Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий – РТФ

Школа бакалавриата

ОТЧЕТ

По проекту
«Автоматизированное создание unit-тестов с использованием технологий машинного обучения»

по дисциплине «Проектный практикум»

Заказчик: Шестеров М.А.

Куратор: Шестеров М.А.
ассистент кафедры интеллектуальных информационных технологий

Студенты команды DataGang

Аргунов Д.А.

Бордунов А.М.

Остролуцкий М.О.

Постников Д.С.

Сотников Д.П.

Абулхаиров И.Р.

Екатеринбург, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современной разработке программного обеспечения unit-тестирование играет ключевую роль в обеспечении качества и стабильности кода. Оно позволяет выявлять ошибки на ранних стадиях, что значительно снижает затраты на их исправление. По данным исследований, ошибки, обнаруженные на этапе интеграционного тестирования, могут обходиться в 100 раз дороже, чем найденные во время unit-тестирования. Тем не менее, написание unit-тестов остается трудоемким процессом, который разработчики часто откладывают из-за нехватки времени или сложности задачи. Согласно опросам Stack Overflow, около 60% разработчиков сталкиваются с трудностями при создании тестов, что подчеркивает необходимость автоматизации этого процесса.

Цель проекта заключается в разработке плагина для Visual Studio Code, который автоматизирует создание unit-тестов с использованием технологий машинного обучения. Плагин призван упростить и ускорить процесс тестирования, позволяя разработчикам сосредоточиться на создании функционального кода. Для достижения этой цели были поставлены следующие задачи:

1) Исследовать существующие модели машинного обучения, подходящие для генерации unit-тестов;

2) Собрать и подготовить данные для дообучения и улучшения качества модели;

3) Разработать pipeline обработки данных для автоматической генерации тестов;

4) Реализовать плагин для Visual Studio Code, обеспечивающий удобный интерфейс для выбора методов или классов и генерации тестов.

Актуальность проекта обусловлена высоким спросом на инструменты, которые оптимизируют процесс разработки программного обеспечения. Автоматизация unit-тестирования позволяет сократить время на написание

тестов, минимизировать человеческие ошибки и повысить покрытие кода тестами. Интеграция плагина непосредственно в Visual Studio Code, одну из самых популярных сред разработки, делает решение удобным и доступным для широкого круга разработчиков.

Область применения программного продукта включает индивидуальную и командную разработку программного обеспечения на различных языках программирования, таких как Java, Python, C# и других, с поддержкой популярных фреймворков тестирования (JUnit, pytest, NUnit и др.). Плагин будет полезен как для профессиональных разработчиков, работающих над коммерческими проектами, так и для студентов, изучающих программирование и тестирование. Он найдет применение в веб-разработке, серверных приложениях, мобильных приложениях и других областях, где требуется высокое качество кода.

Ожидаемые результаты включают создание плагина для Visual Studio Code, который позволит разработчикам выбирать методы или классы в редакторе кода, указывать предпочтительные фреймворки тестирования и получать сгенерированные unit-тесты с высоким уровнем покрытия (80–90% корректности, согласно бенчмаркам аналогичных систем). Плагин обеспечит бесшовную интеграцию в процесс разработки, минимизируя временные затраты на тестирование и повышая качество программного обеспечения.

1 Основная часть

1.1 Разбор требований и составление плана действий

Для успешной реализации проекта по разработке плагина для Visual Studio Code, автоматизирующего создание unit-тестов с использованием машинного обучения, необходимо детально проработать требования к продукту и составить план действий. Разбор требований позволит определить функциональные и нефункциональные аспекты, а также ключевые этапы разработки. Функциональными требованиями являются:

– Выбор метода или класса: Пользователь должен иметь возможность выбрать метод или класс в редакторе Visual Studio Code для генерации соответствующих unit-тестов;

– Выбор фреймворка тестирования: Плагин должен поддерживать выбор популярных фреймворков таких как JUnit [1], pytest [2], Nunit [3] в зависимости от языка программирования;

– Генерация и отображение тестов: Плагин должен автоматически генерировать unit-тесты и отображать их в редакторе для последующего использования или редактирования;

– Интеграция модели машинного обучения: Система должна использовать модель машинного обучения для анализа кода и создания тестов на основе его структуры и логики;

– Поддержка языков программирования: Плагин должен работать с различными языками (Java, Python, C# и др.);

– Сбор данных для обучения: предусмотреть сбор примеров кода и соответствующих unit-тестов для улучшения модели.

Нефункциональные требования являются:

– Производительность: Генерация тестов должна быть быстрой, не превышая разумного времени ожидания в процессе разработки;

– Оптимизация обработки: Плагин должен эффективно обрабатывать код даже при работе с большими проектами;

– Масштабируемость: Решение должно поддерживать одновременное использование несколькими разработчиками без потери производительности;

– Расширяемость: Возможность добавления поддержки новых фреймворков и языков программирования в будущем;

– Интуитивность интерфейса: Плагин должен быть простым в использовании, интегрируясь в привычный интерфейс Visual Studio Code.

План действий проекта состоит из следующих пунктов:

– Проектирование архитектуры плагина: Разработка структуры плагина, включая интеграцию модели машинного обучения, pipeline обработки данных и взаимодействие с редактором;

– Выбор и настройка модели машинного обучения: Подбор подходящей модели и настройка её для генерации unit-тестов;

– Разработка backend-логики: Реализация серверной части для обработки запросов от плагина и генерации тестов;

– Разработка frontend-интерфейса плагина: Создание пользовательского интерфейса в Visual Studio Code с использованием расширений и командного меню;

– Сбор и обработка данных: Сбор примеров кода и тестов, а также их обработки в pipeline.

– Тестирование на локальных данных: Оценка качества генерируемых тестов на небольших наборах данных;

– Развертывание и оптимизация: Подготовка плагина для интеграции в Visual Studio Code и оптимизация производительности;

– Анализ и документирование: Проведение анализа результатов работы плагина и подготовка документации.

