

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

Направленность Информационные системы и технологии

БАКАЛАВРСКАЯ РАБОТА

на тему

Разработка приложения для обучения
иностранным языкам

Студент _____ Габдуллин Айрат Ансарович
(подпись, дата) (ФИО полностью)

Руководитель _____ Кольчугина Е.А.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 9.06.2025 г № 12)

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
А.Ю. Козлов
«20» декабря 2024 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Габдуллин Айрат Ансарович гр. 21ВИ1 факультета ВТ

<Фамилия, имя, отчество полностью>

2. Тема работы «Разработка приложения для обучения иностранным языкам»

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

2. Руководитель работы д.т.н., доцент, доцент кафедры «МО и ПЭВМ» Кольчугина Е.А.

3. Задание на работу (назначение разработки, исходные данные и т.п.): _____

1) Разработать веб-приложение, предназначенное для обучения иностранным языкам посредством интерактивного обучения.

2) Среда функционирования: приложение должно функционировать в браузерах Google Chrome, Яндекс Браузер, Mozilla Firefox

3) Среда и средства разработки:

Языки средства разработки: Javascript, React, Vite, Node.js, UML

Технологии разработки: ООП, архитектура клиент-сервер, REST API

Среда разработки: Visual Studio Code

4) Основные функции разрабатываемой системы:

а) Управление учётными данными пользователей;

б) Обеспечение интерактивности в изучении языков посредством игры;

в) Настройка обучения по изучаемым языкам и уровню пользователей;

г) Просмотр статистики по прогрессу обучения;

д) Языковая практика в реальном времени с элементами ИИ.

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2025 – 14.04.2025	
2	Проектирование программных средств	20%	15.04.2025 – 25.04.2025	
3	Реализация программных средств	30%	26.04.2025 – 12.05.2025	
4	Тестирование программных средств	20%	13.05.2025 – 22.05.2025	
5	Оформление пояснительной записки	20%	23.05.2025 – 03.06.2025	

Дата выдачи задания «20» декабря 2024г.

Руководитель бакалаврской работы _____ Е.А. Кольчугина
(подпись, дата)

Задание к исполнению принял студент _____ А.А. Габдуллин
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 118 листов, 25 рисунков, 7 таблиц, 13 использованных источников и 2 приложения.

ПРИЛОЖЕНИЕ, ИЗУЧЕНИЕ ИНОСТРАННЫХ ЯЗЫКОВ, UML, REACT, NODE.JS, EXPRESS, VITE, IDEF1X, MS SQL, PLAYWRIGHT

Предмет работы: разработка Web-платформы для изучения иностранных языков.

Цель данной работы: разработка приложения для обучения иностранным языкам.

Средства разработки: React, Node.js, MS SQL.

Результаты работы: спроектировано и разработано приложения для обучения иностранным языкам.

Область применения: изучение иностранных языков посредством текстового и интерактивного обучения.

					ПГУ 09.03.02 – 8ВР201.14 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	«Разработка приложения для обучения иностранным языкам» Пояснительная записка	Лит.	Лист	Листов
Разраб.	Габдуллин А.А.						4	117
Провер.	Кольчугина Е.А.							
Н. Контр.	Такташкин Д.В.					группа 21ВИ1		
Утверд.	Козлов А.Ю.							

Содержание

Введение	6
1 Анализ поставленной задачи, выбор методов и средств ее решения..	7
1.1 Анализ требований к поставленной задаче и предметной области.....	7
1.2 Анализ аналогов.....	9
1.3 Выбор методов и средств разработки	13
2 Проектирование	16
2.1 Проектирование алгоритмов	17
2.2 Проектирование структур данных	23
2.3 Проектирование интерфейса	27
3 Реализация	41
4 Отладка и тестирование	54
4.1 Выбор и реализация методов тестирования	54
4.2 Функциональное тестирование	55
4.3 Тестирование совместимости	60
4.4 Тестирование пользовательского интерфейса.....	61
5 Контроль качества	63
6 Расчет бюджета разработки	66
Заключение	72
Список использованных источников	72
Приложение А (обязательное) Листинг программы	76
Приложение Б (рекомендуемое) Результаты тестирования.....	77

Введение

Из-за процессов глобализации в современном мире знание иностранных языков становится всё актуальнее и востребованнее. Но из-за быстрых темпов жизни традиционные методы изучения языков постепенно вымещаются цифровыми способами обучения.

Одним из наиболее популярных и эффективных методов цифрового обучения являются приложения, которые позволяют пользователям изучать языки в интерактивной форме. Такие приложения предлагают обширный функционал: от обучения лексике и грамматике до живого общения с носителями языка (например, в чате или по видеосвязи), а с развитием искусственного интеллекта пользователи все менее зависимы от взаимодействия с реальными носителями языка в процессе разговорной практики. Приложения позволяют изучать языки круглосуточно и в том количестве, который пользователь сочтет нужным. Совокупность этих факторов делает процесс обучения гибким, персонализированным и эффективным.

Целью настоящей выпускной квалификационной работы является разработка приложения для обучения иностранным языкам, предназначенного для пользователей разного уровня подготовки. Разрабатываемое приложение должно объединять интерактивные возможности, функциональность и простоту использования, тем самым обеспечивая эффективную среду для изучения иностранных языков.

В процессе выполнения работы рассматриваются следующие этапы: анализ предметной области, постановка задачи, обзор существующих решений, выбор архитектуры и стека технологий, реализация, тестирование и оценка качества разработанного продукта

Предмет работы: разработка приложения для обучения иностранным языкам.

1 Анализ поставленной задачи, выбор методов и средств ее решения

Использование информационных технологий в сфере изучения иностранных языков расширяет и изменяет традиционные методы обучения, делая их более интерактивными и доступными для обучающихся. Это весьма актуально в данной сфере, так как для успешного обучения важны регулярная практика и разнообразие форм подачи [3].

На рынке существует множество языковых приложений, предлагающих словари, грамматические задания и упражнения. Но существует проблема того, что многие из этих приложений перегружены функциональностью, сложны в освоении и не дают технической возможности практиковать общение с носителем языка в любое время. Помимо этого, в приложениях не всегда реализован игровой подход, способствующий вовлечённости.

1.1 Анализ требований к поставленной задаче и предметной области

Приложение должно обеспечивать выполнение следующих основных функций:

- управление учётными данными пользователей;
- обеспечение интерактивности посредством игры;
- настройка обучения по изучаемым языкам и уровню пользователей;
- просмотр статистики по прогрессу обучения;
- языковая практика в реальном времени с элементами ИИ.

Пользователь вводит свои учетные данные на сайте для входа в систему. Приложение проверяет наличие учетной записи в базе данных и правильность введенных данных. В случае успешной проверки пользователь получает доступ к своему личному кабинету, где может управлять данными аккаунта.

После входа в систему пользователь может выбрать курс для изучения, и попрактиковаться в нескольких режимах обучения.

В режимах обучения приложения должны быть предусмотрены настройки сложности заданий, которые зависят от уровня пользователя.

Приложение должно предусматривать возможность практиковаться в переписке на иностранном языке через чат с ИИ. Это позволяет совершенствоваться в живом общении с носителем языка без потребности в поиске реальных собеседников.

В результате реализации данных функций, приложение будет являться эффективным, минималистичным и увлекательным инструментом для изучения иностранных языков, подходящего как для новичков, так и для пользователей с продвинутыми знаниями.

Для наглядного отображения всех основных функций приложения была составлена диаграмма вариантов использования, представленная на рисунке 1.

Диаграмма вариантов использования – диаграмма, описывающая функционал, доступный каждой из групп пользователей в разрабатываемом приложении.

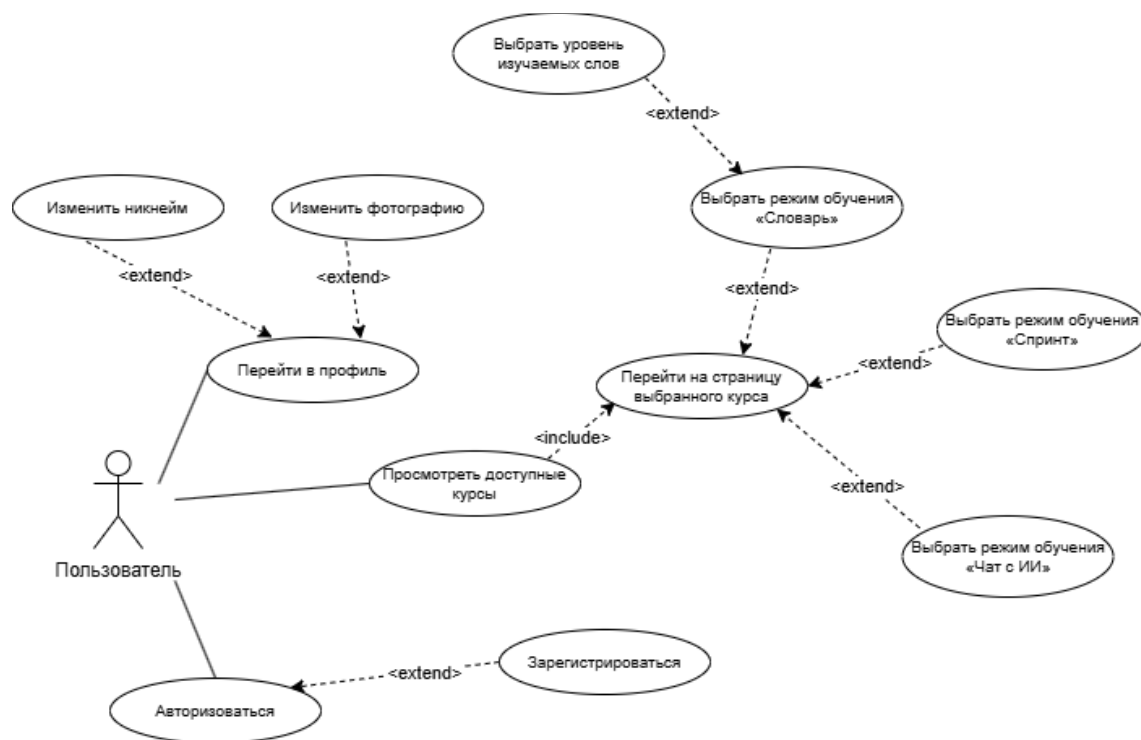


Рисунок 1 – Диаграмма вариантов использования

На диаграмме представлен пользователь, который может перейти в профиль и отредактировать его, авторизоваться, а также посмотреть доступные курсы.

Если пользователь не был ранее зарегистрирован, он может зарегистрироваться на этапе авторизации. При редактировании профиля пользователь может изменить никнейм, а также заменить фотографию профиля.

При просмотре доступных курсов пользователь может перейти на страницу выбранного курса. На странице выбранного курса пользователь может открыть любой из режимов обучения.

В режиме обучения «Словарь» пользователь помимо основного функционала режима может так же выбрать уровень слов для изучения. Также пользователь на странице выбранного курса может перейти в режимы обучения «Спринт» и «Чат с ИИ».

1.2 Анализ аналогов

Перед началом разработки приложения для обучения иностранным языкам необходимо провести анализ существующих аналогов, которые представлены на рынке [20]. Это позволит выявить сильные и слабые стороны аналогичных систем, определить выгодные направления развития, а также сформулировать конкурентные преимущества разрабатываемого приложения. Следующие ключевые аспекты, требующие внимания при проектировании системы, были выделены в процессе анализа:

- архитектура и техническая реализация приложения;
- форматы взаимодействия с пользователем;
- удобство пользовательского интерфейса;
- наличие вспомогательных функций: перевод, подсказки;
- возможность адаптации под уровень пользователя;
- требования к регистрации, безопасности и приватности данных.

Для анализа были выбраны приложения «Duolingo», «Lingualeo» и «Replika», как наиболее популярные и частично или полностью реализующие цели, сходные с целями разрабатываемого проекта.

«Duolingo» – это, пожалуй, самая известная платформа для изучения языков. Она предлагает пользователю выполнять задания по грамматике, лексике, чтению и аудированию в виде коротких уроков. Обучение построено на интерактивном игровом подходе: пользователи зарабатывают очки, повышают уровни и соревнуются с другими людьми.

Ключевые особенности Duolingo:

- структура обучения с разбивкой на темы;
- озвучка слов и предложений;
- система «жизней» и бонусов;
- отчёты о прогрессе;
- адаптивные задания.

Интерфейс приложения представлен на рисунке 2.

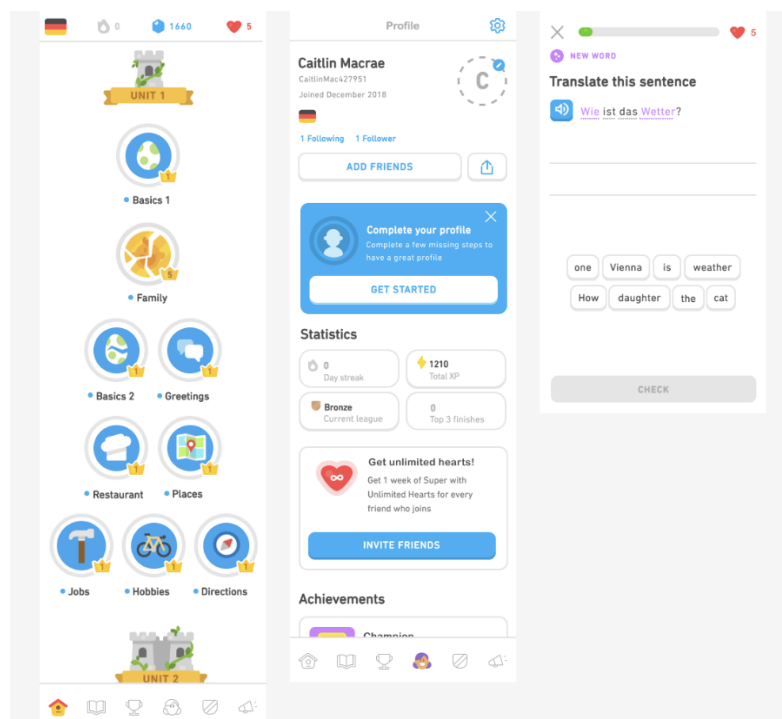


Рисунок 2 – Интерфейс приложения Duolingo

«Lingualeo» – российская образовательная платформа, также ориентированная на изучение иностранных языков. Основной акцент сделан на пополнение словарного запаса и работу с текстами. Несмотря на продвинутую работу со словарём, система не предоставляет возможности полноценной речевой практики, особенно в форме живого общения или ИИ-чата.

Особенности приложения включают:

- интерактивный словарь и возможность тренировки слов;
- чтение адаптированных текстов с переводом;
- раздел «Грамматика» с теорией и упражнениями;
- мотивационные механики (опыт, прогресс, достижения);
- часть контента доступна только в платной версии.

Интерфейс платформы представлен на рисунке 3.

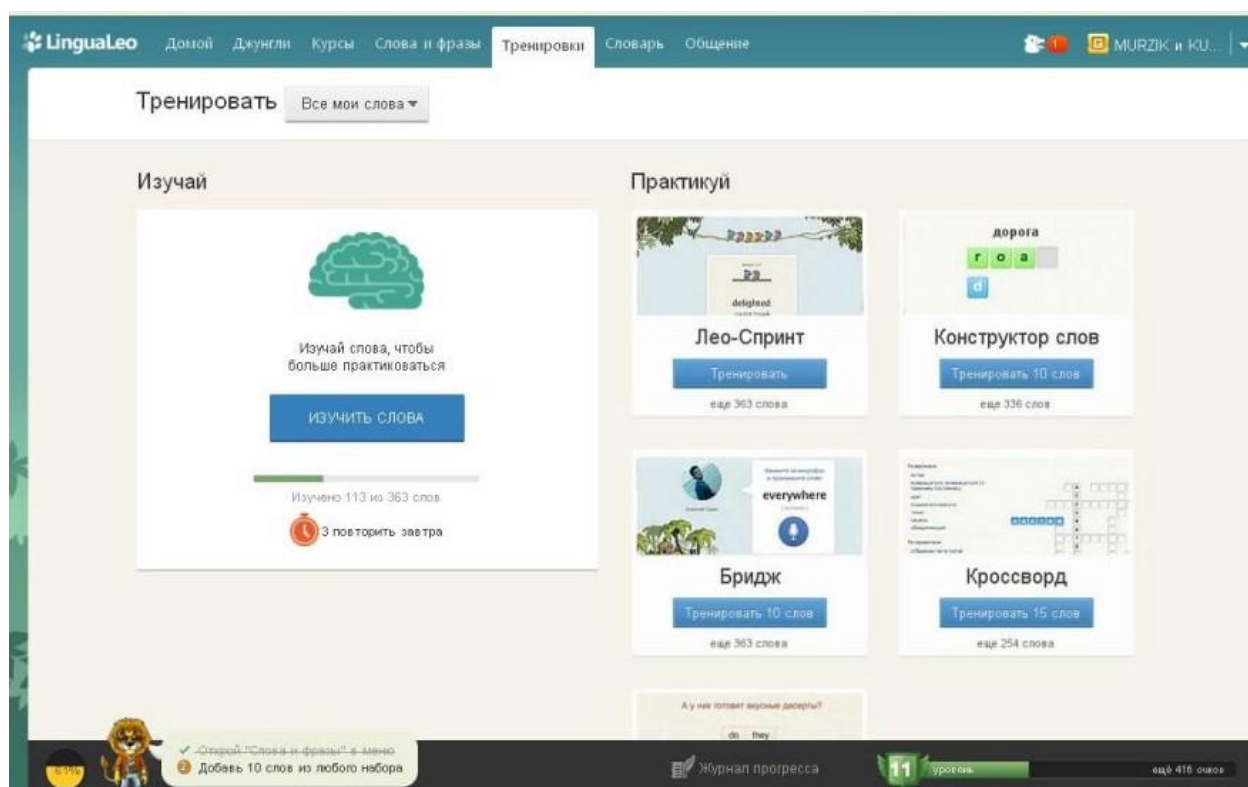


Рисунок 3 – Интерфейс приложения Lingualeo

«Replika» – это ИИ-чатбот, изначально разработанный для психологической поддержки, но активно используемый для языковой

практики. Пользователи могут вести диалоги на иностранном языке, тренируя навык общения в непринуждённой форме. Хотя Replika не является обучающей платформой в полном смысле, её функциональность чата с ИИ близка к одному из основных модулей разрабатываемого приложения.

Особенностями приложения являются:

- реалистичный диалоговый ИИ;
- память об интересах и предпочтениях пользователя;
- мотивационные и ролевые элементы общения;
- визуальное оформление в стиле мессенджера.

