orchestrator предназначен для координирования и управления выполнением операций и сервисов в составном объекте. Он помогает разделить сложные бизнес-процессы на более простые компоненты и координировать их выполнение. В данном проекте применяется для оркестрации провайдеров, обеспечивающих озвучивание текста разными голосами и языками.

Общее описание паттерна и структура паттерна в виде UML-диаграммы классов проекта

UML-диаграмма классов для паттерна Orchestrator представлена на рисунке 4.

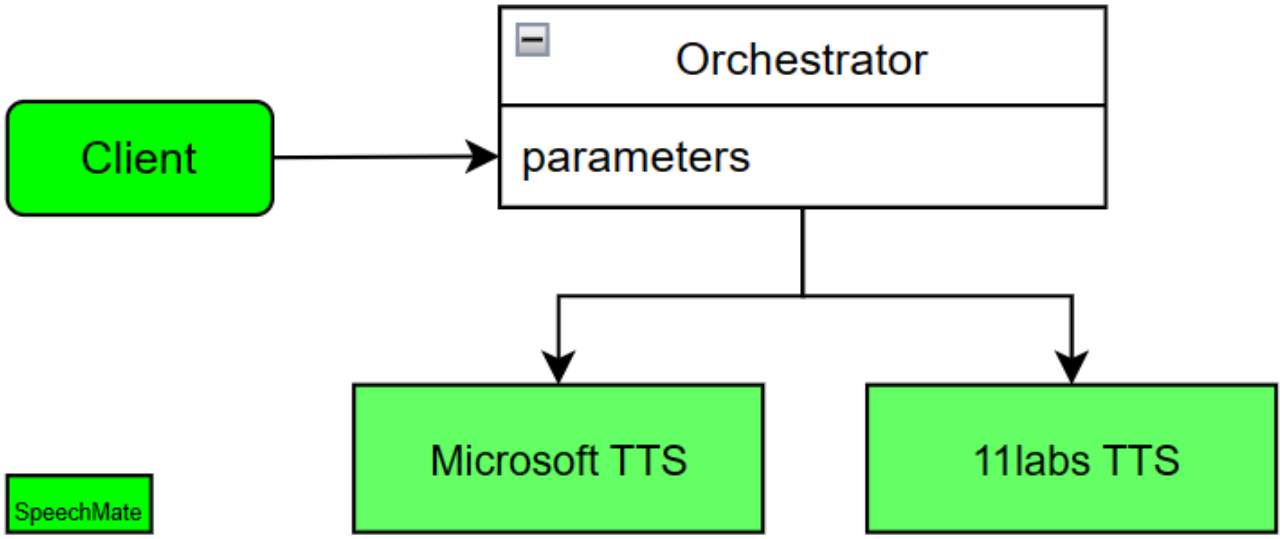

Рисунок 4: UML-диаграмма классов для паттерна Orchestrator

Последствия применения паттерна

Применение паттерна Orchestrator позволяет:

- Упростить и разделить сложные бизнес-процессы на более простые компоненты;
- Улучшить контроль над последовательностью выполнения операций;
- Уменьшить связанность между компонентами системы;
- Повысить гибкость и возможность повторного использования компонентов.

4) Описание типа архитектуры (MVC, Клиент-серверное, P2P, проч.)

Тип архитектуры: микросервисная

Описание типа архитектуры

Микросервисная архитектура является подходом к построению системы, в котором основная функциональность разделена на небольшие, автономные сервисы, каждый из которых выполняет свою специфическую задачу. Каждый микросервис может быть разработан, развернут и масштабирован независимо от других сервисов.

В системе SpeechMate сервисы можно подразделить на 2 типа: ML-сервисы и базовые.

Базовые

1. *Сервис авторизации Oauth*. Отвечает за регистрацию и аутентификацию пользователей в системе, выполняет функции gateway системы;
2. *Сервис оплаты Stripe*. Отвечает за оформление платной подписки на сайте и в телеграм-боте;
3. *Сервис Generator*. Отвечает за интеграцию всех ML-сервисов в одну API, а также за обработку запросов пользователей.
4. *Сервис TelegramAPI*. Отвечает за выгрузку сообщений из чата.

ML-сервисы

1. *Сервис OpenAI Whisper*. Отвечает за создание транскрипта файла и чата в текст.
2. *Сервис GPT-3.5-turbo от OpenAI*. Отвечает за перевод текста на заданный язык.
3. *Сервис MicrosoftTTS*. Отвечает за генерации речи по заданному тексту на заданном языке (более 80 вариантов).
4. *Сервис 11labs*. Отвечает за генерацию речи по заданному тексту на заданном языке (, предоставляет широкий спектр различных голосов.
5. *Сервис для определения пола говорящего*.
6. Сервис для создания краткого пересказа.

Схема архитектуры системы SpeechMate представлена на рисунке 5.

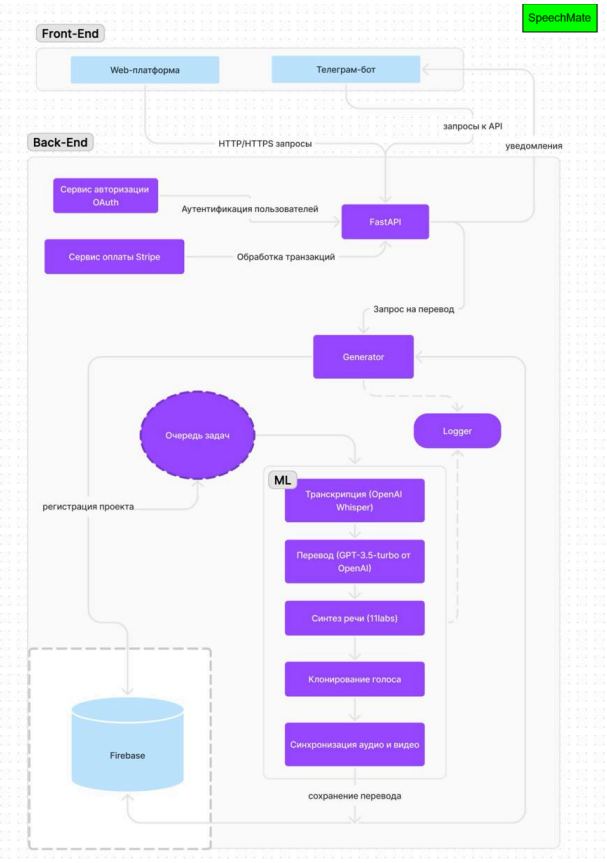

Рисунок 5: Архитектура системы SpeechMate

Обоснование выбора типа архитектуры

Ключевыми факторами при выборе типа архитектуры стали следующие преимущества микросервисов:

1. *Разделение функциональности*. Микросервисная архитектура позволяет разделить различные функции, такие как создание транскрипта, перевод и озвучивание сообщений, на отдельные сервисы. Это упрощает разработку и обслуживание каждой части системы.
2. *Гибкость и масштабируемость*. Микросервисы могут быть масштабированы независимо друг от друга, что позволяет отдельно масштабировать те компоненты, которые испытывают большую нагрузку. Это позволяет добиться более гибкой архитектуры, которая может легко реагировать на изменение требований и нагрузки.
3. *Легкая замена и модификация*: В случае необходимости изменить или заменить один из сервисов, это может быть сделано без необходимости модификации остальных компонентов системы. Это упрощает обновление и поддержку системы в долгосрочной перспективе.

Заказчик _____ / Мажаев В.С.

Ответственный по проекту _____ / Щербакова Е.А.

Дата: 29.12.2023