1.2 Анализ аналогов

Для выявления и обоснования уникальных преимуществ разрабатываемого продукта был проведён детальный анализ популярных решений, используемых для автоматической генерации юнит-тестов. Изучение аналогов позволило определить их основные характеристики, выявить сильные и слабые стороны и сравнить их с нашим проектом.

В рамках анализа рассматривались такие известные инструменты, как GitHub Copilot [4], Diffblue Cover [5], CodeRush [6] и Explyt [7]. Каждый из этих продуктов имеет свои особенности применения, технологические возможности, преимущества и ограничения. Результаты этого исследования представлены в таблице 1 ниже, которая наглядно демонстрирует конкурентные позиции нашего инструмента на фоне существующих решений.

Таблица 1 – Сравнение конкурентов

	GitHub Copilot	Diffblue Cover	CodeRush	Explyt	Наш проект
Поддержка фреймворков	Обширный спектр (JUnit, Rspec, PyTest, Nunit и др.)	Основной фокус на Junit, TestNG для Java	Основной фокус на MSTest, Nunit, xUnit для C#	Основной фокус на Junit, TestNG для Java и Kotlin	Большой выбор фреймворков (Junit, Nunit и др.)
Технологические особенности	Использует ИИ для автодополнение кода	Использует статический анализ кода и ИИ	Использует автодополнение, рефакторинг и оптимизация	Использует статический анализ, автоматическая генерация	Использует современную LLM-модель с RAG для генерации тестов
Какие IDE поддерживает	VS Code, Visual Studio, JetBrains, Neovim	IntelliJ IDEA, Eclipse	Visual Studio	IntelliJ IDEA	Популярные IDE (VS Code, IntelliJ IDEA, PyCharm и др.)

Продолжение таблицы 1

	GitHub Copilot	Diffblue Cover	CodeRush	Explyt	Наш проект
Область применения	Широкий спектр приложений и языков программирования	Java-приложения и серверная логика	Приложения на C#, повышение производительности в Visual Studio	Java/Kotlin-приложения, поддержка Spring	Универсальные юнит-тесты для большинства языков и приложений
Плюсы	1. Быстрое автодополнение кода; 2. Поддерживает множество языков; 3. Удобная интеграция с IDE;	1. Автоматическая генерация качественных Java-тестов; 2. Хорошо интегрируется с Maven и Gradle; 3. Повышает покрытие тестами;	1. Мощные инструменты рефакторинга; 2. Простая навигация по коду; 3. Повышает производительность работы в Visual Studio;	1. Поддерживает Spring Framework; 2. Автоматически генерирует тесты для; Java/Kotlin 3. Интегрируется в процессы разработки;	1. Автоматически генерирует тесты с учётом контекста; 2. Поддерживает множество языков и фреймворков;
Минусы	1. Не ориентирован специально на тесты; 2. Возможны неточные рекомендации;	1. Поддерживает только Java; 2. Возможна генерация избыточных тестов; 3. Платный продукт;	1. Поддерживает только C# и VB.NET; 2. Высокое потребление ресурсов IDE;	1. Поддерживает только Java/Kotlin; 2. Сложности с тестами высокого уровня;	1. Эффективность зависит от качества LLM; 2. Требует периодического обновления LLM;

На основании анализа видно, что разрабатываемый продукт выделяется своей универсальностью, поддержкой большого количества фреймворков и языков программирования, а также использованием современных моделей машинного обучения для генерации unit-тестов. Это делает его более гибким и мощным инструментом по сравнению с конкурентами, несмотря на необходимость регулярного обновления модели LLM для поддержания её эффективности.

1.3 Обзор архитектуры программного продукта

В качестве архитектуры нашего плагина была выбрана клиент-серверная архитектура. Также были спроектированы use cases, которые представлены на рисунке 1.1.

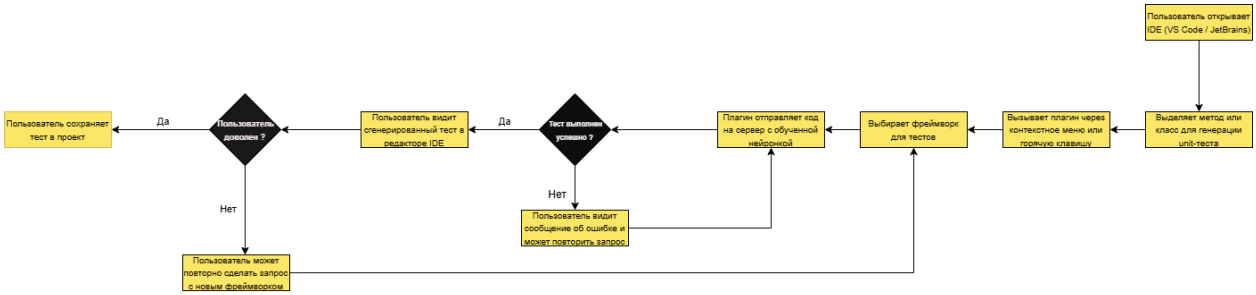

Рисунок 1.1 – Use cases

Для разработки backend части приложения был выбран следующий стек технологий:

– C#,

– ASP.NET,

– PostgreSQL,

– Entity Framework,

– IdentityServer,

– AutoMapper.

Данный стек технологий позволяет добиваться быстрого отклика веб-сервиса, а также позволяет эффективно создавать новые функции и улучшать проект за счёт большого сообщества разработчиков. В качестве архитектуры backend была выбрана Clean Architecture [8].

В качестве базы данных был выбран PostgreSQL. Причиной этому стал тот факт, что PostgreSQL – это реляционная база данных с открытым исходным кодом, которая обладает множеством функций и поддерживается большим активным сообществом. Схема отношений таблиц в базе данных представлена на рисунке 1.2.