Интерфейс приложения Replika представлен на рисунке 4.

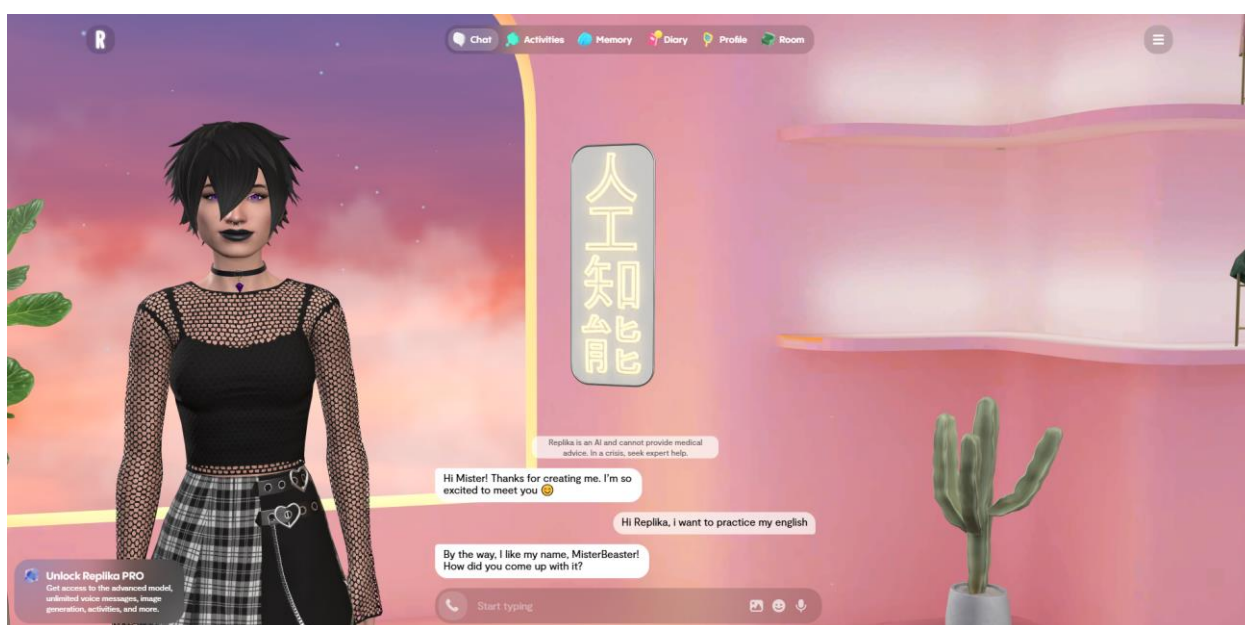


Рисунок 4 – Интерфейс приложения Replika

В ходе анализа для каждого аналога были рассмотрены общие для рассматриваемых приложений функциональные характеристики. Также было произведено сравнение аналогов с разрабатываемым приложением. Заключительным шагом проведения сравнительного анализа было составление сравнительная таблица, которая содержит результаты сравнений. Она приведена в таблице 1.

Таблица 1 – Сравнительный анализ систем-аналогов

Функциональные возможности	Duolingo	Lingualeo	Replika	Разрабатываемая система
1	2	3	4	5
Словарь	Да	Да	Нет	Да
Игровые элементы	Да	Частично	Нет	Да
Чат с ИИ	Нет	Нет	Да	Да
Адаптивные задания	Да	Частично	Нет	Да
Регистрация пользователей	Мобильное приложение	Мобильное приложение	Веб-приложение	Веб-приложение
Поддержка русского язычного интерфейса	Да	Да	Да	Да
Платный контент	Да	Да	Да	Нет

Проведённый анализ показал, что ни одно из рассмотренных решений не совмещает одновременно три ключевых решения: интерактивный словарь, игровые элементы и чат с ИИ. Это позволяет позиционировать разрабатываемое приложение как уникальный продукт, ориентированный на практическое применение языка, вовлечённость и диалоговую тренировку без необходимости постоянного контроля со стороны преподавателя.

Кроме того, было принято решение реализовать бесплатную модель доступа без ограничения функциональности, что делает платформу более конкурентоспособной, доступной и привлекательной для широкой аудитории.

1.3 Выбор методов и средств разработки

Для проектирования приложения, предназначенного для изучения иностранных языков, были выбраны современные средства и технологии,

которые обеспечивают полный цикл проектирования, разработки и внедрения информационной системы. Выбор осуществлялся на основе таких критериев, как функциональность, популярность, простота использования и совместимость между технологиями.

Draw.io был выбран для построения UML-диаграмм и других визуальных средств представления данных. Этот инструмент отличается доступностью и простотой использования, а также возможностью работы в браузере без установки дополнительного программного обеспечения.

Дополнительно использовалось программное средство ERwin Data Modeler для проектирования структуры базы данных. Этот инструмент позволяет разрабатывать как логические, так и физические модели баз данных.

Он обеспечивает наглядную организацию взаимосвязей между сущностями, что приводит к соблюдению нормализации данных, улучшению структуры хранения информации и повышенной эффективности работы с СУБД. Использование ERwin Data Modeler позволяет обеспечить целостность и масштабируемость модели данных, а также значительно облегчает этап разработки серверной части приложения.

В качестве основной интегрированной среды разработки была выбрана среда Visual Studio Code. Данный инструмент, разработанный Microsoft, имеет широкие возможности по работе с JavaScript и Node.js, поддерживает систему управления версиями Git, имеет встроенный терминал и огромное разнообразие расширений, что играет немалую роль в процессе организации разработки приложения.

Для создания пользовательского интерфейса была использована библиотека «React». Она обеспечивает построение интерфейса по компонентному принципу и позволяет легко реализовать интерактивные элементы, обновляющиеся в реальном времени. Как система сборки интерфейса использовалась библиотека «Vite», обладающая высокой скоростью и возможностью горячей замены модулей, что позволяет видеть изменения в компонентах сразу без предварительной пересборки проекта. Это

существенно ускоряет этапы разработки и отладки ПИ.

Серверная часть приложения была реализована с использованием «Node.js». Это позволило применять JavaScript не только на стороне интерфейса, но и на сервере. Node.js обладает широким функционалом: позволяет проводить асинхронную обработку запросов и имеет возможность интеграции с различными внешними модулями через систему управления пакетами «npm», что делает его подходящим языком для эффективной разработки серверной части веб-приложений.

Microsoft SQL Server (MSSQL) была выбрана как система управления базами данных. Она обладает высокой надёжностью, производительностью и поддержкой сложных реляционных структур. С помощью этой СУБД реализовано хранение информации об аккаунтах пользователей и их прогрессе в прохождении курсов.

Выбранные технологии разработки и проектирования информационных систем позволяют успешно выполнить поставленную задачу, одновременно являясь при этом удобными в использовании.

2 Проектирование

Проектирование программных средств начинается с определения целей и задач проекта, а также с формулирования требований к создаваемой системе [19]. Основной целью является обеспечение эффективного, удобного и интерактивного процесса изучения иностранных языков посредством реализации ключевых функциональных модулей.

В рамках проектирования были сформулированы следующие основные требования:

- обеспечение необходимой функциональности и адаптивности приложения к изменяющимся условиям эксплуатации;
- достижение требуемой производительности и времени отклика на пользовательские действия;
- обеспечение устойчивой и безотказной работы;
- соблюдение требований безопасности при обработке и хранении пользовательских данных;
- простота поддержки, сопровождения и возможного масштабирования системы.

Современные методологии предполагают создание и поэтапное преобразование взаимосвязанных моделей, охватывающих весь жизненный цикл программной системы. На каждом этапе жизненного цикла формируются модели, отражающие требования, архитектуру, структуру функциональных модулей и поведение системы. Эти модели обеспечивают целостность проектных решений и упрощают реализацию приложения.

В процессе проектирования были использованы средства визуального моделирования, в том числе диаграммы UML. Разработаны диаграммы классов, компонентов, а также диаграммы вариантов использования, что позволило зафиксировать структуру приложения, специфику взаимодействия между его модулями и основные пользовательские сценарии.

Использование таких инструментов позволило повысить согласованность архитектуры, упростить реализацию и обеспечить гибкость дальнейшего развития приложения [5].

2.1 Проектирование алгоритмов

Диаграмма классов представляет собой один из ключевых элементов объектно-ориентированного проектирования и используется для формализации структуры системы. В процессе разработки приложения построение диаграммы классов позволило отразить основные сущности системы, их свойства, методы, а также взаимосвязи между ними.

Создание данной диаграммы обеспечило наглядное представление архитектуры приложения и способствовало упорядочиванию логики взаимодействия между клиентской и серверной частями. Это особенно важно в условиях индивидуальной разработки, так как позволяет систематизировать проектное решение и упростить дальнейшую реализацию и сопровождение системы.

Диаграмма классов включает в себя ключевые компоненты приложения, а также их взаимосвязи. Визуализация структуры способствует обеспечению масштабируемости системы и снижению вероятности логических ошибок при реализации.

Спроектированная диаграмма классов представлена на рисунке 5.

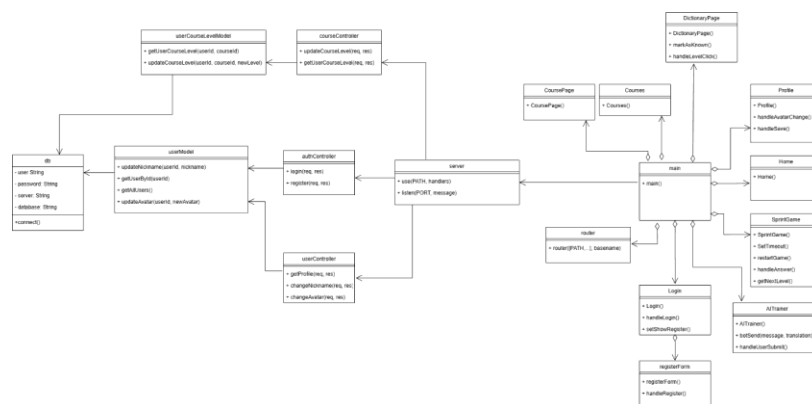


Рисунок 5 – Диаграмма классов

Описание классов и их методов представлены в таблице 2.

Таблица 2 – Описание классов и их методов

Класс	Методы	Описание
1	2	3
main	main()	Вызывает генерацию визуальной части ПИ
router	router([PATH,...], basename)	Генерирует маршруты к страницам ПИ
CoursePage	CoursePage()	Генерирует страницу курса
Login	Login()	Генерирует страницу авторизации
	handleLogin()	Обработка авторизации
	setShowRegister()	Отображение формы регистрации
registerForm	registerForm()	Генерирует форму регистрации
	handleRegister()	Обработка регистрации
DictionaryPage	DictionaryPage()	Генерирует страницу Словаря
	markAsKnown()	Обозначение слова как изученного
	handleLevelClick()	Обработка переключения уровня
Profile	Profile()	Генерирует страницу профиля
	handleAvatarChange()	Обработка смены фотографии
	handleSave()	Обработка изменения никнейма
Home	Home()	Генерирует главную страницу
SprintGame	SprintGame()	Генерирует страницу игры Спринт
	SetTimeout()	Установка таймера
	restartGame()	Перезапуск игры
	handleAnswer()	Обработка ответа пользователя
	getNextLevel()	Обработка повышения уровня
AITrainer	AITrainer()	Генерирует чат с ИИ
	botSend(message, translation)	Обработка отправки сообщения ботом. Принимает сообщение и перевод.
	handleUserSubmit()	Обработка отправки сообщения пользователя

Продолжение таблицы 2

1	2	3
server	use(PATH, handlers)	Обработка передачи запроса на контроллеры. Принимает путь запроса и обработчик запроса.
	Listen(PORT, message)	Обработчик подключения к порту. Принимает порт и сообщение при подключении.
courseController	updateCourseLevel(req,res)	Создаёт запрос с полученными данными к функции в классе userCourseLevelModel
	getUserCourseLevel(req,res)	Создаёт запрос с полученными данными к функции в классе userCourseLevelModel
authController	login(req,res)	Создаёт запрос с полученными данными к функции в классе userModel
	register(req,res)	Создаёт запрос с полученными данными к функции в классе userModel
userController	getProfile(req,res)	Создаёт запрос с полученными данными к функции в классе userModel
	changeNickname(req,res)	Создаёт запрос с полученными данными к функции в классе userModel
	chabgeAvatar(req,res)	Создаёт запрос с полученными данными к функции в классе userModel
	getUserCourseLevel(req,res)	Создаёт запрос с полученными данными к функции в классе userCourseLevelModel

Окончание таблицы 2

1	2	3
userCourseLevel Model	getUserCourseLevel(uerId,coursed)	Обращается к БД с помощью функции connect() из класса db и получает данные прогресса указанного пользователя по курсу
	updateCourseLevel(userId,courseId,newLevel)	Обращается к БД с помощью функции connect() из класса db и обновляет данные прогресса указанного пользователя по курсу
userModel	updateNickname(userId,nickname)	Обращается к БД с помощью функции connect() из класса db и обновляет данные указанного пользователя
	getUserById(userId)	Обращается к БД с помощью функции connect() из класса db и получает данные указанного пользователя
	getAllUsers()	Обращается к БД с помощью функции connect() из класса db и получает данные всех пользователей
	updateAvatar(userId,newAvatar)	Обращается к БД с помощью функции connect() из класса db и обновляет фотографию указанного пользователя
Courses	Courses()	Генерирует страницу со списком курсов
db	connect()	Подключается к указанной БД

Диаграмма деятельности - это один из основных инструментов моделирования процессов. Она используется для графического отображения

последовательности действий, которые выполняют участники системы. Диаграмма деятельности визуализирует логику выполнения операций, определяет порядок взаимодействия между пользователем и приложением, а также помогает выделить ответственных за выполнение конкретных операций.

В рамках проектирования разрабатываемого приложения была построена диаграмма деятельности, отображающая взаимодействие между пользователем и приложением в процессе работы.

Диаграмма отражает основные этапы взаимодействия с приложением: вход в систему и использования функциональных модулей, таких как словарь, игра «Спринт» и чат с ИИ.

Каждый из шагов на диаграмме содержит указание участника, ответственного за выполнение действия, ресурсов, необходимых для реализации операции, и ожидаемый результат. Это помогает в оценке операций, выявлении возможных «бутылочных горлышек», а также помогает в определении оптимальной структуры взаимодействия между компонентами приложения.

Кроме того, такая структура позволяет более эффективно распределять задачи среди команды, улучшает коммуникацию между участниками и способствует более быстрому реагированию на изменения в проекте.

Важно также учитывать возможность обратной связи на каждом этапе, что позволяет своевременно вносить коррективы и повышать качество конечного продукта.

В представленной диаграмме участниками являются пользователь и приложение. Взаимодействие между ними происходит в процессе использования приложения пользователем, а именно авторизация, переход на страницы курсов, взаимодействие с чатом, игрой «Спринт» и ИИ-чатом.

Разработанная диаграмма представлена на рисунке 6.

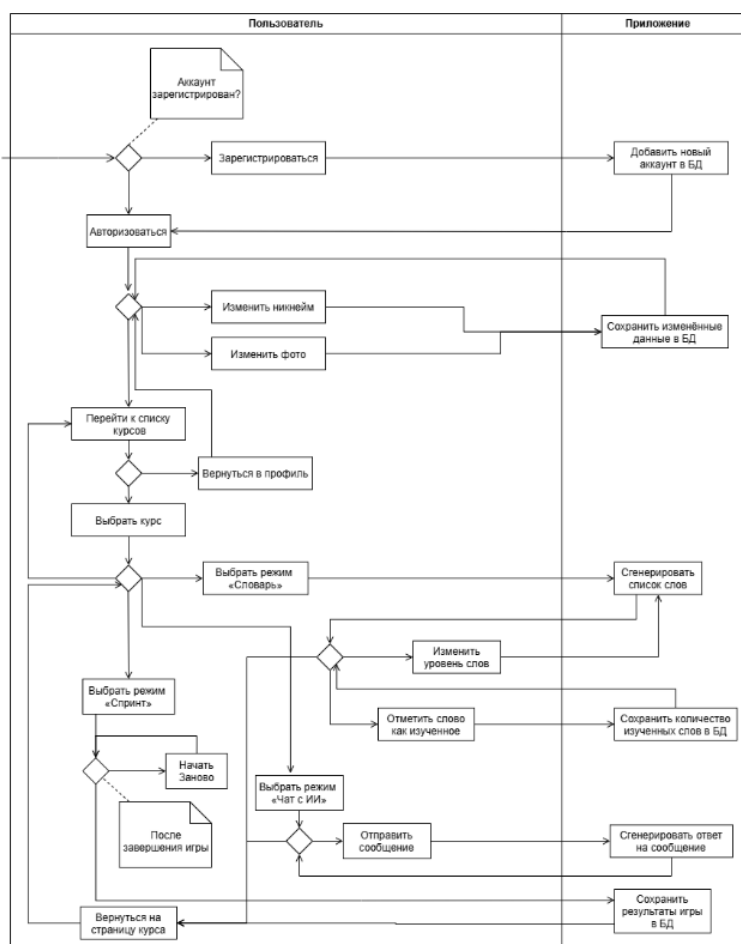


Рисунок 6 – Диаграмма деятельности

Входной точкой является вход пользователя на сайт. После нажатия кнопки «Авторизоваться» приложение проверяет наличие данного аккаунта в БД и при наличии данного аккаунта, производит редирект пользователя на страницу профиля. Также пользователь может зарегистрировать новый аккаунт и затем авторизоваться с указанными данными.

После попадания на страницу профиля, пользователя имеет выбор: изменить никнейм, изменить изображение профиля или перейти на страницу списка курсов. В случае изменения никнейма или изображения, приложение сохранит изменения в базе данных.

На странице со списком курсов пользователь может вернуться в профиль или выбрать интересующий его курс.

При выборе курса у пользователя есть выбор режима обучения: «Словарь», «Спринт» или «Чат с ИИ», при клике на режим обучения

пользователь перенаправляется на соответствующую страницу. Также пользователь может перейти в свой профиль.

При переходе в режим «Словарь» система генерирует список слов для изучения. При изменении уровня слов пользователем, система генерирует новый список слов подходящего уровня. Если слово отмечено как изученное, система обновит количество изученных слов в базе данных для конкретного пользователя и курса.

В режиме «Чат с ИИ» система генерирует и отправляет ответ при отправке сообщения пользователем.

Пользователь, который находится в режиме обучения «Спринт», может начать игру заново после её завершения.

Также из всех режимов обучения пользователь может вернуться на страницу курса.

2.2 Проектирование структур данных

Проектирование структуры данных разрабатываемого приложения осуществлялось с использованием CASE-средства ERwin Data Modeler.