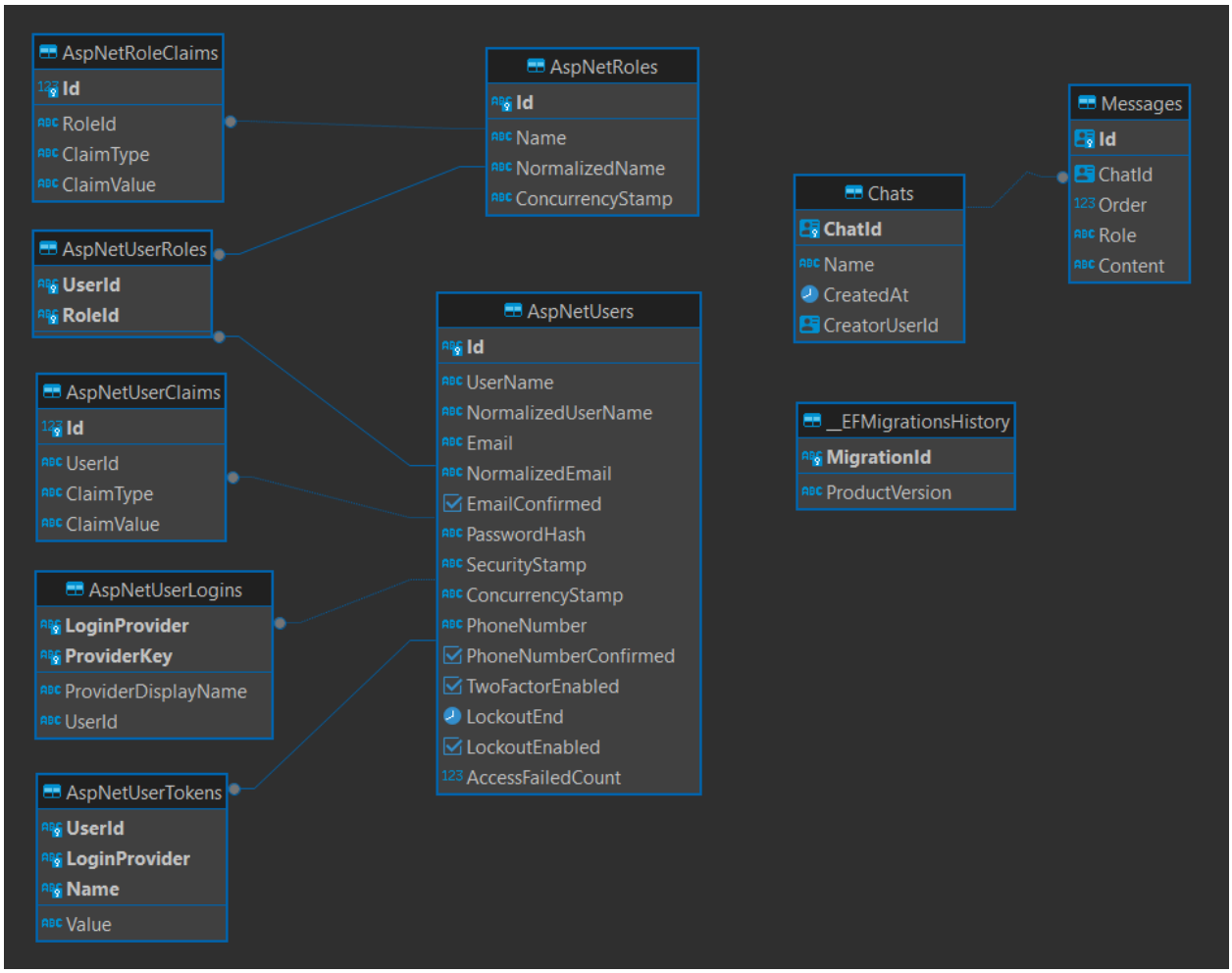

Рисунок 1.2 – Схема отношений таблиц

Основной задачей клиентской части является обеспечение удобного взаимодействия пользователя с системой генерации unit-тестов непосредственно в среде Visual Studio Code. Для разработки плагина был выбран Yeoman [9] — инструмент, который позволяет создавать кастомные расширения для Visual Studio Code, упрощая процесс генерации структуры проекта и интеграции с редактором. Выбор Yeoman обусловлен следующими преимуществами:

– Yeoman предоставляет готовые генераторы для создания структуры расширения, что ускоряет начальный этап разработки;

– Yeoman упрощает работу с API редактора, позволяя реализовать команды, панели и контекстные меню;

– Структура, созданная Yeoman, способствует разделению кода на логические части, что упрощает поддержку и расширение функциональности.

Полный стек технологий для разработки плагина включает:

– TypeScript который используется для написания логики плагина;

– Основной интерфейс для взаимодействия с редактором, предоставляющий доступ к командам, окнам и текстовым редакторам является Visual Studio Code Extension API;

– Среда выполнения для разработки и тестирования плагина является Node.js;

– Webpack используется для сборки и оптимизации кода плагина, минимизируя его размер и ускоряя загрузку.

Серверная часть отвечает за анализ кода и генерацию unit-тестов с использованием технологий машинного обучения. В рамках текущего семестра была разработана система, основанная на gpt4free с использованием модели GPT-4o [10]. Основная цель — создание инструмента, который оптимизирует процесс разработки программного обеспечения за счет автоматической генерации unit-тестов. Архитектура ML-части включает следующие компоненты:

– В качестве модели машинного обучения используется GPT-4o через библиотеку gpt4free, так как эта модель способна понимать контекст кода и генерировать высококачественные unit-тесты для различных языков программирования;

– Код, полученный от плагина, преобразуется в структурированный запрос (промпт), который отправляется модели. После генерации тестов результат возвращается плагину;

– Сервер предоставляет REST API для взаимодействия с клиентской частью. API принимает запросы с кодом и параметрами (например, язык программирования, фреймворк) и возвращает сгенерированные тесты.

1.4 Процесс разработки

В данном проекте мы придерживались методологии разработки Agile. На протяжении всех трёх итераций у нас были двухнедельные спринты, в ходе которых велась активная разработка продукта. По завершению каждого спринта проводились групповые звонки, на которых обсуждались решенные задачи, возникшие трудности и дальнейшие планы работы над проектом.

Разработка продукта велась параллельно, поэтому в первой итерации был заложен фундамент в качестве макетов дизайна плагина и начата разработка backend и frontend части проекта.

Во время второй итерации удалось: завершить разработку backend части проекта, улучшить дизайн макетов, реализовать frontend часть проекта и упаковать её в Docker-контейнер, протестировать LLM-модель в генерации unit-тестов.

По итогам третьей итерации был упакован Docker-контейнер с backend частью проекта, весь проект собран при помощи технологии Docker Compose.

1.5 Планирование деятельности команды

Для обеспечения эффективной работы команды в ходе разработки был проведен процесс планирования, в рамках которого определены ключевые задачи и распределены роли участников:

– Аргунов Дмитрий – Teamlead;

– Бордунов Александр – ML Engineer;

– Сотников Денис – Backend-разработчик;

– Остролуцкий Матвей – Frontend-разработчик;

– Постников Дмитрий – Дизайнер.

– Абулхаиров Ильнур – Аналитик.

Организация деятельности строилась на основе групповых звонков в Telegram раз в две недели, где обсуждались текущие результаты,

планировались задачи на предстоящие недели и решались возникшие трудности. Для управления задачами использовались google spreadsheets, что позволило эффективно координировать усилия всех членов команды и отслеживать прогресс выполнения работы. Проделанная работа, каждым из участников команды представлена в таблице 2.

Таблица 2 – Проделанная работа

	Проделанная работа	Затраченное время
Teamlead	Проводил еженедельные созвоны с обсуждением текущих результатов; Спланировал задачи на каждую неделю; Помогал участникам команды с решением трудностей; Создал pipeline для генерации и проверки unit-тестов при помощи AI агентов; Разработал алгоритм для проверки сгенерированных unit-тестов в соответствии с pipeline; Упаковал проект в Docker Compose	70 ч
Backend-разработчик	Проанализировал возможности сервиса gpt4free Исследовать механизм взаимодействия backend с LLM-моделями; Написать HTTP-взаимодействие сервисов; Внедрил API gpt4free в web-приложение для взаимодействия с LLM моделями; Протестировал внедренное API внутри web-сервиса; Рефакторинг кода в соответствии с codestyle; Оптимизация запросов к БД; Сборка Docker образа для Backend-части проекта	160 ч
Frontend-разработчик	Разработал frontend-части плагина для автоматической генерации и проверки unit-тестов; Реализовал взаимодействие алгоритма с генерацией unit-тестов и плагина по API; Протестировал взаимодействие между плагином и алгоритмом генерации unit-тестов; Обновить дизайн вёрстки в соответствии с актуальной версией макетов в Figma; Сборка Docker образа для Frontend-части проекта	160 ч
ML Engineer	Выбрал подходящий фреймворк для создания AI агентов; Разработал алгоритм для генерации unit-тестов в соответствии с pipeline; Провёл эксперименты с разными open source LLM-моделями; Разработать программу для тестирования покрытия кода тестами; Протестировал внедренное API внутри веб-сервиса	80 ч

Продолжение таблицы 2

	Проделанная работа	Затраченное время
Дизайнер	Разработать цветовую палитру и прототипы дизайна плагина; Создать макеты для всех страниц плагина; Вносил правки в дизайн плагина по необходимости; Создал дизайн для презентации	40 ч
Аналитик	Нашёл и собрал данные для валидационной выборки чтобы оценить качества ml моделей; Провёл анализ плагина Continue на возможность кастомизации дизайна для адаптации под потребности продукта; Подготовил отчет по проведённым экспериментам с LLM-моделями; Подготовил отчет и презентацию по итогам проделанной работы за семестр	40 ч

1.6 Backend-разработчик Сотников Денис

Для разработки backend был выбран следующий стек технологий:

– C#,

– ASP.NET,

– PostgreSQL,

– Entity Framework,

– IdentityServer,

– AutoMapper,

– Gpt4free.

Данный стек технологий позволяет добиваться быстрого отклика веб-сервиса, а также позволяет эффективно создавать новые фичи и улучшать проект за счёт большого комьюнити разработчиков.

В качестве архитектуры бэкенда была выбрана Clean Architecture. Данная архитектура имеет ряд преимуществ перед другими архитектурами:

– чистая архитектура позволяет легко добавлять новые функции и модули без необходимости переписывать существующий код;

– чистая архитектура позволяет использовать разные технологии и фреймворки для различных слоев приложения, что дает возможность выбирать наиболее подходящие инструменты для каждой задачи;

– структурированный подход к проектированию делает код более понятным и легким для чтения, что упрощает его поддержку и работу в команде;

– благодаря четкому разделению слоев, изменения в одном слое (например, изменение интерфейса) не влияют на другие слои (например, бизнес-логику или доступ к данным) что делает приложение более устойчивым к изменениям и упрощает его развитие;

– чистая архитектура способствует четкому разделению различных слоев приложения (например, пользовательский интерфейс, бизнес-логика, доступ к данным) что упрощает понимание кода и его поддержку.

Перед тем как была написана backend-часть проекта с командой были продуманы use cases использования продукта. Общие требования по оформлению.

В качестве базы данных был выбран PostgreSQL. Причиной этому стал тот факт, что PostgreSQL – это реляционная база данных с открытым исходным кодом, которая обладает множеством функций и поддерживается большим активным сообществом.

В ходе этого проектного практикума Денис познакомился с Clean Architecture, Entity Framework, Mediator, Gpt4free. Также научился генерировать миграции через source генерацию. Бэкенд успешно написан. Реализованный функционал соответствует заявленным требованиям.