Данный инструмент предназначен для построения логических и физических моделей баз данных и широко применяется при проектировании информационных систем.

В процессе моделирования были использованы два уровня представления данных:

- логические модели, отражающие структуру данных в терминах предметной области с использованием диаграмм «сущность–связь»;
- физические модели, предназначенные для создания баз данных информационных систем.

Логическая модель представляет собой абстрактное представление данных и может иметь аналоги в реальном мире [1]. Эта модель является универсальной и может использоваться в различных контекстах. На основе логической модели ERwin Data Modeler позволил автоматически

сгенерировать физическую модель и соответствующие SQL-скрипты, необходимые для создания таблиц и связей в базе данных приложения. Это обеспечило целостность структуры и ускорило процесс внедрения базы данных.

В рамках проектирования структуры данных приложения была разработана физическая модель базы данных, представленная в нотации IDEF1X. Разработка логической модели на предварительном этапе не потребовалась, поскольку структура базы данных является относительно простой и не содержит большого количества сущностей и взаимосвязей. Это позволило сразу перейти к моделированию физического уровня.

Физическая модель определяет конкретные таблицы, их поля, типы данных, первичные и внешние ключи, а также связи между таблицами. Такая детализация необходима для обеспечения корректной реализации базы данных, устойчивости к ошибкам, целостности информации и высокой производительности при взаимодействии с серверной частью приложения.

Модель была создана с использованием CASE-средства Erwin Data Modeler, что позволило обеспечить точность описания структуры данных и автоматическую генерацию SQL-скриптов для последующего развёртывания базы данных в среде Microsoft SQL Server.

Спроектированная физическая модель базы данных представлена на рисунке 7.

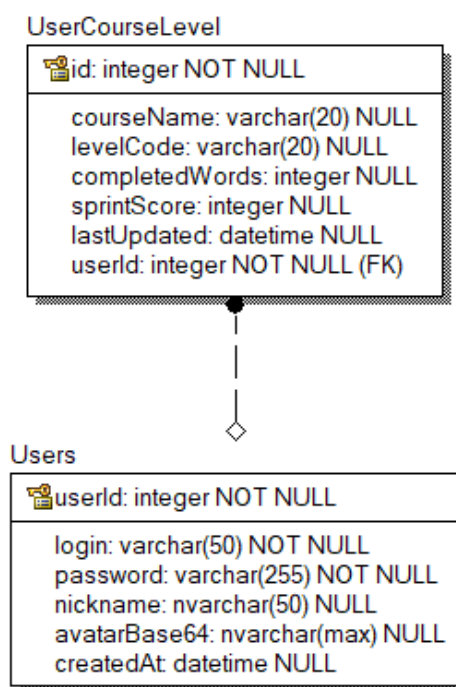


Рисунок 7 – Диаграмма IDEF1x

Представленная диаграмма состоит из сущностей, их атрибутов, и соединений между сущностями.

Описание сущностей и их атрибутов физической модели представлены в таблице 3.

Таблица 3 – Описание сущностей физической модели данных

Сущность	Атрибуты	Размерность	Роль	Описание
1	2	3	4	5
Users	userId	INTEGER	ПК	Уникальный код пользователя
	login	VARCHAR		Логин пользователя

Окончание таблицы 3

1	2	3	4	5
	password	VARCHAR		Пароль пользователя
	nickname	BLOB		Никнейм пользователя
	avatarBase64	NVARCHAR(max)		Фото пользователя
	createdAt	DATETIME		Дата создания
UserCourseLevel	id	INTEGER	PK	Уникальный код курса
	courseName	INTEGER		Язык курса
	levelCode	INTEGER		Уровень курса
	copmletedWords	INTEGER		Количество изученных слов
	sprintScore	INTEGER		Уровень в игре «Спринт»
	lastUpdated	DATETIME		Дата обновления
	userId	INTEGER	FK	Уникальный код пользователя

2.3 Проектирование интерфейса

Пользовательский интерфейс (ПИ) представляет собой совокупность средств и механизмов, обеспечивающих взаимодействие пользователя с программной системой [2]. Он охватывает все визуальные и функциональные элементы, посредством которых осуществляется управление приложением, а также восприятие информации, представленной системой. Пользовательский интерфейс может включать в себя графические компоненты (кнопки, поля ввода, выпадающие списки, окна и формы), текстовые сообщения, а также элементы навигации и обратной связи.

К основным критериям оценки качества пользовательского интерфейса относятся [4]:

- удобство использования;
- эффективность;
- визуальная привлекательность;
- последовательность.

Элементы интерфейса должны быть удобно расположены: вызов любой функции должен быть осуществим взаимодействием с одним или несколькими элементами, но не занимать больше нескольких кликов пользователя.

Интерфейс должен быть интуитивно понятным и легко осваиваемым даже для пользователя без специальной подготовки, оформление интерфейса должно быть интересным, без лишних деталей и соответствовать современным стандартам дизайна.

Также элементы интерфейса должны обладать единым стилем и поведением элементов на всех страницах приложения.

Соблюдение этих принципов способствует лучшему восприятию и предсказуемости действий для пользователя.

Для наглядного отображения взаимодействия пользователя с приложением была составлена диаграмма состояний. Она позволяет отразить

возможные состояния объекта в течение его жизненного цикла, условия переходов между ними, а также события, инициирующие данные переходы.

Использование диаграммы состояний особенно целесообразно в случаях, когда объект системы может находиться в различных логических состояниях, каждое из которых определяет особенности его поведения и доступные операции. Построение такой диаграммы способствует формализации пользовательских сценариев, обеспечению целостности логики и снижению вероятности возникновения ошибок при реализации функционала.

Основные элементы диаграммы состояний:

- состояния: отражают различные условия объекта в определённый момент времени;
- переходы: стрелки, соединяющие состояния, показывающие возможные изменения состояния;
- события: триггеры, которые вызывают переходы между состояниями.

В качестве объекта моделирования была выбрана сущность «UserCourseLevel», отражающая прогресс пользователя в рамках определённого курса и уровня.

Данный объект изменяется в зависимости от активности пользователя и учитывает такие параметры, как количество изученных слов, набранные баллы в игре и дату последнего взаимодействия с курсом.

В процессе построения диаграммы состояний были выделены начальное, конечное и промежуточные состояния, описывающие основные этапы взаимодействия пользователя с курсом от момента регистрации.

Также были определены события, запускающие переходы между состояниями, включая начало обучения, выполнение заданий, завершение уровня и повторное обращение к материалам.

Диаграмма состояний представлена на рисунке 8.

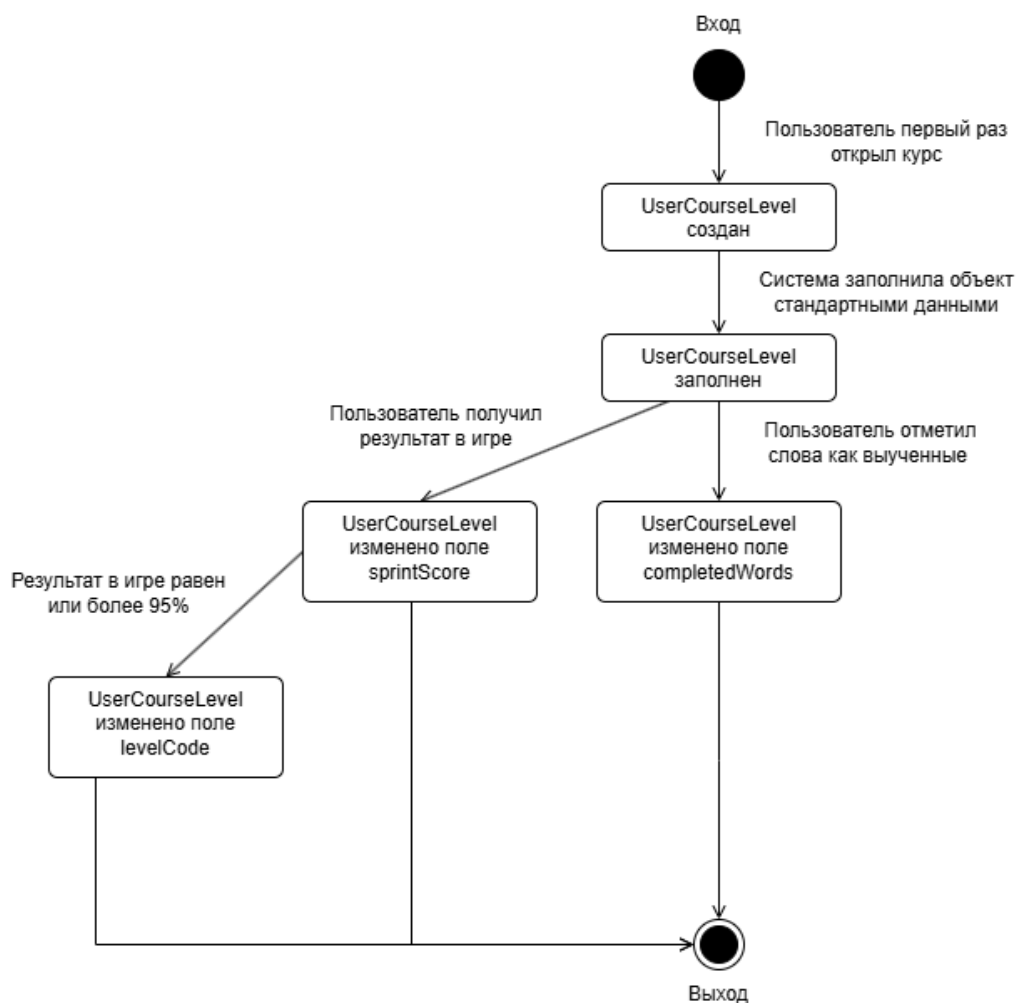


Рисунок 8 – Диаграмма состояний объекта «UserCourseLevel»

Спецификации переходов объекта из одного состояния в другое определены в таблице 4.

Таблица 4 – Спецификации переходов объекта user из состояния в состояние

Состояние 1	Событие	Состояние 2	Описание перехода
1	2	3	4
Начальное состояние	Создание объекта userCourseLevel	userCourseLevel создан	При первом заходе пользователя в курс создаётся незаполненный объект userCourseLevel
userCourseLevel создан	Система заполнила объект	userCourseLevel заполнен	Система заполняет объект стандартными данными и ID текущего пользователя

Окончание таблицы 4

1	2	3	4
userCourseLevel заполнен	Пользователь получил результат в игре	userCourseLevel изменено поле sprintScore	После завершения игры «Спринт» изменяется поле sprintScore в объекте
userCourseLevel изменено поле sprintScore	Пользователь получил результат в игре	userCourseLevel изменено поле levelCode	Если результат, который получил пользователь в игре «Спринт» больше или равен 95%, изменяется поле levelCode, которое отвечает за уровень пользователя в курсе
userCourseLevel заполнен	Пользователь отметил слова как выученные	userCourseLevel изменено поле completedWords	Если пользователь в режиме «Словарь» отметил слова как выученные, то изменяется поле completedWords, которое означает количество изученных слов пользователем
userCourseLevel изменено поле completedWords	Закрытие страницы пользователем	Конечное состояние	После закрытия пользователем страницы, объект userCourseLevel переходит в конечное состояние
userCourseLevel изменено поле levelCode	Закрытие страницы пользователем	Конечное состояние	После закрытия пользователем страницы, объект userCourseLevel переходит в конечное состояние
userCourseLevel изменено поле sprintScore	Закрытие страницы пользователем	Конечное состояние	После закрытия пользователем страницы, объект userCourseLevel переходит в конечное состояние

Интерфейс разрабатываемого приложения представлен в виде набора веб-страниц, каждая из которых выполняет определённую функцию при взаимодействии с пользователем. Элементы интерфейса обладают адаптивностью, то есть корректно отображаются на различных устройствах, экранах и браузерах.

Для данного приложения были разработаны следующие страницы:

- начальная страница;
- страница авторизации и регистрации;
- профиль пользователя;
- список курсов;
- страница курса;
- страница словаря;
- страница игры «Спринт»;
- страница чата с ИИ.

Начальная страница — это первая страница, открываемая пользователю. Она содержит краткое описание возможностей системы и ссылки на авторизацию или регистрацию. Изображение начальной страницы представлено на рисунке 9.

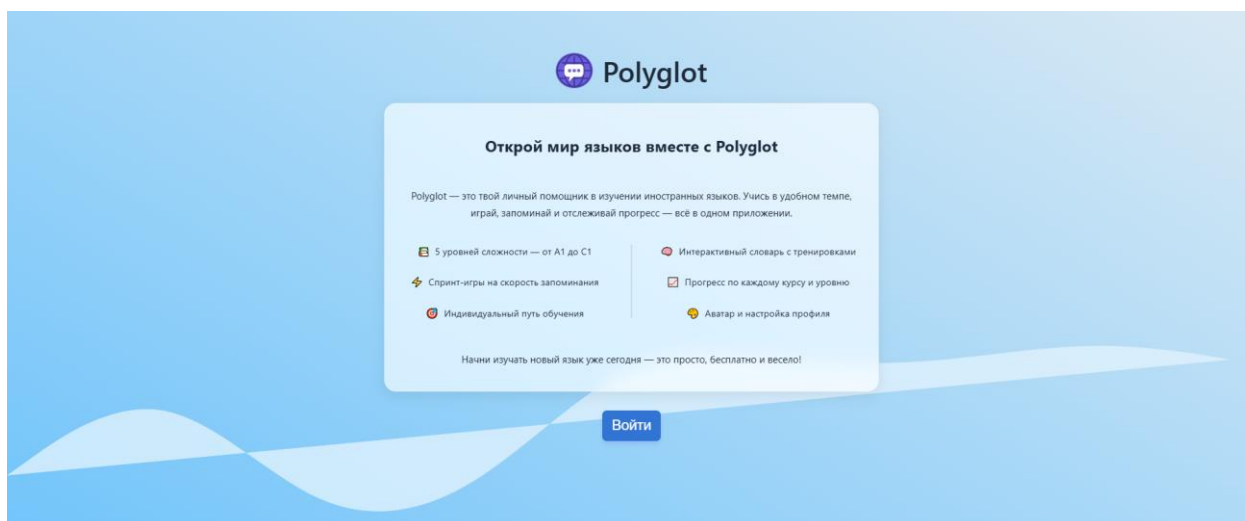


Рисунок 9 – Начальная страница

На этой странице присутствует логотип разрабатываемого приложения, а также краткое описание основных функций и преимуществ проекта. Кнопка «Войти» при нажатии отправляет пользователя на страницу авторизации и регистрации.

Страница авторизации – это страница, на которой зарегистрированный пользователь может войти в свой аккаунт, а также при нажатии на слово «Зарегистрироваться» пользователь будет перенаправлен на форму регистрации, где сможет создать новый аккаунт.

Для процесса авторизации пользователь должен ввести логин и пароль поля «Логин» и «Пароль» соответственно, а затем нажать кнопку «Войти».

Если введенные данные соответствуют данным из базы данных приложения, происходит перенаправление пользователя на страницу профиля. При введении некорректных или несуществующих данных, выводится сообщение «Неверный логин или пароль», а пользователь остаётся на той же странице.

Страница авторизации представлена на рисунке 10, форма регистрации представлена на рисунке 11.

Рисунок 10 – Страница авторизации

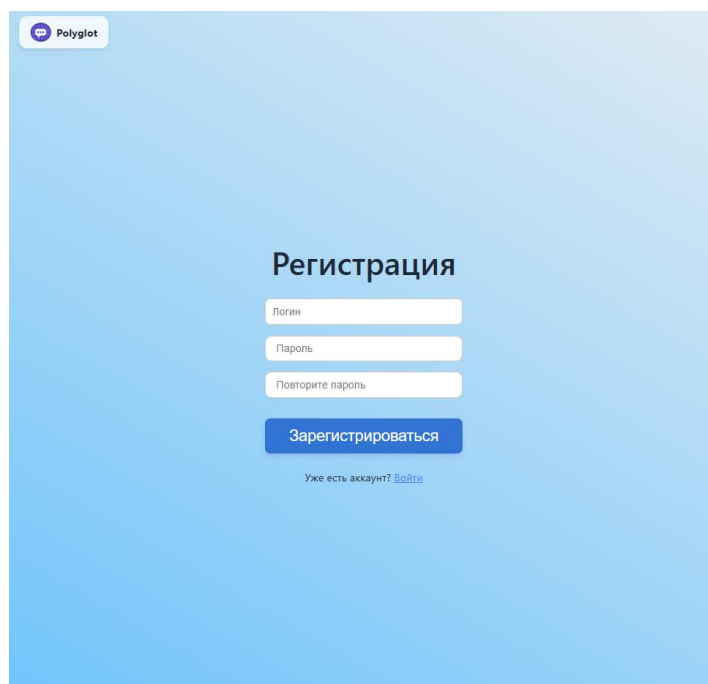
The image shows a registration form for an application named "Polyglot". The form is centered on a light blue background. At the top left, there is a logo with a speech bubble icon and the word "Polyglot". The title "Регистрация" (Registration) is centered above the input fields. There are three input fields: "Логин" (Login), "Пароль" (Password), and "Повторите пароль" (Repeat password). Below these fields is a blue button with white text that says "Зарегистрироваться" (Register). At the bottom, there is a link that says "Уже есть аккаунт? Войти" (Already have an account? Log in).

Рисунок 11 – Форма регистрации

В поля, подписанные как «Логин» и «Пароль» вводятся данные пользователя, кнопка «Зарегистрироваться» запускает процесс занесения введённых данных в базу данных приложения.

После успешной регистрации, пользователь перенаправляется на страницу авторизации, где может войти по введённым ранее данным.

Также стоит упомянуть, что на странице авторизации и последующих страницах в левом верхнем углу находится кнопка в виде логотипа проекта, при нажатии на которую пользователь переносится на главную страницу приложения.

Профиль пользователя – это страница, на которой пользователь может увидеть свои данные (никнейм, фото, прогресс, изучаемые и знакомые языки), а также изменить их, например поставить новый никнейм или загрузить на сайт новое фото, которое сохранится в БД приложения.

Помимо этого, пользователь может перейти на страницу выбора курсов. Страница личного кабинета представлена на рисунке 12.