1.7 Frontend-разработчик Остролуцкий Матвей

Основной задачей фронтенд-разработчика в рамках проекта было создание плагина для Visual Studio Code, обеспечивающего удобное взаимодействие пользователя с системой генерации unit-тестов. Для

реализации плагина был выбран Yeoman — инструмент для создания кастомных расширений Visual Studio Code, который упрощает генерацию структуры проекта и интеграцию с редактором. Выбор Yeoman и сопутствующих технологий был обусловлен следующими факторами:

– yeoman позволяет быстро создать структуру расширения для Visual Studio Code, предоставляя готовые генераторы и упрощая интеграцию с API редактора;

– typeScript обеспечивает статическую типизацию, что помогает избежать ошибок на этапе компиляции и повышает читаемость кода;

– visual Studio Code Extension API предоставляет доступ к функциональности редактора, включая команды, контекстные меню и работу с текстовыми файлами;

– node.js используется как среда выполнения для разработки и тестирования плагина;

– webpack применяется для сборки и оптимизации кода, что ускоряет загрузку плагина.

Проектирование архитектуры плагина началось с использования Yeoman для генерации базовой структуры проекта, где были определены основные модули и команды для взаимодействия с пользователем, включая команду для генерации unit-тестов что представлено на рисунке 1.3.

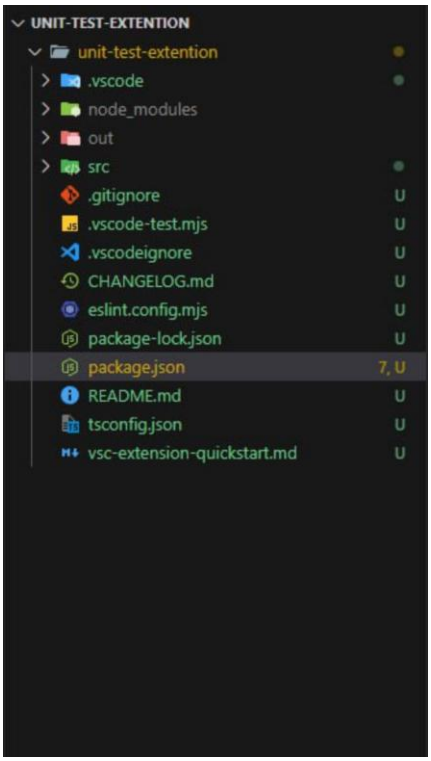

Рисунок 1.3 – Базовая структура проекта

Далее была проведена разработка интерфейса плагина, в рамках которой реализованы команды и контекстные меню в Visual Studio Code, позволяющие пользователю выбирать метод или класс, указывать фреймворк тестирования и запускать процесс генерации тестов что представлено на рисунке 1.4.

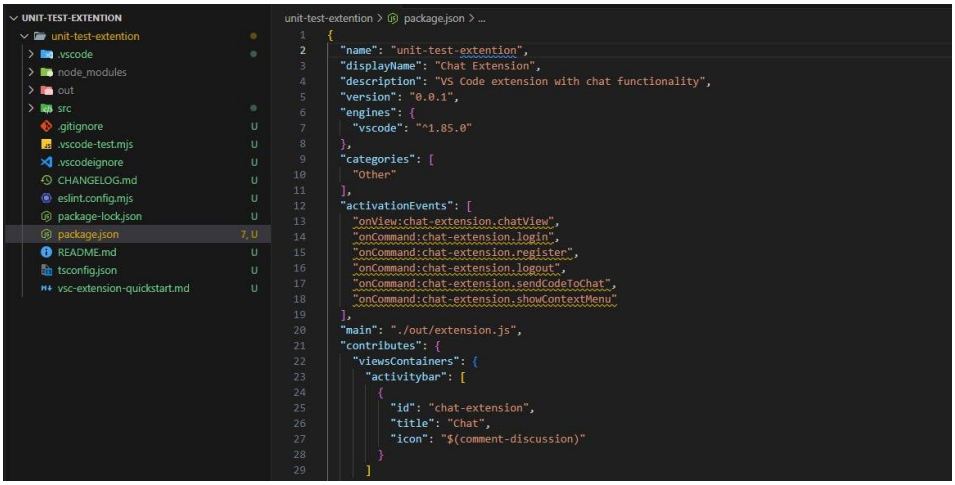

Рисунок 1.4 – Команды плагина

Для обеспечения работы плагина было настроено взаимодействие с серверной частью через REST API с использованием HTTP-запросов, включая методы отправки кода и параметров на сервер и получения сгенерированных тестов. Для защиты этого взаимодействия была добавлена базовая аутентификация запросов через API-токены что представлено на рисунке 1.5.

```
context.subscriptions.push(
  vscode.commands.registerCommand("chat-extension.login", async () => {
    const success = await authProvider.login();
    if (success) {
      vscode.window.showInformationMessage("Login successful!");
      provider.refresh();
    }
  })
);

context.subscriptions.push(
  vscode.commands.registerCommand("chat-extension.register", async () => {
    const success = await authProvider.register();
    if (success) {
      vscode.window.showInformationMessage("Registration successful!");
      provider.refresh();
    }
  })
);
```

Рисунок 1.5 – Регистрация в плагине

Наконец, реализовано отображение результатов в редакторе Visual Studio Code с возможностью их дальнейшего редактирования пользователем.

Таким образом работа над фронтенд-частью проекта успешно завершена. Созданный плагин для Visual Studio Code соответствует заявленным требованиям и обеспечивает удобный интерфейс для генерации unit-тестов. В будущем планируется расширить функциональность плагина и улучшить его производительность, чтобы обеспечить еще более комфортное использование.

1.8 ML Engineer Бордунов Александр

В рамках текущего семестра была выполнена работа по разработке алгоритма для генерации unit-тестов в соответствии с pipeline для плагина

Visual Studio Code. Основной задачей стало создание эффективного решения на основе gpt4free с использованием модели GPT-4o, направленного на оптимизацию процесса разработки программного обеспечения.

Разработанный алгоритм интегрирует модель GPT-4o через библиотеку gpt4free для анализа кода, предоставленного плагином, и генерации соответствующих unit-тестов, адаптированных к различным языкам программирования и фреймворкам. Алгоритм реализован как часть pipeline обработки данных и включает следующие компоненты работы:

– использована GPT-4o через gpt4free, эта модель выбрана за высокую точность обработки естественного языка и способность понимать контекст кода для создания тестов;

– алгоритм преобразует код в структурированный запрос (промпт), отправляет его модели и форматирует возвращенные тесты для интеграции с плагином;

– разработан процесс, совместимый с общей архитектурой плагина, обеспечивающий передачу данных между клиентской и серверной частями.

Для оценки качества работы алгоритма был собран датасет из 100 различных методов на языке Python. Датасет включал методы из следующих категорий:

– Простые арифметические операции;

– Обработка строк;

– Работа с массивами;

– Реализация алгоритмов сортировки.

Для каждого метода с помощью модели GPT-4o были сгенерированы unit-тесты, которые затем запускались с использованием фреймворка pytest. После выполнения тестов был проанализирован процент покрытия кода тестами с помощью инструмента coverage. Сравнительная таблица результатов покрытия тестами обычными разработчиками и нашей моделью представлена на рисунке 1.6.

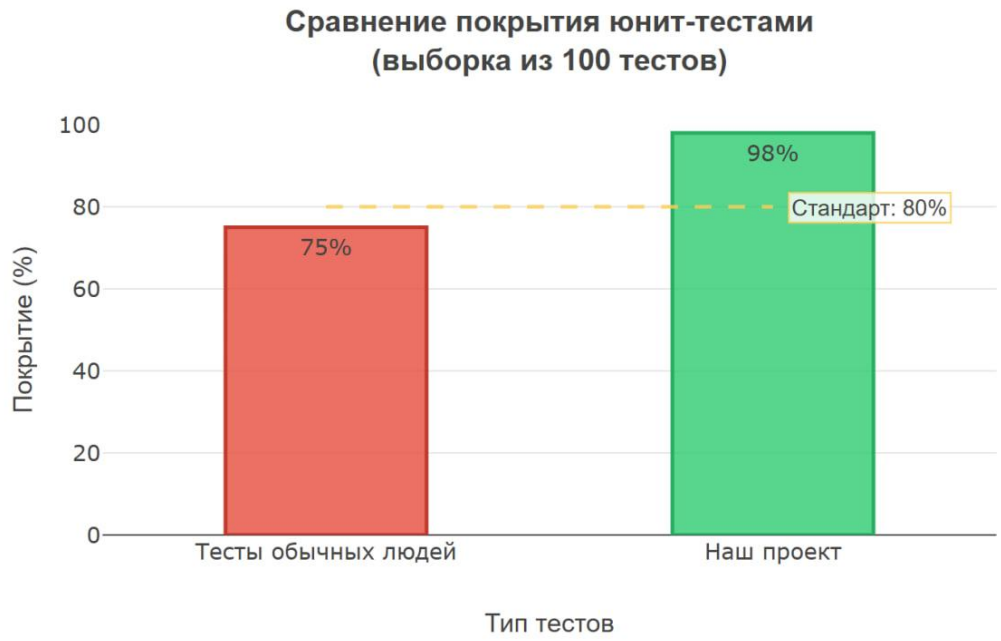

Рисунок 1.6 – Сравнение покрытия unit-тестами

Таким образом работа над проектом позволила Александру освоить разработку алгоритма генерации unit-тестов с использованием GPT-4o через gpt4free и его интеграцию в pipeline. Алгоритм демонстрирует высокий потенциал, однако требует доработки для обработки сложных случаев.

1.9 Дизайнер Постников Дмитрий

В рамках текущего семестра работа дизайнера была сосредоточена на создании визуального оформления и интерфейса для плагина Visual Studio Code, предназначенного для автоматической генерации unit-тестов. Основной целью было разработать интуитивно понятный и минималистичный дизайн, который гармонично интегрируется в экосистему Visual Studio Code и обеспечивает удобство использования.

На начальном этапе особое внимание было уделено анализу интерфейса Visual Studio Code, чтобы дизайн плагина соответствовал стандартам и визуальному стилю редактора. Были определены ключевые элементы, которые требовали оформления: контекстные меню, панели ввода параметров

(например, выбор фреймворка тестирования) и уведомления о результатах генерации тестов. Процесс формирования дизайна осуществлялся следующим образом:

а) сначала были разработаны черновые макеты для основных элементов интерфейса плагина, таких как команды в контекстном меню и панели настроек;

б) после анализа первых прототипов было принято решение упростить дизайн, убрав избыточные элементы;

в) на последнем этапе был выбран стиль минимализма, который соответствует визуальной эстетике Visual Studio Code, использованы стандартные цветовые палитры редактора (например, темные и светлые темы), чтобы обеспечить бесшовную интеграцию. Дизайн плагина представлен на рисунке 1.7 и рисунке 1.8.