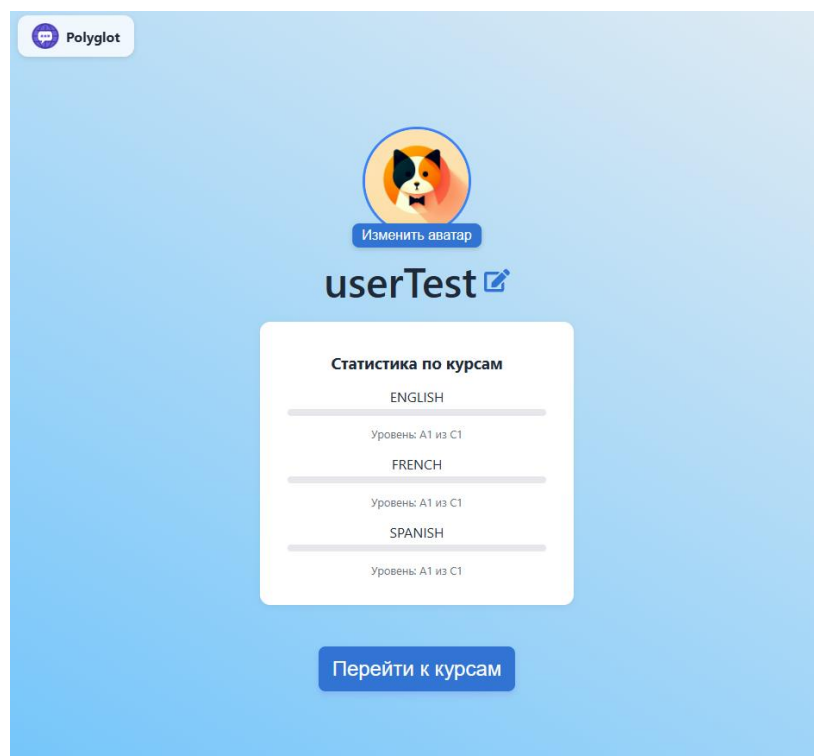


Рисунок 12 – Личный кабинет

В поле «Имя» отображается имя аккаунта таким, как его видят остальные пользователи. Его можно изменить, кликнув на специальную иконку с карандашом. После успешного изменения имени аккаунта, новое имя сохранится в базе данных приложения.

В разделе «Прогресс» пользователь может увидеть, свой прогресс по разным курсам, полоска заполняется в зависимости от процентов прохождения игры «Сплит», а также пользователь видит свой текущий уровень от «A1» до «C1», который также автоматически обновляется при переходе пользователя на новый уровень.

Нажав на кнопку «Изменить аватар» (появляется при наведении на фото), пользователь может загрузить своё фото, а при нажатии на значок справа от никнейма, он сможет изменить его. Кнопка «Перейти к курсам» перенаправляет пользователя на страницу выбора курсов.

Страница выбора курсов – это список всех доступных пользователю курсов. В карточке каждого курса пользователь также может видеть свой

уровень (от A1 до C1) и прогресс по этому уровню. Страница выбора курсов представлена на рисунке 13.

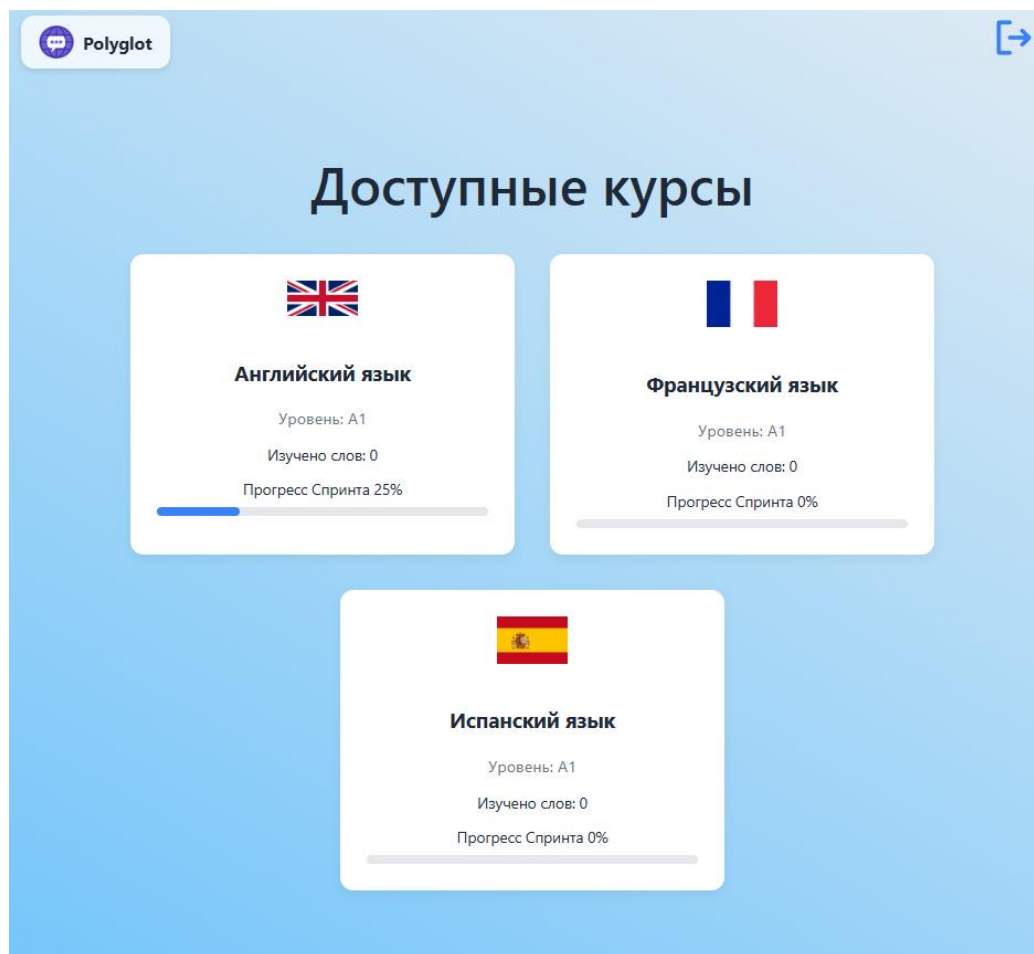


Рисунок 13 – Страница выбора курсов

В блоке «Доступные курсы» отображаются все карточки доступных пользователю курсов. На данный момент всем пользователям доступно три курса: по английскому, французскому и испанскому языкам. Помимо названий курсы также можно отличить по флагам стран, где эти языки наиболее распространены.

В правом верхнем углу находится иконка перехода на страницу профиля.

Кликнув на карточку любого из курсов, пользователь перейдёт на страницу курса, где сможет перейти к конкретным занятиям курса.

Страница курса представлена на рисунке 14.

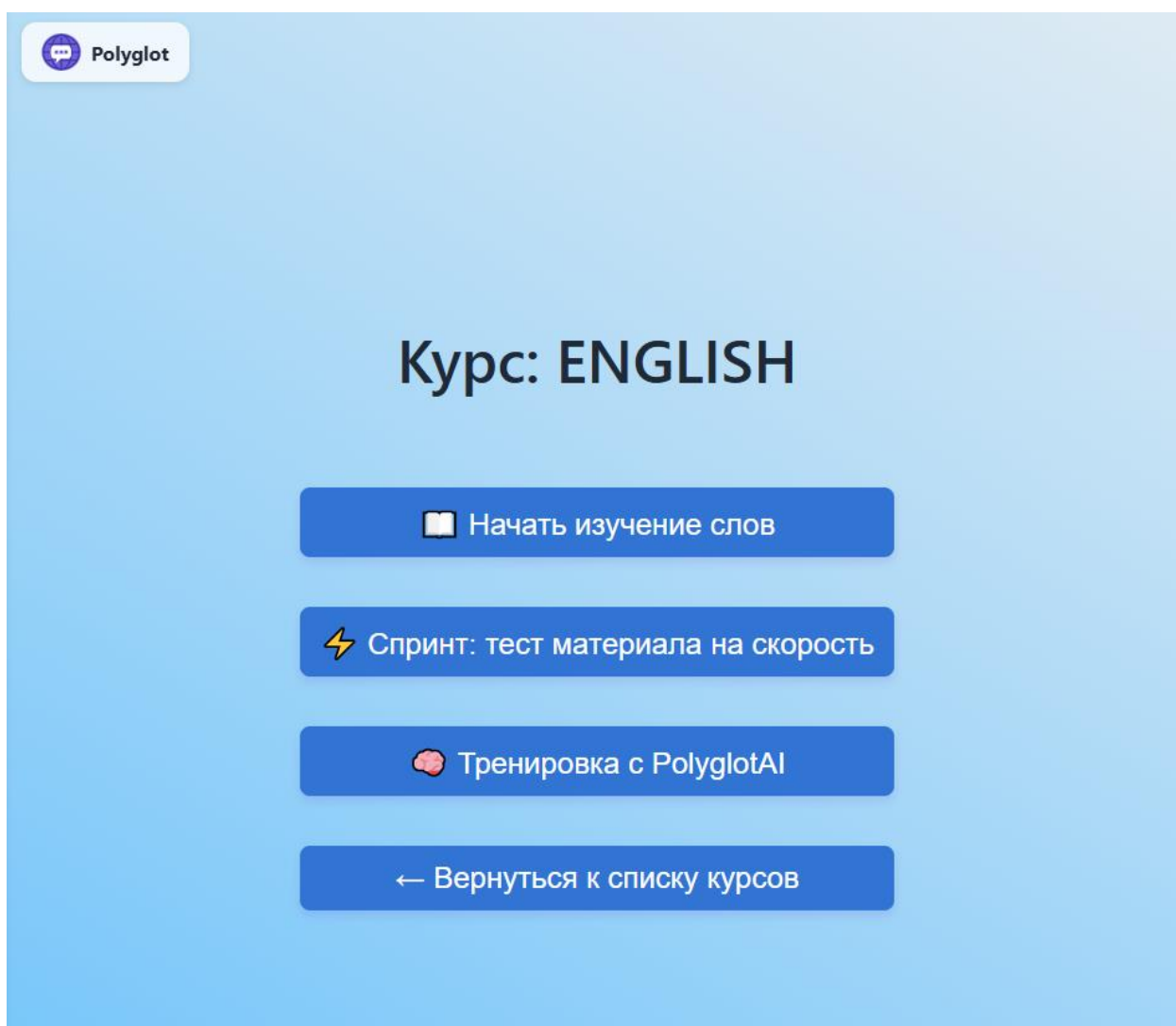


Рисунок 14 – Страница курса

В заголовке курса обозначен изучаемый язык, ниже представлены режимы обучения.

Пользователь может перейти в раздел «Словарь», кликнув по кнопке «Начать изучение слов», в раздел игры «Спринт», кликнув по кнопке «Спринт: тест материала на скорость» и в чат с ИИ, кликнув на кнопку «Тренировка с PolyglotAI». Также пользователь может вернуться на страницу курсов, нажав соответствующую кнопку. Рассмотрим подробнее страницы режимов обучения.

Страница режима «Словарь» представлена на рисунке 15.

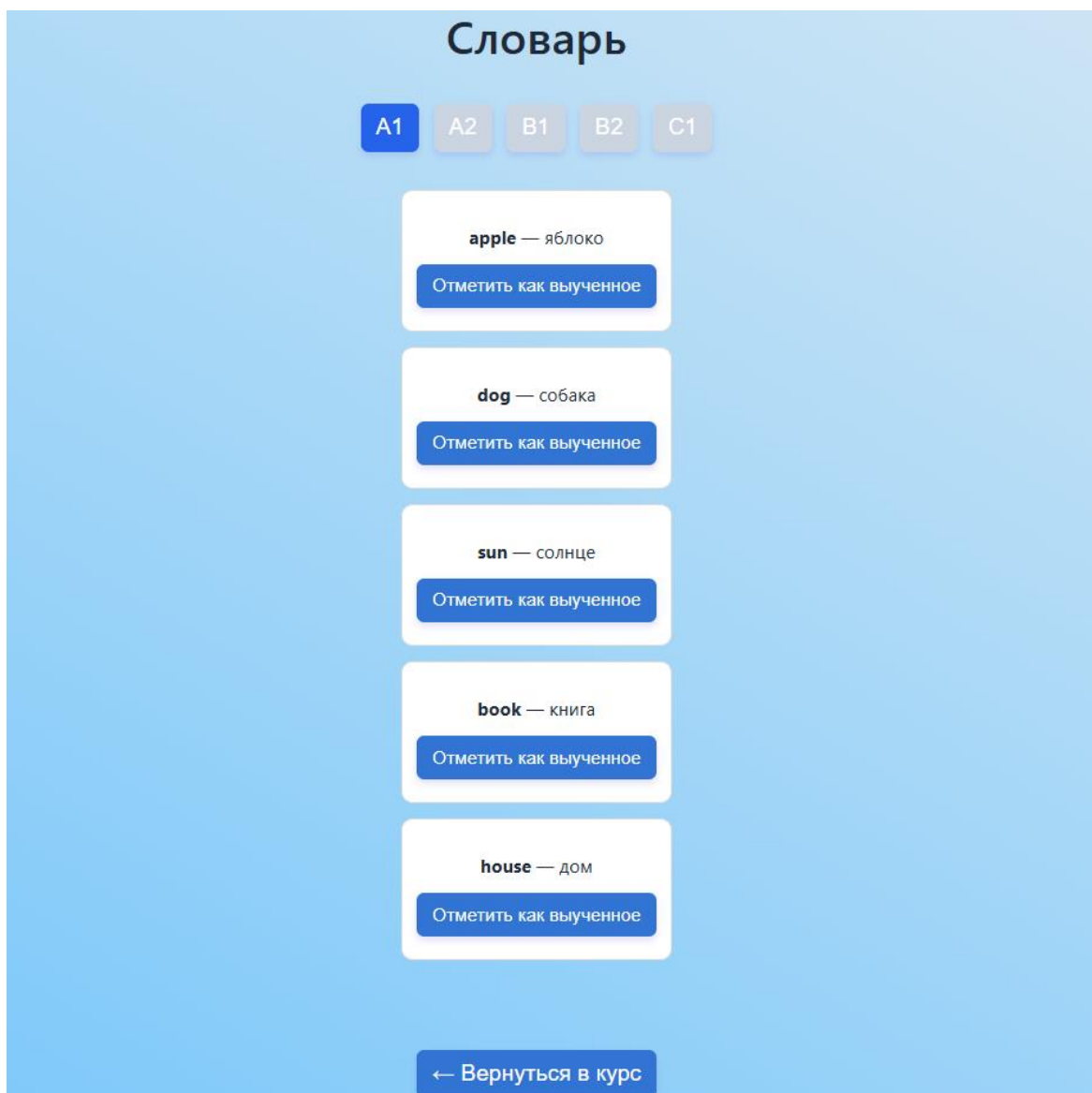


Рисунок 15 – Страница чата

В верхнем блоке отображаются уровни изучаемых слов. Изначально пользователю доступен уровень A1, и постепенно в процессе обучения будут разблокированы последующие уровни.

В блоке со словами пользователь видит слово на изучаемом языке и его перевод на русском. Пользователь должен отметить все слова как изученные с помощью соответствующей кнопки, чтобы перейти к следующей странице неизученных слов.

Также возможно вернуться на страницу курса при нажатии на кнопку «Вернуться в курс». Далее рассмотрим страницу игры «Спринт». Она изображена на рисунке 16 и рисунке 17.

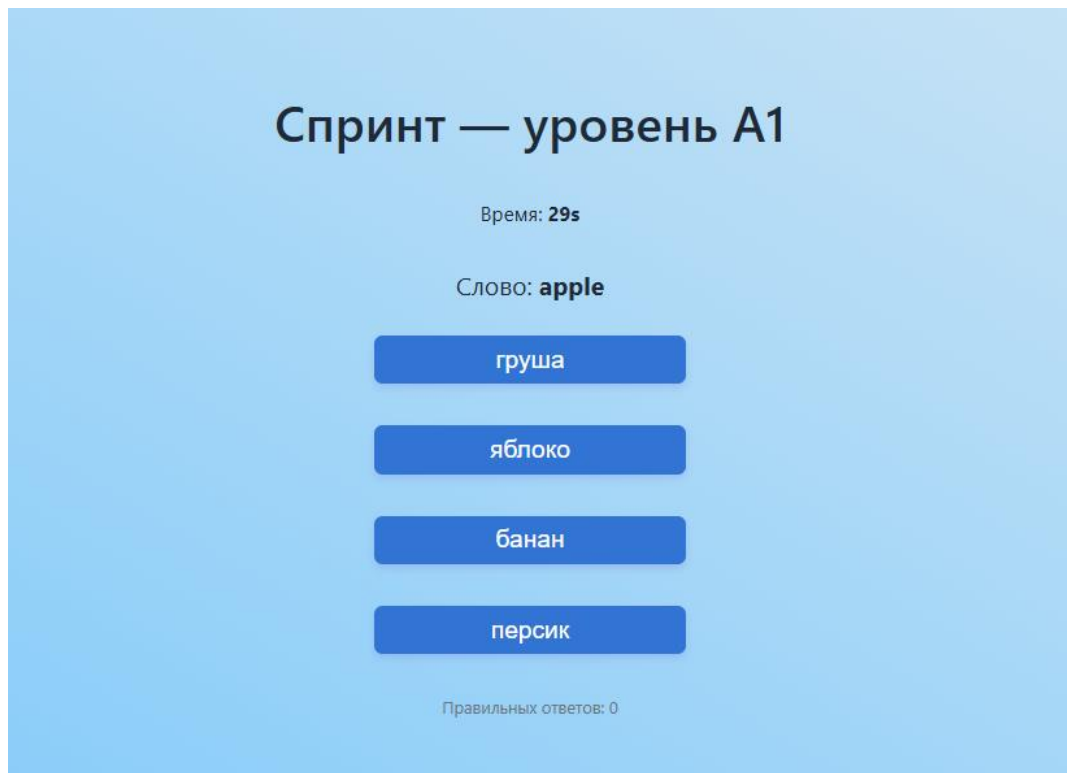


Рисунок 16 – Страница игры «Спринт»



Рисунок 17 – Результат игры «Спринт»

При переходе на страницу игры «Спринт» пользователь видит интерфейс игры (рисунок 16). В верхнем блоке отображается название игры и

текущий уровень сложности. Уровень сложности подбирается автоматически в зависимости от прогресса пользователя.

В основном блоке отображается таймер (он является обратным отсчётом от 30 до 0 секунд), слово на изучаемом языке, которое требуется перевести, и 4 варианта перевода, один из которых правильный. После окончания таймера или решения всех слов из подборки, отображается результат (рисунок 17).

В основном блоке результата игры пользователь может узнать количество правильно переведённых слов и общее количество переведённых слов. Если процент правильно переведённых слов больше 95%, то пользователю отображается поздравление с переходом на новый уровень сложности, если процент правильно переведённых слов меньше 95%, то пользователю отображается сообщение о необходимом проценте для открытия нового уровня.