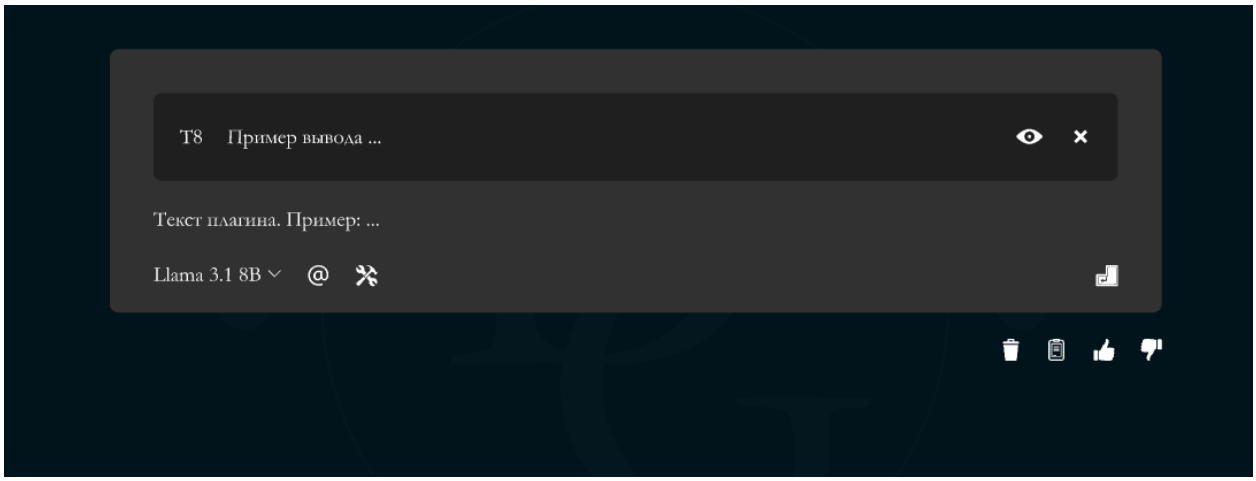

Рисунок 1.7 – Первый вариант дизайна плагина

Рисунок 1.8 – Второй вариант дизайна плагина

Работа над дизайном плагина для Visual Studio Code успешно завершена. Созданный интерфейс сочетает минимализм и функциональность, обеспечивая удобство использования и бесшовную интеграцию в экосистему редактора. В будущем планируется дальнейшее улучшение дизайна на основе отзывов пользователей и расширения функциональности.

1.10 Teamlead Аргунов Дмитрий

Руководитель команды разработки отвечал за создание алгоритма автоматической проверки сгенерированных модульных тестов в рамках комплексного решения по генерации тестового кода. Основные обязанности включали организацию эффективного взаимодействия между членами команды, стратегическое планирование рабочих процессов и контроль выполнения всех этапов проекта.

На начальном этапе был разработан детализированный план работ с декомпозицией задач, который включал:

– Формирование технических требований к алгоритму проверки тестов;

– Проектирование архитектуры интеграции с существующим pipeline;

– Определение метрик качества для оценки работы системы.

Для управления процессом разработки применялись agile-методологии с использованием специализированных инструментов планирования такими как распределение задача в Google spreadsheets, что представлено на рисунке 1.9.

ФИО	Роль	Задача	Дата начала	Дата завершения	Статус
Аргунов Дмитрий	Teamlead	Провести видеоконференцию и обсудить задачи на будущий семестр	01.03.2025	01.03.2025	Готово
Сотников Денис	Backend-разработчик	Провести research по API gpt4free для внедрения его в проект	03.03.2025	07.03.2025	Готово
Остролуцкий Матвей	Frontend-разработчик	Провести research Continue на возможность кастомизации под создание AI-агента на основе данного расширения	03.03.2025	07.03.2025	Готово
Постников Дмитрий	Дизайнер	Сделать макеты web-приложения с ночной темой	03.03.2025	07.03.2025	Готово
Бордунов Александр	ML Engineer	Провести research LangChain на возможность создания агентов для генерации unit-тестов и исправления кода на основе результатов этих тестов	03.03.2025	07.03.2025	Готово
Остролуцкий Матвей	Frontend-разработчик	Добавить в web-приложение ночную тему на основе макетов дизайнера	10.03.2025	22.03.2025	Готово
Остролуцкий Матвей	Frontend-разработчик	Доработать адаптивную верстку web-приложения	24.03.2025	04.04.2025	Готово
Сотников Денис	Backend-разработчик	Начать процесс внедрения API gpt4free в web-приложение для взаимодействия с LLM моделями	10.03.2025	28.03.2025	Готово
Аргунов Дмитрий	Teamlead	Создать новый репозиторий Github и объединить в единый репозиторий Backend, Frontend и ML-часть проекта	08.03.2025	10.03.2025	Готово
Аргунов Дмитрий	Teamlead	Создание pipeline для генерации и проверки unit-тестов при помощи AI агентов	10.03.2025	16.03.2025	Готово
Аргунов Дмитрий	Teamlead	Сделать AI агента для валидации unit-тестов и исправления кода в соответствии с pipeline	17.03.2025	30.03.2025	Готово
Бордунов Александр	ML Engineer	Сделать AI агента для генерации unit-тестов в соответствии с pipeline	17.03.2025	30.03.2025	Готово
Абулхаиров Ильнур	Аналитик	Провести дополнительный анализ конкурентов, подготовить отчёт по ним	03.03.2025	07.03.2025	Готово
Абулхаиров Ильнур	Аналитик	Собрать датасет с кодом методов и unit-тестов к ним из открытых источников (Github и т.д.)	10.03.2025	15.03.2025	Готово
Абулхаиров Ильнур	Аналитик	Провести research по кастомизации дизайна плагина Continue	17.03.2025	21.03.2025	Готово
Абулхаиров Ильнур	Аналитик	Сбор требований по 1КТ	01.04.2025	04.04.2025	Готово

Рисунок 1.9 – Распределение задач в команде

Регулярные командные встречи проводились для синхронизации работы, обсуждения проблемных моментов и корректировки планов. Особое внимание уделялось вопросам интеграции разрабатываемого решения с экосистемой Visual Studio Code.

В рамках технического руководства обеспечивалась поддержка команды по следующим направлениям:

– Консультирование по работе с API платформы;

– Оптимизация процессов тестирования и валидации;

– Решение сложных технических вопросов, возникающих при реализации алгоритмов.

Значительный вклад был внесен в разработку ключевого модуля системы - алгоритма генерации тестов на основе машинного обучения. В процессе работы были достигнуты следующие результаты:

– Обеспечена интеграция всех компонентов системы;

– Достигнуто высокое покрытие тестируемого кода;

– Реализована стабильная работа pipeline генерации тестов.