Нижний блок результата отображает две кнопки: «Вернуться в курс», которая возвращает на страницу курса и «Начать заново», при нажатии на которую игра будет запущена ещё раз.

Рассмотрим последний из режимов обучения: чат с ИИ. Страница чата изображена на рисунке 18.

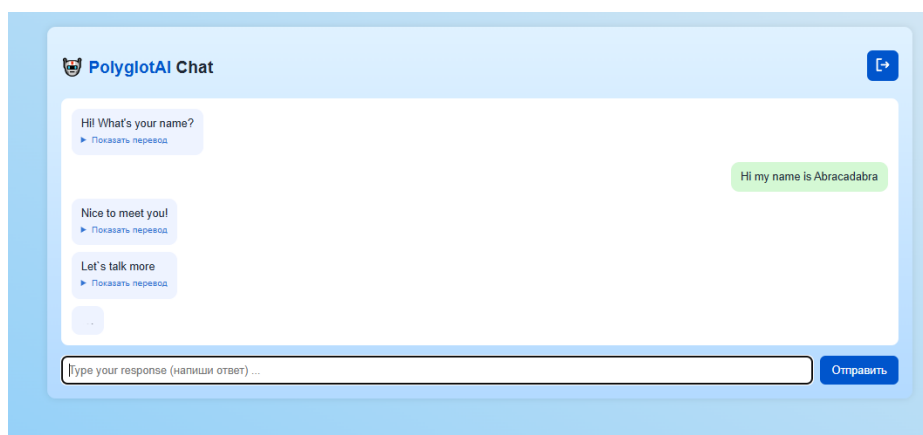


Рисунок 18 – Страница чата с ИИ

В верхней части страницы находится название режима обучения «PolyglotAI Chat» и кнопка для возврата на страницу курса в виде иконки. Основной блок является окном чата, фразы ИИ отображаются слева и имеют

синий фон, сообщения пользователя отображаются справа и имеют зелёный фон. Общая стилистика чата похожа на мессенджер. В нижнем блоке находится строка для ввода ответа, правее от неё находится кнопка «Отправить», которая отправляет сообщение пользователя.

3 Реализация

Разрабатываемое приложение реализовано на основе клиент-серверной архитектуры, которая обеспечивает разделение функций между двумя основными частями системы: клиентской и серверной. Такой подход позволяет повысить масштабируемость, гибкость и отказоустойчивость приложения, а также упростить сопровождение и последующую модернизацию.

Серверная часть приложения разработана на языке программирования JavaScript с использованием платформы Node.js и фреймворка Express, обеспечивающих обработку HTTP-запросов, маршрутизацию, работу с базой данных и реализацию бизнес-логики [11]. Сервер взаимодействует с базой данных Microsoft SQL Server через специализированные драйверы и модули.

Клиентская часть создана с использованием библиотеки React, которая позволяет строить пользовательский интерфейс по компонентному принципу, обеспечивая высокую интерактивность и производительность. Сборка и управление клиентским проектом осуществляются с использованием Vite - современного инструмента для быстрой разработки и оптимизации приложений.

Взаимодействие между клиентской и серверной частью организовано с помощью REST API – архитектурного стиля, который определяет стандартизированный обмен данными по протоколу HTTP. REST API позволяет клиенту отправлять запросы на сервер для получения или изменения данных и получать структурированные ответы в формате JSON.

Для наглядности программная реализация рассматривается поэтапно: сначала опишем общую архитектуру приложения, а затем приведём описание кода, реализованного в отдельных функциональных модулях.

Продемонстрировать компоненты, из которых состоит разрабатываемое приложение можно с помощью диаграммы компонентов, построенной с помощью UML-средств.

Диаграмма компонентов описывает физические части системы и их отношения [5]. Компоненты могут быть программными модулями, библиотеками, внешними системами и так далее. Эта диаграмма помогает понять, как компоненты взаимодействуют друг с другом в системе.

Для большей наглядности было решено отобразить архитектуру приложения на 3-х различных диаграммах компонентов:

- обобщенная диаграмма компонентов;
- диаграмма компонентов пользовательского интерфейса (ПИ);
- диаграмма компонентов серверной части.

Обобщенная диаграмма компонентов [18] помогает понять, как взаимодействуют крупные компоненты приложения: база данных (polyglot), серверная часть (backend) и пользовательский интерфейс (frontend). Данная диаграмма представлена на рисунке 19.

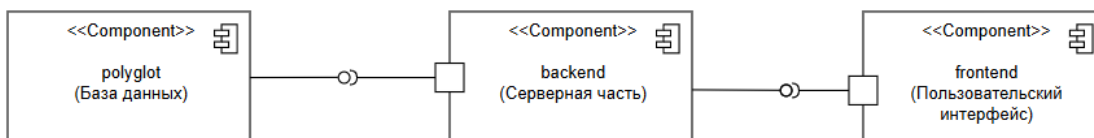


Рисунок 19 – Обобщенная диаграмма компонентов

На данной диаграмме видно, что серверная часть представляет интерфейс взаимодействия с базой данных приложения, а пользовательский интерфейс в свою очередь представляет понятный пользователю интерфейс взаимодействия с серверной частью.

В компоненте «frontend» (ПИ) находятся компоненты, которые видит пользователь при использовании приложения, а также компоненты для связи с серверной частью. Диаграмма компонентов пользовательского интерфейса представлена на рисунке 20.

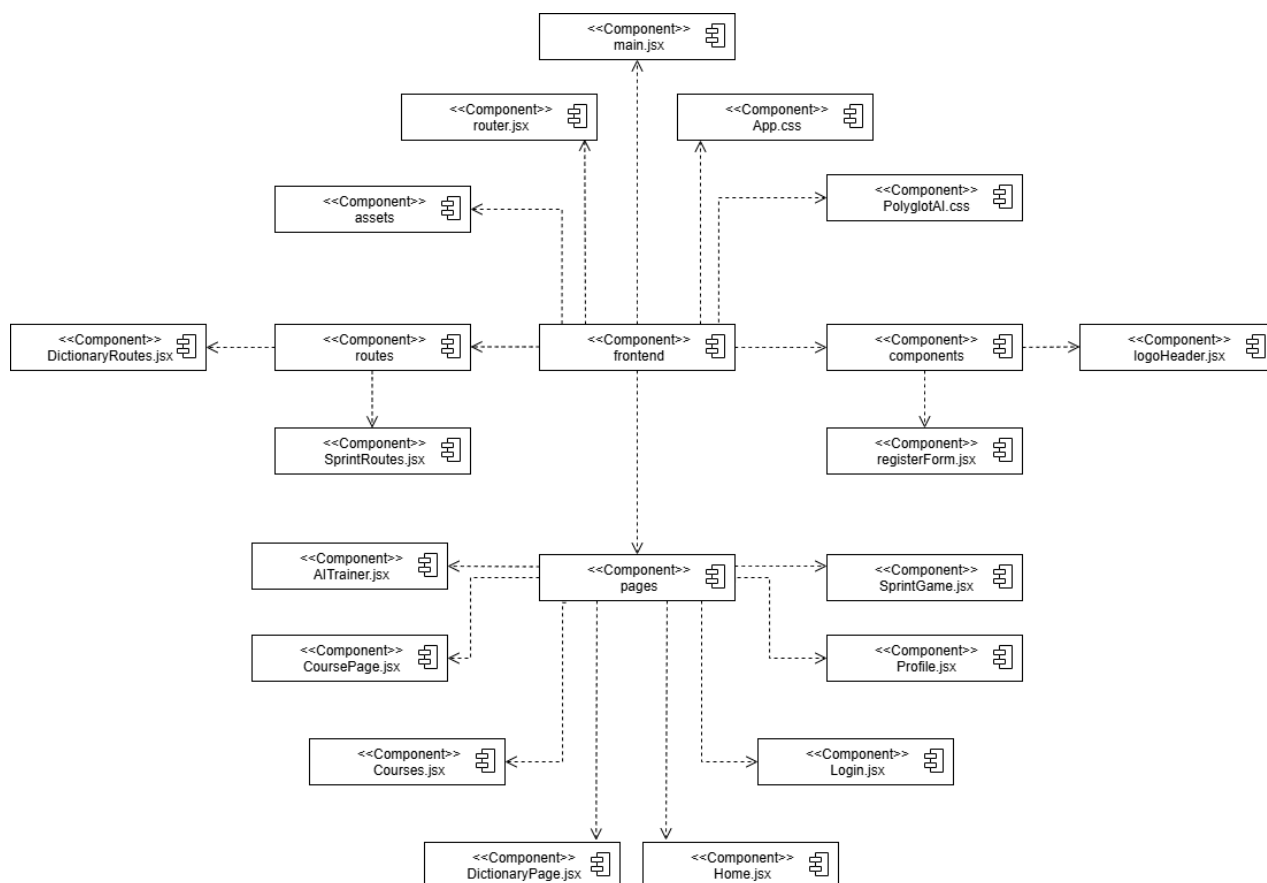


Рисунок 20 – Диаграмма компонентов ПИ

Рассмотрим подробнее компоненты данной диаграммы:

1. компонент «frontend» включает в себя: «main.jsx», «router.jsx», «assets», «components», «pages», «routes», «PolyglotAI.css» и «App.css»;
2. компонент «assets» служит хранилищем статических ресурсов (например, изображений, иконок);
3. компонент «App.css» содержит основные стили приложения, включая оформление элементов интерфейса;
4. компонент «PolyglotAI.css» отвечает за стилизацию интерфейса модуля AItrainer;
5. компонент «main.jsx» является точкой входа приложения, где инициализируется рендеринг React;
6. компонент «router.jsx» определяет маршрутизацию между страницами приложения, связывая пути с соответствующими компонентами;

7. компонент «routes» объединяет файлы маршрутов «DictionaryRoutes.jsx» и «SprintRoutes.jsx»;
8. компонент «components» содержит элементы интерфейса, которые вызываются в других компонентах;
9. компонент «logoHeader.jsx» является модулем для генерации верхней панели с логотипом приложения;
10. компонент «registerForm.jsx» является модулем для генерации формы регистрации нового пользователя;
11. компонент «pages» объединяет все основные страницы приложения;
12. компонент «Home.jsx» является модулем для генерации главной (начальной) страницы приложения;
13. компонент «Login.jsx» является модулем для генерации страницы авторизации пользователя;
14. компонент «Profile.jsx» отвечает за генерацию страницы личного кабинета пользователя;
15. компонент «Courses.jsx» генерирует страницу списка доступных языковых курсов;
16. компонент «CoursePage.jsx» является модулем для отображения содержания конкретного курса;
17. компонент «DictionaryPage.jsx» генерирует страницу для работы со словарём;
18. компонент «SprintGame.jsx» отвечает за реализацию игры Спринт;
19. компонент «AITrainer.jsx» является модулем для взаимодействия с ИИ-чатом.

В компоненте «backend» находятся модули, обрабатывающие запросы, поступающие из ПИ, а также компоненты для взаимодействия с базой данных приложения. Диаграмма компонентов серверной части представлена на рисунке 21.

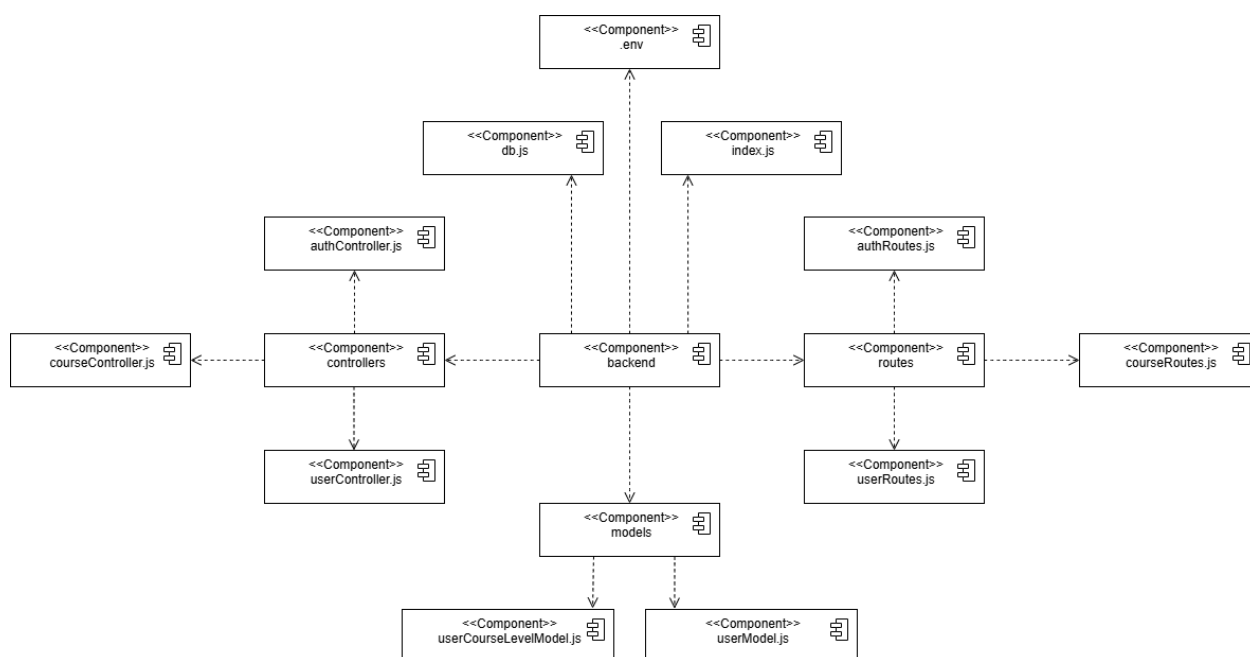


Рисунок 21 – Диаграмма компонентов серверной части

Рассмотрим подробнее компоненты данной диаграммы:

1. компонент «backend» объединяет в себе ключевые модули серверной части: «routes», «index.js», «db.js», «config», «controllers», «.env» и «models»;
2. компонент «routes» содержит описание маршрутов, по которым обрабатываются запросы к серверу. Здесь прописаны пути и указаны соответствующие контроллеры, выполняющие операции, связанные с пользователями, аутентификацией и курсами. Он включает в себя компоненты «authRoutes», «userRoutes» и «courseRoutes»;
3. компонент «controllers» реализует логику обработки полученных запросов, взаимодействуя с моделями данных. Контроллеры принимают входные данные, выполняют необходимые операции и формируют ответы для клиентской части. В него входят компоненты «authController.js», «courseController.js» и «userController.js»;
4. компонент «db.js» используется для организации подключения к базе данных;
5. компонент «index.js» инициирует работу Express-сервера;

6. компонент «.env» содержит конфигурационные данные (настройки доступа к базе данных и другие важные параметры окружения);
7. компонент «models» содержит описание структур данных и методы работы с ними;
8. компонент «authController.js» отвечает за логику, связанную с регистрацией и входом пользователей в систему;
9. компонент «userController.js» обрабатывает операции, связанные с пользовательскими данными, например получение профиля, изменение никнейма или изображения профиля;
10. компонент «courseController.js» управляет обработкой данных курсов, включая получение списка курсов и информации о них;
11. компонент «authRoutes.js» описывает маршруты, обеспечивающие процессы авторизации и регистрации пользователей;
12. компонент «userRoutes.js» определяет маршруты, связанные с действиями пользователей;
13. компонент «courseRoutes.js» содержит маршруты, необходимые для обработки запросов, касающихся курсов;
14. компонент «userModel.js» предоставляет методы для работы с данными о пользователях в базе данных;
15. компонент «userCourseLevelModel.js» реализует взаимодействие с данными о прогрессе пользователей в рамках курсов, включая информацию о пройденных заданиях, набранных баллах и времени последнего обновления.

Также для разрабатываемого приложения была составлена диаграмма развёртывания. Диаграмма развёртывания (Deployment Diagram) предназначена для отображения физического размещения программных компонентов на различных узлах, таких как серверы, клиентские устройства или иные элементы инфраструктуры [10]. Этот тип диаграммы позволяет наглядно представить, как программная система функционирует в реальной среде, а также каким образом организовано взаимодействие между её отдельными узлами.

Диаграмма развёртывания включает в себя следующие элементы:

- узлы – представляются в виде кубов и отражают физические сущности, на которых выполняются компоненты системы, подсистемы или исполняемые файлы, они могут включать как аппаратные, так и программные элементы;
- артефакты – конкретные элементы, возникающие в процессе разработки, например библиотеки, конфигурационные файлы, исполняемые модули, архивы и другие;
- коммуникационная ассоциация – отображаются линиями, соединяющими узлы, и указывают на существующие между ними пути взаимодействия.

Диаграмма развёртывания для разрабатываемого Web-приложения представлена на рисунке 22.

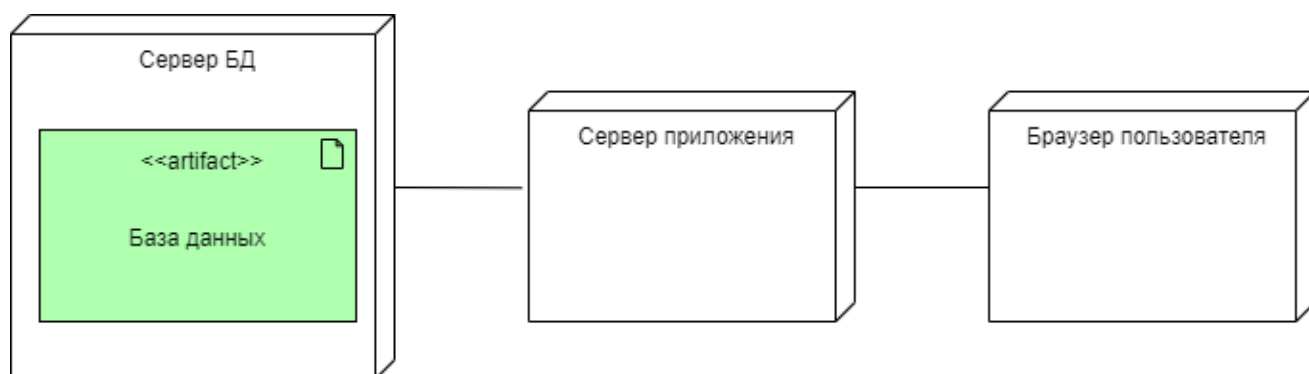


Рисунок 22 – Диаграмма развертывания

На представленной диаграмме в качестве узлов выступают следующие элементы:

1. сервер базы данных (Сервер БД). В данном приложении был использован SQL Server под управлением SQL Server Management Studio 2020;
2. сервер приложения. Был развёрнут локальный сервер под управлением Node.js на порте :5000;
3. браузер пользователя – это браузер, на котором можно открыть разрабатываемое приложение.