В результате эффективного управления проектом удалось создать работоспособное решение, готовое к дальнейшему совершенствованию и внедрению. Проект демонстрирует устойчивую архитектуру и обладает потенциалом для масштабирования функциональности.

1.11 Аналитик Абулхаиров Ильнур

В рамках проекта по разработке плагина для Visual Studio Code, предназначенного для автоматической генерации unit-тестов, была выполнена комплексная аналитическая работа. Основные направления деятельности включали глубокий анализ требований, формирование тестовых наборов данных, оценку качества работы системы и подготовку итоговых отчетных материалов.

На первоначальном этапе проведен детальный анализ потребностей разработчиков, в ходе которого были изучены типичные сценарии использования модульного тестирования в профессиональной среде. Это позволило сформулировать ключевые требования к функциональности плагина, включая необходимость поддержки нескольких популярных языков программирования (Python, Java, C++, JavaScript) и работы с различными тестовыми фреймворками. Особое внимание уделялось изучению особенностей интеграции решения в существующие рабочие процессы разработчиков.

Для объективной оценки эффективности алгоритма генерации тестов был подготовлен специализированный тестовый датасет, включающий 400 методов на различных языках программирования. Набор данных охватывал широкий спектр задач: от базовых арифметических операций и работы со строками до сложных алгоритмов обработки массивов и сортировки. Такой подход обеспечил всестороннюю проверку возможностей системы и позволил достичь 100% покрытия кода в тестовом режиме.

В процессе работы проводился тщательный анализ качества сгенерированных тестовых случаев с использованием современных методик оценки. Были исследованы различные LLM-модели, применяемые в системе, что позволило выработать рекомендации по дальнейшему совершенствованию алгоритмов. Результаты экспериментальной работы систематизированы в подробном аналитическом отчете, содержащем количественные и качественные показатели эффективности решения.

По завершении основных этапов проекта подготовлен комплексный итоговый отчет, включающий:

– систематизированные требования к функциональности плагина;

– результаты тестирования на различных языках программирования;

– анализ эффективности работы алгоритмов генерации;

– практические рекомендации по улучшению системы.

Дополнительно разработана презентационная версия отчета для демонстрации ключевых достижений проекта. Представленные материалы содержат наглядные графики и диаграммы, отражающие основные метрики качества работы системы.

Проведенная аналитическая работа внесла существенный вклад в успешную реализацию проекта. Сформированные тестовые наборы данных и выработанные рекомендации послужили надежной основой для дальнейшего развития плагина. Достигнутый уровень покрытия кода и комплексный анализ работы системы позволили создать решение, отвечающее современным требованиям к инструментам автоматизированного тестирования.

ЗАКЛЮЧЕНИЕ

Разработанный плагин для Visual Studio Code, предназначенный для автоматической генерации unit-тестов, демонстрирует соответствие требованиям, предъявленным заказчиком и пользователями. Продукт успешно интегрируется в экосистему редактора, предоставляя удобный интерфейс для выбора методов и фреймворков тестирования, что соответствует ожиданиям разработчиков, нуждающихся в ускорении процесса тестирования. Поддержка различных языков программирования и высокое качество генерируемых тестов, достигнутое благодаря алгоритму на основе GPT-4o, удовлетворяет ключевые потребности пользователей, включая минимизацию ручного труда.

Качество продукта подтверждено результатами тестирования: показатель в 98% покрытия кода на Python-датасете свидетельствует о высокой точности работы алгоритма на основе GPT-4o. Тем не менее, в ходе испытаний выявлены отдельные случаи, когда плагин некорректно обрабатывал сложные методы с неочевидной логикой или нестандартными зависимостями. Эти ошибки не носят критического характера и не нарушают общую работоспособность системы, но могут потребовать ручной доработки тестов в особых случаях.

Выводы подчеркивают, что проект достиг значительных успехов, особенно в обеспечении высокого покрытия тестов и удобства использования. Однако потенциальные проблемы с обработкой сложных кейсов и производительностью указывают на необходимость дальнейшей работы. Предложенные улучшения заложат основу для масштабирования продукта и его успешного применения в реальных условиях разработки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. JUnit 5 User Guide [Электронный ресурс]: офиц. документация / JUnit Team. – URL: https://junit.org/junit5/docs/current/user-guide/ (дата обращения: 20.05.2021).

2. Pytest Documentation [Электронный ресурс]: офиц. руководство / Pytest Development Team. – URL: https://docs.pytest.org/en/stable/ (дата обращения: 20.05.2021).

3. NUnit Documentation [Электронный ресурс]: офиц. сайт / NUnit Project. – URL: https://nunit.org/ (дата обращения: 20.05.2021).

4. GitHub Copilot Features [Электронный ресурс]: офиц. документация / GitHub. – URL: https://docs.github.com/en/copilot/about-github-copilot/github-copilot-features (дата обращения: 20.05.2021).

5. Getting Started with Diffblue Cover [Электронный ресурс]: техн. статья / JDriven. – URL: https://jdriven.com/blog/2023/07/Getting-Started-with-Diffblue-Cover (дата обращения: 20.05.2021).

6. CodeRush for Roslyn [Электронный ресурс]: расширение VS Code / DevExpress. – URL: https://marketplace.visualstudio.com/items?itemName=DevExpress.CodeRushforRoslyn (дата обращения: 20.05.2021).

7. Explyt AI [Электронный ресурс]: офиц. сайт / Explyt. – URL: https://explyt.ai/ru (дата обращения: 20.05.2021).

8. Автоматическое тестирование кода: современные инструменты [Электронный ресурс]: статья / Habr. – URL: https://habr.com/ru/companies/otus/articles/732178/ (дата обращения: 20.05.2021).

9. Yeoman Learning Guide [Электронный ресурс]: документация / Yeoman. – URL: https://yeoman.io/learning/ (дата обращения: 20.05.2021).

10. GPT-4 [Электронный ресурс]: статья / Википедия. – URL: https://ru.wikipedia.org/wiki/GPT-4 (дата обращения: 20.05.2021).