В качестве клиентских браузеров могут использоваться любые современные браузеры с поддержкой JavaScript; в рамках тестирования платформы были задействованы браузеры «Google Chrome», «Mozilla Firefox» и «Яндекс Браузер».

Артефактом выступает созданная база данных «polyglot». БД располагается на локальном SQL-сервере, под управление MS MSSQL. Она представлена на рисунке 23.

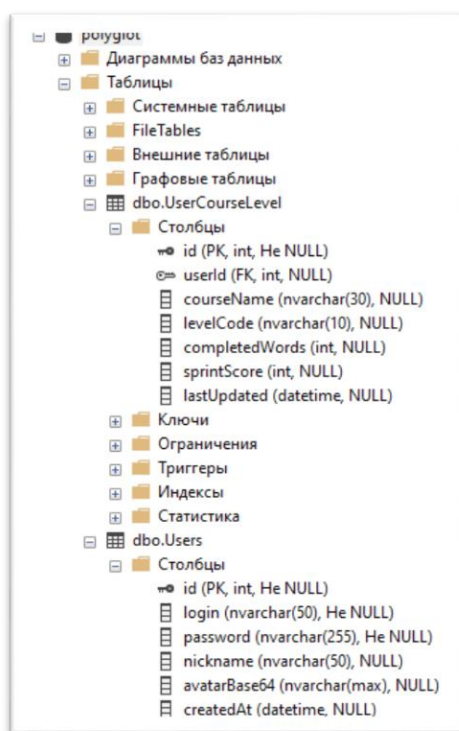


Рисунок 23 – База данных «polyglot» в среде MSSQL MS

Все модули разрабатываемого Web-приложения можно условно разделить на три типа [13]:

- модули генерации визуальных компонентов;
- модули обработки запросов;
- модули взаимодействия с базой данных.

Приведём наглядный пример модуля, отвечающего за генерацию визуальных компонентов. Для этого обратимся к модулю «logoHeader.jsx», который генерирует интерактивный логотип, используемый в большинстве страниц проекта. Код данного модуля представлен в листинге 1.

Листинг 1 – Программный код модуля «logoHeader.jsx»

```
import { useNavigate } from 'react-router-dom'
import logo from '../assets/logo.svg'
export default function LogoHeader() {
  const navigate = useNavigate()
  return (
    <div
      style={{
        position: 'fixed',
        top: '1rem',
        left: '1rem',
        cursor: 'pointer',
        display: 'flex',
        alignItems: 'center',
        gap: '0.5rem',
        backgroundColor: 'rgba(255, 255, 255, 0.8)',
        padding: '0.5rem 1rem',
        borderRadius: '12px',
        boxShadow: '0 2px 6px rgba(0, 0, 0, 0.1)',
        zIndex: 1000
      }}
      onClick={() => navigate('/')}>
      <img src={logo} alt="Polyglot Logo" style={{ width:
'32px', height: '32px' }} />
      <span style={{ fontWeight: 'bold', fontSize: '1rem',
color: '#1f2937' }}>
        Polyglot
      </span>
    </div>
  )
}
```

В первой части модуля находятся «импорты» – подключения внешних компонентов и библиотек, используемых в работе модуля. Мы можем увидеть импорт компонента «useNavigate» из библиотеки «react-router-dom», используемый для реализации навигации между страницами, а также импорт изображения логотипа «logo.svg» из папки «assets».

Далее следует объявление основной функции компонента «LogoHeader», возвращающей структуру визуальных элементов, построенную с помощью JSX. Основным контейнером выступает элемент «div», содержащий картинку с логотипом и текстовое название приложения. Атрибут «style={{}}» используется для задания встроенных стилей, аналогичных inline-стилям в HTML, что позволяет задать оформление непосредственно внутри элемента.

Особое внимание следует уделить обработчику события «onClick», который присваивается основному контейнеру. Он делает элемент интерактивным и обеспечивает переход пользователя на главную страницу приложения при клике по логотипу, используя вызов функции «navigate».

Завершающий блок «export default» делает модуль доступным для использования в других частях приложения, позволяя подключать его при формировании интерфейса.

Далее рассмотрим модуль обработки запросов на примере модуля «userController.js». Код данного модуля представлен в листинге 2.

Листинг 2 – Программный код модуля «authController.js»

```
import { getUserById, updateNickname, updateUserAvatar } from
'../models/userModel.js'

export async function getProfile(req, res) {
  const { id } = req.params
  try {
    const user = await getUserById(id)
    if (user) {
      res.json(user)
    } else {
      res.status(404).json({ message: 'User not found' })
    }
  } catch (err) {
    res.status(500).json({ error: err.message })
  }
}

export async function changeNickname(req, res) {
  const { userId } = req.params
  const { nickname } = req.body

  try {
    await updateNickname(userId, nickname)
    res.json({ success: true, message: 'Nickname updated' })
  } catch (err) {
    res.status(500).json({ error: err.message })
  }
}

export const updateAvatar = async (req, res) => {
  const { userId } = req.params
  const { avatarBase64 } = req.body

  const result = await updateUserAvatar(userId, avatarBase64)
```

```

    if (result.success) {
      res.json({ success: true, avatarBase64 })
    } else {
      res.status(500).json({ success: false, message:
result.message })
    }
  }
}

```

В первой части модуля происходит подключение функций из модуля «userModel» с помощью конструкции «import». В частности, подключаются функции «getUserById», «updateNickname» и «updateUserAvatar», которые обеспечивают взаимодействие с базой данных для получения и изменения данных пользователя.

Далее в модуле определены три асинхронные функции: «getProfile», «changeNickname» и «updateAvatar». Опишем их подробнее.

Функция «getProfile» берёт параметр «id» из тела запроса и вызывает функцию «getUserById» для поиска пользователя по идентификатору. Затем функция возвращает найденные данные в формате JSON. В случае ошибки или отсутствия пользователя возвращается сообщение об ошибке с соответствующим статусом.

Функция «changeNickname» получает параметр «userId» из параметров запроса и новое значение никнейма из тела запроса. Затем вызывает функцию «updateNickname» для обновления данных в базе, после чего формируется ответ о результате операции.

Функция «updateAvatar» извлекает из запроса параметры «userId» и новое изображение в формате Base64, а затем передаёт их в функцию «updateUserAvatar». После этого возвращает информацию о том, было ли обновление данных успешным.

Заключительный блок модуля «export» обеспечивает экспорт этих функций, что позволяет другим частям приложения, например модулям маршрутов, подключать их и использовать при обработке запросов.

Модули, отвечающие за взаимодействие с базой данных, размещаются в компоненте «models». В качестве примера рассмотрим модуль «userModel.js»,

обеспечивающий работу с данными пользователей. Код данного модуля приведён в листинге 3.

Листинг 3 – Программный код модуля «userModel.js»

```
import db from '../db.js'

export async function getUserById(id) {
  const result = await db.request()
    .input('id', id)
    .query('SELECT * FROM Users WHERE id = @id')

  return result.recordset[0]
}

export async function updateNickname(userId, nickname) {
  await db.request()
    .input('userId', userId)
    .input('nickname', nickname)
    .query('UPDATE Users SET nickname = @nickname WHERE id = @userId')
}

export const updateUserAvatar = async (userId, avatarBase64) =>
{
  try {
    console.log('avatarBase64 length:', avatarBase64.length)
    console.log('avatarBase64 preview:', avatarBase64.slice(0,
100))

    await db.request()
      .input('userId', userId)
      .input('avatarBase64', avatarBase64)
      .query('UPDATE Users SET avatarBase64 = @avatarBase64
WHERE id = @userId')

    return { success: true }
  } catch (err) {
    console.error('Ошибка в updateUserAvatar:', err)
    return { success: false, message: 'Ошибка базы данных' }
  }
}
```

В начальной части модуля подключается объект «db», который отвечает за установку соединения с базой данных и выполнение SQL-запросов. С помощью этого подключения модули получают возможность отправлять запросы и обрабатывать возвращаемые данные. Далее объявляется асинхронная функция «getUserById», которая выполняет выборку данных пользователя из таблицы «users» по переданному идентификатору. Для этого

используется цепочка методов: «db.request()» создаёт новый запрос, а «query()» отправляет SQL-запрос на сервер базы данных.

Функция «updateNickname» обновляет никнейм пользователя с заданным идентификатором. В этой функции также используется «db.request()», «input()» и «query()», где «input()» позволяет безопасно передавать параметры запроса.

Функция «updateUserAvatar» обновляет в базе данных изображение аватара, закодированное в формате Base64. При успешном выполнении возвращается положительный результат, при возникновении ошибки формируется ответ с описанием ошибки.

Заключительный блок модуля содержит экспорт этих функций, что делает их доступными для вызова в других частях серверной логики, например, в контроллерах. Программный код остальных модулей приложения приведён в приложении А.

4 Отладка и тестирование

Тестирование приложений является неотъемлемой частью процесса разработки [17]. Этот процесс обеспечивает качество, надежность и соответствие продукта изначальным требованиям. Существуют различные виды тестирования, каждый из которых проверяет специфические задачи и используется на разных этапах жизненного цикла разработки приложения. Ниже представлены основные виды тестирования и их особенности:

- модульное тестирование – предполагает проверку отдельных модулей или компонентов программы в изоляции от других частей системы;
- интеграционное тестирование – направлено на проверку взаимодействия между различными модулями или компонентами системы;
- нагрузочное тестирование - проверяет приложение под нагрузкой для оценки её производительности;
- тестирование безопасности – направлено на выявление уязвимостей в системе, которые могут повлечь нарушение работы сайта или утечку данных пользователей;
- тестирование совместимости – проверяет, как приложение работает в различных средах (например, браузерах);
- функциональное тестирование – проверяет, что конкретные функции приложения работают в соответствии с требованиями;
- тестирование пользовательского интерфейса – помогает выявить и исправить дефекты интерфейса.

4.1 Выбор и реализация методов тестирования

Основная задача разрабатываемого приложения – это предоставлять удобный и функциональный интерфейс для изучения иностранных языков. Для осуществления поставленной задачи от приложения требуется:

- исправность всех используемых функций;

- наличие кроссплатформенности (приложение должно открываться во всех популярных браузерах);
- интуитивно понятный и визуально приятный пользовательский интерфейс.

Чтобы удостовериться, что приложение соответствует всем названным выше требованиям, было проведено автоматизированное функциональное тестирование, тестирование совместимости для браузеров «Google Chrome», «Mozilla Firefox» и «Яндекс Браузер», а также было осуществлено тестирование пользовательского интерфейса.

4.2 Функциональное тестирование

Автоматизированное функционально тестирование было произведено при помощи фреймворка Playwright. Playwright – это фреймворк с открытым исходным кодом для автоматизации тестирования конечного пользователя. Он поддерживает почти все современные браузеры, включая Firefox, Chromium и Webkit с использованием единого API [7]. Playwright создан и поддерживается компанией Microsoft и обретает всё большую популярность.

При выборе инструмента тестирования также рассматривался фреймворк «WebdriverIO», но при сравнении данных фреймворков, Playwright показал своё преимущество. При сравнении WebdriverIO и Playwright ключевыми факторами стали скорость Playwright, поддержка скриншотов и записи видео, а также поддерживаемые ЯП (языки программирования).

Помимо этого, тренды установок через Npm показывает, что количество установок Playwright превзошло количество установок WebdriverIO [9]. Этот тренд указывает на то, что Playwright с большой вероятностью станет лидером среди фреймворков для e2e тестирования веб-приложений, что приведет к росту числа разработчиков и поддержки сообщества.

Для начала тестирования требуется создать проект тестирования, затем написать модули тестов, и после этого запустить проект тестирования [8].

Было создано 3 теста: «auth.сpec.js» для тестирования процесса авторизации, «courses.сpec.js» для тестирования функционала курсов и «profile.сpec.js», которые тестирует функции изменения данных на странице профиля. Процесс тестирования регистрации пользователей был произведён вручную, чтобы пронаблюдать, хэшируются ли пароли, которые записываются в базу данных приложения в ходе регистрации. Также вручную был протестирован переход на новый уровень курса, чтобы проверить, записываются ли новые данные в БД или сохраняются только локально в браузере пользователя.

Коды созданных модулей тестирования представлены в листинге 4, листинге 5 и листинге 6 соответственно.

Листинг 4 – Код модуля «auth.сpec.js»

```
import { test, expect } from '@playwright/test';

test.describe('Аутентификация', () => {
  test('Успешный логин', async ({ page }) => {
    await page.goto('http://localhost:5173');

    await page.getByTestId('home-login').click();
    await page.fill('input[placeholder="Логин"]', 'testuser');
    await page.fill('input[placeholder="Пароль"]',
'password123');
    await page.getByTestId('login-submit').click();

    await expect(page).toHaveURL(/.*\//profile/);
    await
expect(page.locator('h2.heading')).toContainText('testuser');
  });

  test('Ошибка логина', async ({ page }) => {
    await page.goto('http://localhost:5173');

    await page.getByTestId('home-login').click();
    await page.fill('input[placeholder="Логин"]', 'wronguser');
    await page.fill('input[placeholder="Пароль"]', 'wrongpass');
    await page.getByTestId('login-submit').click();

    await expect(page.getByTestId('login-
error')).toContainText(/Пользователь не найден|ошибка/i);

  });
});
```

Листинг 5 – Код модуля «courses.сpec.js»

```
import { test, expect } from '@playwright/test';
test.describe('Курсы', () => {
  test.beforeEach(async ({ page }) => {
    await page.goto('http://localhost:5173');
    await page.getByTestId('home-login').click();
    await page.fill('input[placeholder="Логин"]', 'testuser');
    await page.fill('input[placeholder="Пароль"]',
'password123');
    await page.getByTestId('login-submit').click();
    await page.click('button:has-text("Перейти к курсам")');
    await expect(page).toHaveURL(/.*\/courses/);
  });
  test('Просмотр курсов', async ({ page }) => {
    const courseCards = await page.locator('[data-
testid^="course-card-"]').count();
    expect(courseCards).toBeGreaterThan(0);
  });
  test('Переход в курс', async ({ page }) => {
    await page.getByTestId('course-card-english').click();
    await expect(page).toHaveURL(/.*\/courses\/english/);
  });
});
```

Листинг 6 – Код модуля «profile.сpec.js»

```
import { test, expect } from '@playwright/test';
test.describe('Профиль', () => {
  test.beforeEach(async ({ page }) => {
    await page.goto('http://localhost:5173');
    await page.getByTestId('home-login').click();
    await page.fill('input[placeholder="Логин"]', 'testuser');
    await page.fill('input[placeholder="Пароль"]',
'password123');
    await page.getByTestId('login-submit').click();
    await expect(page).toHaveURL(/.*\/profile/);
  });
  test('Редактирование никнейма', async ({ page }) => {
    await page.getByTestId('edit-nickname').click()
    await page.fill('input', 'NewNickname');
    await page.getByTestId('save-nickname').click();
    await
expect(page.locator('h2.heading')).toContainText('NewNickname');
  });
});
```

Описание функционального тестирования разработанного приложения представлено в таблице 5.

Результаты автоматического и ручного функционального тестирования приложения приведены в приложении Б на рисунках Б.1-Б.3.

Таблица 5 – Описание функционального тестирования

Тестируемая ситуация	Описание теста	Тестовые данные	Ожидаемый результат	Наблюдаемый результат	Результат тестирования
1	2	3	4	5	6
Авторизация пользователя с правильными данными	Пользователь заходит на страницу «/login», вводит данные, совпадающие с данными из БД	Логин = «testuser» Пароль = «password123»	Пользователь перенаправляется на страницу «/profile»	Модуль автоматического тестирования завершается с положительным результатом	Тест пройден (рисунок Б.1)
Авторизация пользователя с неправильными данными	Пользователь заходит на страницу «/login», вводит данные, не совпадающие с данными из БД	Логин = «wronguser» Пароль = «wrongpass»	Авторизация не происходит, выводится ошибка «Пользователь не найден»	Модуль автоматического тестирования завершается с положительным результатом	Тест пройден (рисунок Б.1)
Регистрация пользователя	Пользователь заходит на страницу «/login», нажимает на «Зарегистрироваться», вводит данные, которые сохраняются в БД.	Логин = «test» Пароль = «test»	В таблице «Users» появляется запись с «login=test» и зашифрованным паролем	Результат соответствует ожидаемому	Тест пройден (рисунок Б.2)

Продолжение таблицы 5

1	2	3	4	5	6
Редактирование профиля	Пользователь переходит на страницу «/profile», нажимает на иконку «Изменить», вводит новый никнейм и нажимает кнопку «Сохранить».	Никнейм = «NewNickname»	Изменения отображаются на странице профиля и сохраняются в БД.	Модуль автоматического тестирования завершается с положительным результатом	Тест пройден (рисунк Б.1)
Переход на страницу курсов	Пользователь, который находится на странице профиля, нажимает на кнопку «Перейти к курсам»	Нет	Пользователь попадает на страницу «/courses»	Модуль автоматического тестирования завершается с положительным результатом	Тест пройден (рисунк Б.1)
Переход в курс	Пользователь, который находится на странице курсов, нажимает на карточку курса	Нет	Пользователь попадает на страницу курса	Модуль автоматического тестирования завершается с положительным результатом	Тест пройден (рисунк Б.1)

Окончание таблицы 5

1	2	3	4	5	6
Повышение уровня курса	Пользователь с уровнем курса «A1» проходит игру «Спринт» на 95% или более.	userId пользователя = «6», courseName = «english», levelCode = «A1»	Уровень курса у данного пользователя становится «A2», изменения сохраняются в БД.	Результат соответствует ожидаемому	Тест пройден (рисунок Б.3)

Осуществление функционального тестирования показало, что приложение функционирует корректно и не содержит явных ошибок и сбоев в работе. Данный результат является положительным результатом проведения тестирования.

4.3 Тестирование совместимости

Основной целью данного вида тестирования является проверка того, что функциональность и отображение приложения сохраняются на одинаково высоком уровне во всех основных браузерах без существенных ошибок и отклонений [9]. Разрабатываемое приложение было протестировано в наиболее популярных браузерах, чтобы убедиться в его стабильной работе и корректном отображении интерфейса.

В рамках тестирования осуществлялось открытие и сравнение ключевых страниц приложения, включая главную страницу, страницы авторизации и регистрации, страницу личного кабинета и страницу чата. Подробные результаты проведённых испытаний представлены в приложении Б.

Результаты тестирования показали, что во всех проверенных браузерах приложение функционирует корректно, без выявленных критических ошибок,

что свидетельствует о положительном результате и подтверждает соответствие системы заявленным требованиям.

4.4 Тестирование пользовательского интерфейса

Для успешного проведения тестирования пользовательского интерфейса прежде всего следует выделить тестируемые элементы [12]:

- окно (диалоговое окно, модальное окно, плавающее окно);
- меню (главное, всплывающее, контекстное, системное);
- виджеты/элементы управления/контролы (кнопка, радио-кнопка;
- чек-бокс, значок (иконка), список, панель инструментов, полоса прокрутки, ползунок);
- вкладка;
- элементы взаимодействия: курсор мыши, текстовый курсор, поинтер («ладошка»), курсор перетаскивания и др.
- Также следует выделить основные задачи тестирования:
- проверить расположение, размер, цвет, ширину, длину элементов; возможность ввода букв или цифр;
- проверить размещение всех сообщений об ошибках, уведомлений (а также шрифт, цвет, размер, расположение и орфография текста);
- удостовериться, что использованный шрифт читабелен;
- выявить, переходит ли курсор из текстового в поинтер при наведении на активные элементы, выделяются ли выбранные элементы;
- проверить расположение и отображение всех элементов при различных разрешениях экрана, а также при изменении размера окна (проверить, появляется ли скролл);
- проверить текст на орфографические, пунктуационные ошибки;
- сделать унификацию дизайна (цвета, шрифты, текст сообщений, названия кнопок).

Результаты тестирования пользовательского интерфейса разработанного приложения представлен в таблице 6.

Таблица 6 – Результаты тестирования ПИ

Тестируемый параметр	Наблюдаемый результат	Результат теста
1	2	3
Орфография и пунктуация	Орфографических и пунктуационных ошибок не обнаружено	Тест пройден
Унификация дизайна	Все элементы единообразны на основных страницах приложения	Тест пройден
Шрифт	На всех страницах используется одинаковый шрифт, размер шрифта единообразен для схожих элементов	Тест пройден
Адаптация элементов	Элементы успешно адаптируются при изменении размера экрана	Тест пройден
Элементы взаимодействия	Элементы взаимодействия функциональны и визуально изменяются при взаимодействии с ними (рисунок Б.4 и Б.5)	Тест пройден

Проведённое тестирование ПИ показало, что все элементы работают и визуально отображаются корректно. Данный результат является положительным результатом проведения тестирования.

5 Контроль качества

Метрики кода представляют собой количественные показатели, применяемые для оценки характеристик программного обеспечения, таких как его сложность, читаемость, качество и объём. Применение метрик позволяет выявлять потенциальные проблемы в архитектуре и реализации программных модулей, а также оптимизировать процессы сопровождения и доработки.

Для разрабатываемого программного обеспечения в качестве метрик кода взяты метрики SLOC (Source Lines of Code) и метрика Холстеда [14].

Source Lines of Code (SLOC) – количество строк исходного кода, это метрика оценки программного обеспечения, которая оценивает общий объём программного кода, подсчитывает физические и логические строки кода, а также комментарии. Проведем подсчет объема модуля «userModel.js», используя данную метрику. Код модуля приведен в листинге 7.

Листинг 7 – Код модуля «userModel.js»

```
import db from '../db.js'

export async function getUserById(id) {
  const result = await db.request()
    .input('id', id)
    .query('SELECT * FROM Users WHERE id = @id')

  return result.recordset[0]
}

export async function updateNickname(userId, nickname) {
  await db.request()
    .input('userId', userId)
    .input('nickname', nickname)
    .query('UPDATE Users SET nickname = @nickname WHERE id = @userId')
}

export const updateUserAvatar = async (userId, avatarBase64) =>
{
  try {
    console.log('avatarBase64 length:', avatarBase64.length)
    console.log('avatarBase64 preview:', avatarBase64.slice(0,
100))
```

```

    await db.request()
        .input('userId', userId)
        .input('avatarBase64', avatarBase64)
        .query('UPDATE Users SET avatarBase64 = @avatarBase64
WHERE id = @userId')

    return { success: true }
} catch (err) {
    console.error('Ошибка в updateUserAvatar:', err)
    return { success: false, message: 'Ошибка базы данных' }
}
}

```

Для кода данного модуля показатели равны:

- физические строки кода (PLOC) – 33 строк;
- логические строки кода (LLOC) – 25 строк;
- комментарии – 0 строка;
- количество строк кода SLOC – 27 строк.

Теперь подсчитаем метрику Холстеда для данного модуля. Основу метрики Холстеда составляют четыре измеряемые характеристики программы:

- NUOprtr (Number of Unique Operators) – число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов), в представленном коде равен 15;
- NUOprnd (Number of Unique Operands) – число уникальных операндов программы (словарь операндов), в представленном коде равен 13;
- Noprtr (Number of Operators) – общее число операторов в программе, в представленном коде равен 36;
- Noprnd (Number of Operands) – общее число операндов в программе, в представленном коде равен 35.

Словарь модуля рассчитывается по формуле:

$$n = n1 + n2 = 15 + 13 = 28 ,$$

где n – словарь модуля;

$n1$ – количество уникальных операторов;

$n2$ – количество уникальных операндов.

Длина модуля рассчитывается по формуле:

$$N = N1 + N2 = 36 + 35 = 71 ,$$

где: N – длина модуля;

N1 – общее количество операторов;

N2 – общее количество операндов.

Объём процедуры рассчитывается по формуле:

$$V = N * \log_2(n) = 341,33$$

Сложность процедуры рассчитывается по формуле:

$$D = \left(\frac{n_1}{2}\right) * \left(\frac{N_2}{n_2}\right) = \left(\frac{15}{2}\right) * \left(\frac{35}{13}\right) = 20,19$$

Анализ полученных значений позволяет сделать вывод, что модуль обладает компактным и логически структурированным кодом, что положительно сказывается на его читаемости и сопровождении. Однако, показатели метрики Хольстеда указывают на то, что разработка и тестирование данного модуля требуют определённых усилий, связанных, прежде всего, с использованием асинхронных операций и интеграцией с базой данных. Эти особенности следует учитывать при распределении задач в процессе разработки и планировании этапов тестирования.

6 Расчет бюджета разработки

Функционально-стоимостный анализ (ФСА) – метод системного исследования функций отдельного изделия или технологического, производственного процесса, организационной структуры, ориентированный на повышение эффективности использования ресурсов путем оптимизации соотношения между потребительскими свойствами объекта исследования и затратами на его разработку, производство и эксплуатацию.

Цель ФСА состоит в достижении предприятием реализации поставленных целей при рациональном использовании ресурсов [6].

В качестве основных моментов при разработке системы рассматриваются оплата труда программиста и стоимость электроэнергии.

Также стоит учитывать, что сложность и объем разработки зависят от определенной среды и сроков выполнения работы.

Основные компоненты затрат на разработку программного обеспечения показаны на рисунке 24.

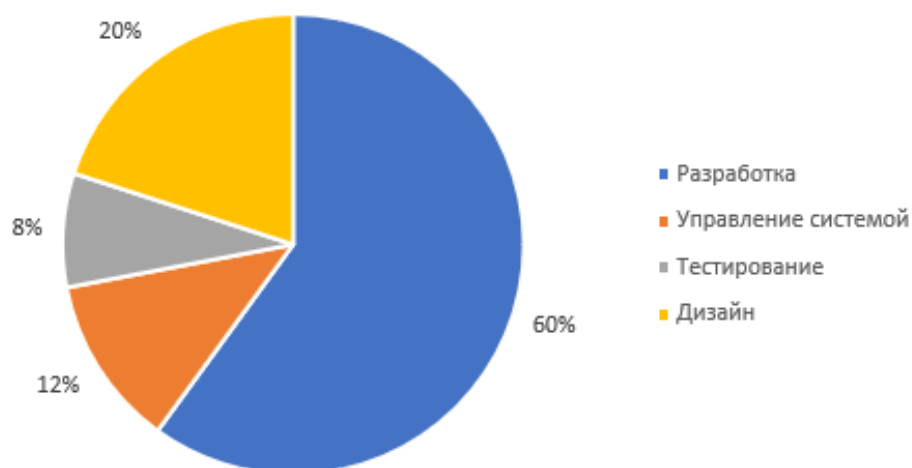


Рисунок 24 – Основные компоненты затрат на разработку

По результатам исследования сервиса «Хабр Карьера» [15], в первом полугодии 2025 г. размер оплаты труда IT-специалистов вырос до ₽190,8 тыс.

При рабочей неделе в 40 часов, в месяц выходит 160 часов работы.

Это означает, что разработчик в среднем получает $190800/160=1192,5$ руб/час. Компьютер разработчика расходует приблизительно 0,6 кВт электроэнергии. Тариф на электроэнергию составляет 3,63 рубля за кВт.

Выполнение выпускной квалификационной работы (ВКР) состоит из пяти бизнес-процессов: «Анализ предметной области», «Анализ требований и проектирование», «Реализация», «Тестирование» и «Оформление ВКР».

Был проведен анализ затрат времени и количества дней выполнения на каждый этап для одного разработчика. Для упрощённого подсчёта количества часов, нагрузка при подсчёте была распределена равномерно по 2 часа в день.

Для наглядного графического отображения плана разработки ВКР была спроектирована диаграмма Ганта [16], представленная на рисунке 25.



Рисунок 25 – Диаграмма Ганта

Учитывая все факторы, были сделаны вычисления, которые представлены в таблице 7.

Таблица 7 – Затраты на разработку

Этапы	Время на выполнение (часы)	Тариф (рублей в час)	Затраты на электроэнергию (рублей в час)	Всего
1	2	3	4	5
Анализ предметной области и требований к системе	12	1192,5	2	14334
Проектирование программных средств	20	1192,5	2	23890
Реализация	32	1192,5	2	38224
Тестирование	18	1192,5	2	21501
Оформление ВКР	20	1192,5	2	23890
Всего	102			121839

Предположительная стоимость разработки продукта составляет 121839 рублей.

Заключение

В ходе выполнения выпускной квалификационной работы был проведён анализ предметной области, рассмотрен процесс разработки приложения для изучения иностранных языков. В процессе анализа были изучены существующие решения, выявлены их сильные и слабые стороны, а также сформулированы основные функциональные требования и структура разрабатываемого приложения.

Помимо этого, было выполнено проектирование системы с использованием UML-диаграмм, таких как диаграмма вариантов использования, диаграмма классов, диаграмма деятельности, диаграмма состояний, диаграмма компонентов и диаграмма развёртывания. Эти диаграммы помогли наглядно представить архитектуру разрабатываемого приложения, его ключевые модули и процессы.

Для реализации приложения были выбраны современные технологии и средства разработки, такие как «React» для создания клиентской части, «Node.js» с библиотекой «Express» для разработки серверной части приложения, а также СУБД «MSSQL» для хранения и управления данными.

В ходе разработки реализованы такие функциональные возможности, как просмотр курсов, игра «Спринт», работа с персонализированным словарём и чат с элементами искусственного интеллекта, что делает разрабатываемое приложение эффективным инструментом для самостоятельного изучения иностранных языков.

Завершающим этапом было функциональное тестирование, тестирование интерфейса, а также тестирование совместимости. Результаты тестирования подтвердили функциональную полноту разработанного приложения, удобство его пользовательского интерфейса. Элементы ПИ также успешно прошли тестирование по всем необходимым критериям в самых популярных браузерах.

Разработанное приложение обладает потенциалом для дальнейшего развития и расширения функциональности, например, за счёт добавления новых обучающих игр, расширенной системы статистики, соревновательного элемента между пользователями и персональных рекомендаций.

Список использованных источников

1. Проектирование информационных систем [Электронный ресурс]. URL: <https://moodle.kstu.ru/course/view.php?id=4638&> (Дата обращения 29.04.2025)
2. Что такое интерфейс пользователя? [Электронный ресурс]. URL: <https://www.internet-technologies.ru/articles/newbie/polzovatelskiy-interfeys.html> (Дата обращения: 10.05.2025).
3. Как выучить иностранный язык: алгоритм [Электронный ресурс]. URL: <https://habr.com/ru/articles/530898/> (Дата обращения: 18.05.2025).
4. Проектирование интерфейса: 8 принципов, которые должен знать каждый UX-дизайнер [Электронный ресурс]. URL: <https://practicum.yandex.ru/blog/proektirovanie-interfejsov/> (Дата обращения: 12.05.2025).
5. Четыре типа диаграмм в помощь разработчику или зачем нужен UML [Электронный ресурс]. URL: <https://worksolutions.ru/blog/why-use-uml/> (Дата обращения 15.05.2025).
6. Никитина Е.Б. Функционально-стоимостный анализ – П., 2021. – 100 с.
7. End-to-end тестирование React приложения с помощью Playwright [Электронный ресурс]. URL: <https://habr.com/ru/articles/791618/> (Дата обращения 20.05.2025)
8. Начало работы с Playwright [Электронный ресурс]. URL: <https://habr.com/ru/articles/597293/> (Дата обращения: 21.05.2025).
9. Автоматическое тестирование в браузере [Электронный ресурс]. URL: <https://habr.com/ru/companies/ozontech/articles/592295/> (Дата обращения: 21.05.2025).
10. Простое руководство к диаграммам развертывания UML [Электронный ресурс]. URL: <https://creately.com/blog/ru/uncategorized-ru/учебное-пособие-по-диаграмме-развёрт/> (Дата обращения: 20.05.2025).

11. Как настроить Node.js Express сервер для React [Электронный ресурс]. URL: <https://habr.com/ru/articles/746906/> (Дата обращения: 14.05.2025).
12. Почему необходимо выполнять тестирование пользовательского интерфейса? [Электронный ресурс]. URL: <https://testmattick.com/ru/our-qa-services/user-interface-testing-services/#> (Дата обращения: 21.05.2025).
13. React.js: понятное руководство для начинающих [Электронный ресурс]. URL: <https://habr.com/ru/companies/ruvds/articles/428077/> (Дата обращения: 12.05.2025).
14. Программный код и его метрики [Электронный ресурс]. URL: <https://habr.com/ru/companies/intel/articles/106082/> (Дата обращения: 14.05.2025).
15. Зарплаты в IT в 2025: зачем мы решили оцифровать рынок [Электронный ресурс]. URL: <https://habr.com/ru/articles/918804/> (Дата обращения: 16.06.2025).
16. Руководство по созданию и использованию диаграммы Ганта [Электронный ресурс]. URL: <https://www.atlassian.com/ru/agile/project-management/gantt-chart> (Дата обращения: 12.05.2025).
17. Что такое веб-тестирование? Типы тестирования веб приложений [Электронный ресурс]. URL: <https://habr.com/ru/articles/715376/> (Дата обращения: 17.05.2025).
18. Разновидности диаграмм UML [Электронный ресурс]. URL: <https://www.lucidchart.com/blog/ru/types-of-UML-diagrams/> (Дата обращения: 14.05.2025).
19. Проектирование программного обеспечения [Электронный ресурс]. URL: <https://habr.com/ru/companies/edison/articles/267569/> (Дата обращения: 07.05.2025).
20. 15 приложений для изучения иностранных языков [Электронный ресурс]. URL: <https://kudago.com/all/list/10-prilozhenij-dlya-izucheniya/> (Дата обращения: 06.05.2025).

Приложение А
(обязательное)
Листинг программы

Листинг А.1 – Код модуля «db.js»

```
import sql from 'mssql'
import dotenv from 'dotenv'

dotenv.config()

const config = {
  user: process.env.DB_USER,
  password: process.env.DB_PASS,
  server: process.env.DB_SERVER,
  database: process.env.DB_NAME,
  options: {
    trustServerCertificate: true
  }
}

console.log('DB_SERVER:', process.env.DB_SERVER)
const pool = await sql.connect(config)
export default pool
```

Листинг А.2 – Код модуля «index.js»

```
import express from 'express'
import cors from 'cors'
import dotenv from 'dotenv'
import authRoutes from './routes/authRoutes.js'
import userRoutes from './routes/userRoutes.js'
import courseRoutes from './routes/courseRoutes.js'

dotenv.config()
const app = express()

app.use(cors())
app.use(express.json({ limit: '3mb' }))

app.use('/api/auth', authRoutes)
app.use('/api/user', userRoutes)

app.use('/api/course', courseRoutes)

const PORT = process.env.PORT || 5000
app.listen(PORT, () => console.log(`Server running on port ${PORT}`))
```

Листинг А.3 – Код модуля «authController.js»

```
import express from 'express'
import cors from 'cors'
import dotenv from 'dotenv'
import authRoutes from './routes/authRoutes.js'
import userRoutes from './routes/userRoutes.js'
```

```

import courseRoutes from './routes/courseRoutes.js'

dotenv.config()
const app = express()

app.use(cors())
app.use(express.json({ limit: '3mb' }))

app.use('/api/auth', authRoutes)
app.use('/api/user', userRoutes)

app.use('/api/course', courseRoutes)

const PORT = process.env.PORT || 5000
app.listen(PORT, () => console.log(`Server running on port ${PORT}`))

```

Листинг А.4 – Код модуля «courseController.js»

```

import { getUserCourseLevel, createUserCourseLevel,
updateUserCourseLevel } from '../models/userCourseLevelModel.js'

export const enterCourseLevel = async (req, res) => {
  const { userId } = req.params
  const { courseName, levelCode } = req.body

  try {
    const existing = await getUserCourseLevel(userId,
courseName)

    if (existing) {
      res.json({ success: true, exists: true, data: existing })
    } else {
      await createUserCourseLevel(userId, courseName, levelCode)
      const newRecord = await getUserCourseLevel(userId,
courseName)
      res.json({ success: true, exists: false, data: newRecord
})
    }
  } catch (err) {
    console.error('Ошибка при входе в курс:', err)
    res.status(500).json({ success: false, message: 'Ошибка базы
данных' })
  }
}

export async function fetchUserCourseLevel(req, res) {
  const { id, courseName } = req.params

  try {
    const data = await getUserCourseLevel(id, courseName)
    res.json(data || {})
  } catch (err) {

```

```

        console.error('Ошибка при получении уровня курса:', err)
        res.status(500).json({ success: false, message: 'Ошибка
сервера' })
    }
}

export async function updateCourseData(req, res) {
    const { userId } = req.params
    const { courseName, updates } = req.body

    try {
        await updateUserCourseLevel(userId, courseName, updates)
        res.json({ success: true })
    } catch (err) {
        console.error(err)
        res.status(500).json({ success: false, message: 'Ошибка
обновления данных' })
    }
}

```

Листинг А.5 – Код модуля «userController.js»

```

import { getUserById, updateNickname, updateUserAvatar } from
'../models/userModel.js'

export async function getProfile(req, res) {
    const { id } = req.params
    try {
        const user = await getUserById(id)
        if (user) {
            res.json(user)
        } else {
            res.status(404).json({ message: 'User not found' })
        }
    } catch (err) {
        res.status(500).json({ error: err.message })
    }
}

export async function changeNickname(req, res) {
    const { userId } = req.params
    const { nickname } = req.body

    try {
        await updateNickname(userId, nickname)
        res.json({ success: true, message: 'Nickname updated' })
    } catch (err) {
        res.status(500).json({ error: err.message })
    }
}

export const updateAvatar = async (req, res) => {
    const { userId } = req.params

```

```

const { avatarBase64 } = req.body

const result = await updateUserAvatar(userId, avatarBase64)

if (result.success) {
  res.json({ success: true, avatarBase64 })
} else {
  res.status(500).json({ success: false, message:
result.message })
}
}

```

Листинг А.6 – Код модуля «userCourseLevelModel.js»

```

import { getUserById, updateNickname, updateUserAvatar } from
'../models/userModel.js'

export async function getProfile(req, res) {
  const { id } = req.params
  try {
    const user = await getUserById(id)
    if (user) {
      res.json(user)
    } else {
      res.status(404).json({ message: 'User not found' })
    }
  } catch (err) {
    res.status(500).json({ error: err.message })
  }
}

export async function changeNickname(req, res) {
  const { userId } = req.params
  const { nickname } = req.body

  try {
    await updateNickname(userId, nickname)
    res.json({ success: true, message: 'Nickname updated' })
  } catch (err) {
    res.status(500).json({ error: err.message })
  }
}

export const updateAvatar = async (req, res) => {
  const { userId } = req.params
  const { avatarBase64 } = req.body

  const result = await updateUserAvatar(userId, avatarBase64)

  if (result.success) {
    res.json({ success: true, avatarBase64 })
  } else {

```

```

    res.status(500).json({ success: false, message:
result.message })
  }
}

```

Листинг А.7 – Код модуля «userModel.js»

```

import db from '../db.js'

export async function getUserById(id) {
  const result = await db.request()
    .input('id', id)
    .query('SELECT * FROM Users WHERE id = @id')

  return result.recordset[0]
}

export async function updateNickname(userId, nickname) {
  await db.request()
    .input('userId', userId)
    .input('nickname', nickname)
    .query('UPDATE Users SET nickname = @nickname WHERE id =
@userId')
}

export const updateUserAvatar = async (userId, avatarBase64) =>
{
  try {
    console.log('avatarBase64 length:', avatarBase64.length)
    console.log('avatarBase64 preview:', avatarBase64.slice(0,
100))

    await db.request()
      .input('userId', userId)
      .input('avatarBase64', avatarBase64)
      .query('UPDATE Users SET avatarBase64 = @avatarBase64
WHERE id = @userId')

    return { success: true }
  } catch (err) {
    console.error('Ошибка в updateUserAvatar:', err)
    return { success: false, message: 'Ошибка базы данных' }
  }
}

```

Листинг А.8 – Код модуля «authRoutes.js»

```

import express from 'express'
import { loginUser, registerUser } from
'../controllers/authController.js'

const router = express.Router()

router.post('/login', loginUser)

```

```
router.post('/register', registerUser)

export default router
```

Листинг А.9 – Код модуля «courseRoutes.js»

```
import { enterCourseLevel, fetchUserCourseLevel,
updateCourseData } from '../controllers/courseController.js'
import express from 'express'

const router = express.Router()
router.post('/:userId/enter', enterCourseLevel)
router.get('/:id/:courseName', fetchUserCourseLevel)
router.put('/:userId/update', updateCourseData)

export default router
```

Листинг А.10 – Код модуля «userRoutes.js»

```
import express from 'express'
import { getProfile, changeNickname, updateAvatar } from
'../controllers/userController.js'

const router = express.Router()

router.get('/:id', getProfile)
router.put('/:userId/nickname', changeNickname)
router.put('/:userId/avatar', updateAvatar)

export default router
```

Листинг А.11 – Код модуля «logoHeader.jsx»

```
import { useNavigate } from 'react-router-dom'
import logo from '../assets/logo.svg'

export default function LogoHeader() {
  const navigate = useNavigate()

  return (
    <div
      style={{
        position: 'fixed',
        top: '1rem',
        left: '1rem',
        cursor: 'pointer',
        display: 'flex',
        alignItems: 'center',
        gap: '0.5rem',
        backgroundColor: 'rgba(255, 255, 255, 0.8)',
        padding: '0.5rem 1rem',
        borderRadius: '12px',
        boxShadow: '0 2px 6px rgba(0, 0, 0, 0.1)',
        zIndex: 1000
      }}
    >
```

```

    }}
    onClick={() => navigate('/')}}
  >
    <img src={logo} alt="Polyglot Logo" style={{ width:
'32px', height: '32px' }} />
    <span style={{ fontWeight: 'bold', fontSize: '1rem',
color: '#1f2937' }}>
      Polyglot
    </span>
  </div>
)
}

```

Листинг A.12 – Код модуля «registerForm.jsx»

```

import { useState } from 'react'

export default function RegisterForm({ onCancel }) {
  const [login, setLogin] = useState('')
  const [password, setPassword] = useState('')
  const [repeatPassword, setRepeatPassword] = useState('')
  const [error, setError] = useState('')
  const [success, setSuccess] = useState(false)

  const handleRegister = async (e) => {
    e.preventDefault()
    setError('')
    setSuccess(false)

    if (password !== repeatPassword) {
      setError('Пароли не совпадают')
      return
    }

    try {
      const response = await
fetch('http://localhost:5000/api/auth/register', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    login,
    password,
    nickname: login,           // можно позволить менять
    avatarBase64: null        // пока без аватара
  })
    })

      const data = await response.json()

      if (data.success) {
        setSuccess(true)
        alert('Успешная регистрация! Теперь можно войти.')
        onCancel()
      }
    }
  }
}

```

```

    } else {
      setError(data.message || 'Ошибка регистрации')
    }
  } catch (err) {
    setError('Сервер не отвечает')
  }
}

return (
  <form
    onSubmit={handleRegister}
    style={{ display: 'flex', flexDirection: 'column', width:
'300px' }}
  >
    <input
      type="text"
      placeholder="Логин"
      className="input"
      value={login}
      onChange={(e) => setLogin(e.target.value)}
      required
    />
    <input
      type="password"
      placeholder="Пароль"
      className="input"
      value={password}
      onChange={(e) => setPassword(e.target.value)}
      required
    />
    <input
      type="password"
      placeholder="Повторите пароль"
      className="input"
      value={repeatPassword}
      onChange={(e) => setRepeatPassword(e.target.value)}
      required
    />

    {error && <p style={{ color: 'red', margin: '0.5rem 0'
}}>{error}</p>}
    {success && <p style={{ color: 'green' }}>Регистрация
успешна!</p>}

    <button className="button"
type="submit">Зарегистрироваться</button>

    <p style={{ marginTop: '1rem' }}>
      Уже есть аккаунт?{' '}
      <span
        onClick={onCancel}
        style={{ color: '#3b82f6', cursor: 'pointer',
textDecoration: 'underline' }}

```

```

        >
        Войти
      </span>
    </p>
  </form>
)
}

```

Листинг A.13 – Код модуля «AITrainer.jsx»

```

import { useState, useRef, useEffect } from 'react'
import { useParams, useNavigate } from 'react-router-dom'
import '../PolyglotAI.css'
import LogoHeader from '../components/logoHeader'
import { PiSignOutBold } from "react-icons/pi"
import dialogScript from '../assets/dialogScript.json'

export default function PolyglotAI() {
  const [chat, setChat] = useState([
    { from: 'bot', text: dialogScript[0].question, translation: dialogScript[0].question_ru }
  ])
  const [userInput, setUserInput] = useState('')
  const [step, setStep] = useState(0)
  const [isTyping, setIsTyping] = useState(false)
  const [awaitingContinue, setAwaitingContinue] = useState(false)
  const { lang } = useParams()
  const navigate = useNavigate()

  const chatRef = useRef(null)

  useEffect(() => {
    if (chatRef.current) {
      chatRef.current.scrollTop = chatRef.current.scrollHeight
    }
  }, [chat, isTyping])

  const botSend = (message, translation, callback) => {
    setIsTyping(true)
    setTimeout(() => {
      setIsTyping(false)
      setChat(prev => [...prev, { from: 'bot', text: message, translation: translation || null }])
      if (callback) callback()
    }, 1500)
  }

  const handleUserSubmit = () => {
    if (!userInput.trim()) return

    const userText = userInput.trim()

```

```

    setChat(prev => [...prev, { from: 'user', text: userText
    }])

    setUserInput('')

    if (awaitingContinue) {
        const cont = ['yes', 'ok', 'yeah', 'next']
        if (cont.some(word =>
userText.toLowerCase().includes(word))) {
            const nextStep = step + 1
            if (nextStep < dialogScript.length) {
                setStep(nextStep)
                botSend(dialogScript[nextStep].question,
dialogScript[nextStep].question_ru)
                setAwaitingContinue(false)
            } else {
                botSend("That's all for now! Great job 🌟",
"На этом пока что всё! Отличная работа 🌟")
                botSend("Try yourself in another study
mode.", "Проверь себя в другом режиме обучения.")
            }
        } else {
            botSend("No problem! You can come back
anytime.", "Без проблем! Ты можешь продолжить в любое время.")
            setAwaitingContinue(false)
        }
    } else {
        const current = dialogScript[step]
        const matched = current.expected.some(phrase =>
userText.toLowerCase().includes(phrase))
        const replyObj = matched
            ? current.responses[Math.floor(Math.random() *
current.responses.length)]
            : { text: "Hmm...", translation: null }

        const secondReplyObj = current.second_responses
            ?
current.second_responses[Math.floor(Math.random() *
current.second_responses.length)]
            : null

        botSend(replyObj.text, replyObj.translation, () => {
            if (secondReplyObj) {
                setTimeout(() => {
                    botSend(secondReplyObj.text,
secondReplyObj.translation, () => {
                        setAwaitingContinue(true)
                        setTimeout(() => {
                            botSend("Would you like to
continue?", "Хочешь продолжить?")
                        }, 500)
                    })
                }, 600)
            }
        })
    }
}

```

```

        } else {
            setAwaitingContinue(true)
            setTimeout(() => {
                botSend("Would you like to continue?",
"Хочешь продолжить?")
            }, 500)
        }
    })
}

return (
    <div className="ai-container">
        <LogoHeader />
        <div className="header-row">
            <h2><img alt="Polyglot AI logo" data-bbox="385 315 415 335"/> <span className="brand">PolyglotAI</span>
Chat</h2>
            <button
                className="back-button"
                onClick={() => navigate(`/courses/${lang}`)}
                title="Назад к курсу"
            >
                <PiSignOutBold size={24} />
            </button>
        </div>
        <div className="chat-box" ref={chatRef}>
            {chat.map((entry, index) => (
                <div key={index} className={`msg
${entry.from}`}>
                    {entry.text}
                    {entry.translation && entry.from ===
'bot' && (
                        <details className="translation">
                            <summary>Показать
перевод</summary>
                            <p>{entry.translation}</p>
                        </details>
                    )}
                </div>
            ))}
            {isTyping && (
                <div className="msg bot">
                    <span className="dots">
                        <span>.</span><span>.</span><span>.</span><span>.</span>
                    </span>
                </div>
            )}
        </div>
        <div className="input-area">
            <input
                type="text"

```

```

        value={userInput}
        onChange={e => setUserInput(e.target.value)}
        onKeyDown={e => e.key === 'Enter' &&
handleUserSubmit() }
        placeholder="Type your response (напиши
ответ) ..."
    />
    <button
onClick={handleUserSubmit}>Отправить</button>
    </div>
</div>

)

}

```

Листинг A.14 – Код модуля «coursePage.jsx»

```

import { useParams, useNavigate } from 'react-router-dom'
import { useState, useEffect, useRef } from 'react'
import LogoHeader from '../components/logoHeader'

export default function CoursePage() {

  const hasSentRequest = useRef(false)
  const { lang } = useParams()
  const navigate = useNavigate()
  const [userLevel, setUserLevel] = useState('A1')

  useEffect(() => {
    if (hasSentRequest.current) return
    hasSentRequest.current = true
    const user = JSON.parse(localStorage.getItem('user'))
    const userId = user?.id

    fetch(`http://localhost:5000/api/course/${userId}/enter`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        courseName: lang,
        levelCode: 'A1'
      })
    })
  })

  .then(res => res.json())
  .then(data => {
    if (data.success) {
      setUserLevel(data.data.levelCode || 'A1')
      console.log(data.exists ? 'Уже был' : 'Добавлен
впервые', data.data)
    }
  })
  .catch(() => {
    alert('Ошибка подключения к курсу')
  })
}

```

```

    })
  }, [lang])

  return (
    <div className="container">
      <LogoHeader />
      <h2 className="heading">Курс: {lang.toUpperCase()}</h2>
      <div style={{ display: 'flex', flexDirection: 'column',
gap: '1rem', marginTop: '2rem' }}>
        <button className="button" onClick={() =>
navigate(`/courses/${lang}/dictionary`)>
          📖 Начать изучение слов
        </button>
        <button className="button" onClick={() =>
navigate(`/courses/${lang}/sprint`, {
          state: {
            levelCode: userLevel
          }
        })>
          ⚡ Спринт: тест материала на скорость
        </button>
        <button className="button" onClick={() =>
navigate(`/courses/${lang}/ai`)>
          🧠 Тренировка с PolyglotAI
        </button>
        <button
          className="button"
          onClick={() => navigate('/courses')}
        >
          ← Вернуться к списку курсов
        </button>
      </div>
    </div>
  )
}

```

Листинг A.15 – Код модуля «Courses.jsx»

```

import { useNavigate } from 'react-router-dom'
import LogoHeader from '../components/logoHeader'
import { useEffect, useState } from 'react'
import { PiSignOutBold } from "react-icons/pi";

export default function Courses() {
  const navigate = useNavigate()
  const user = JSON.parse(localStorage.getItem('user'))
  const [userProgress, setUserProgress] = useState({})

  const courses = [
    {
      id: 'english',

```

```

        title: 'Английский язык',
        image: 'https://flagcdn.com/w320/gb.png',
        totalWords: 100
      },
      {
        id: 'french',
        title: 'Французский язык',
        image: 'https://flagcdn.com/w320/fr.png',
        totalWords: 120
      },
      {
        id: 'spanish',
        title: 'Испанский язык',
        image: 'https://flagcdn.com/w320/es.png',
        totalWords: 150
      }
    ]

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

      courses.forEach(course => {
        fetch(`http://localhost:5000/api/course/${user.id}/${course.id}`)
          .then(res => res.json())
          .then(data => {
            setUserProgress(prev => ({
              ...prev,
              [course.id]: data
            }))
          })
          .catch(err => console.error('Ошибка загрузки прогресса по курсу', course.id))
        })
    }, [])

    const getProgressData = (courseId, totalWords) => {
      const entry = userProgress[courseId]
      const level = entry?.levelCode || 'A1'
      const completedWords = entry?.completedWords || 0
      const completedLessons = Math.floor(completedWords / 10)
      const totalLessons = Math.ceil(totalWords / 10)
      const progressPercent = Math.min(100,
        Math.round((completedLessons / totalLessons) * 100))
      return { level, completedLessons, totalLessons, progressPercent }
    }

    return (
      <div style={{ padding: '6rem 2rem' }}>
        <LogoHeader />
        <PiSignOutBold
          onClick={() => navigate('/profile')}
        />
      </div>
    )
  )
}

```

```

        title="Вернуться в профиль"
        style={{
            fontSize: '2.5rem',
            position: 'absolute',
            top: '1rem',
            right: '1.5rem',
            color: '#3b82f6',
            cursor: 'pointer'
        }}
    />

    <h2 className="heading" style={{ textAlign:
'center', marginBottom: '2rem' }}>
        Доступные курсы
    </h2>

    <div style={{ display: 'flex', gap: '2rem',
justifyContent: 'center', flexWrap: 'wrap' }}>
        {courses.map(course => {
            const userData = userProgress[course.id]
            const level = userData?.levelCode || 'A1'
            const sprintScore = userData?.sprintScore ||
0
            const completedWords =
userData?.completedWords || 0

            return (
                <div
                    data-testid={`course-card-${
${course.id}`}
                    key={course.id}
                    style={{
                        backgroundColor: '#fff',
                        borderRadius: '12px',
                        boxShadow: '0 4px 10px
rgba(0,0,0,0.05)',
                        width: '300px',
                        padding: '1.5rem',
                        textAlign: 'center',
                        cursor: 'pointer'
                    }}
                    onClick={() =>
navigate(`/courses/${course.id}`)}
                >
                    <img src={course.image}
alt={course.title} style={{ width: '64px', marginBottom: '1rem'
}} />
                    <h3>{course.title}</h3>
                    <p style={{ fontSize: '0.9rem',
color: '#6b7280', marginBottom: '0.5rem' }}>
                        Уровень: {level}
                    </p>
                </div>
            )
        })}
    </div>

```

```

marginBottom: '0.5rem' }}>
    <p style={{ fontSize: '0.85rem',
        Изучено слов: {completedWords}
    </p>

marginBottom: '0.5rem' }}>
    <p style={{ fontSize: '0.85rem',
        Прогресс Спринта {sprintScore}%
    </p>

    <div style={{ background: '#e5e7eb',
height: '8px', borderRadius: '4px' }}>
        <div
            style={{
                background: '#3b82f6',
                width:
`${sprintScore}%`,
                height: '100%',
                borderRadius: '4px'
            }}
        ></div>
    </div>
</div>
    )
    )))
</div>
</div>
)
}

```

Листинг A.16 – Код модуля «DictionaryPage.jsx»

```

import { useState, useEffect } from 'react'
import wordsEn from '../assets/wordsForEnglishCourse.json'
import wordsFr from '../assets/wordsForFrenchCourse.json'
import wordsEs from '../assets/wordsForSpanishCourse.json'

const allLevels = ['A1', 'A2', 'B1', 'B2', 'C1']
const wordsPerPage = 5

const courseWords = {
    english: wordsEn,
    french: wordsFr,
    spanish: wordsEs
}

export default function DictionaryPage({ courseId }) {
    const user = JSON.parse(localStorage.getItem('user'))
    const userId = user?.id

```

```

const [level, setLevel] = useState('A1')
const [openLevels, setOpenLevels] = useState(['A1'])
const [page, setPage] = useState(0)
const [learnedWords, setLearnedWords] = useState([])
const [showLearned, setShowLearned] = useState(false)
const [showLockMessage, setShowLockMessage] = useState(false)
const [currentCompletedWords, setCurrentCompletedWords] =
useState(0)

const wordsData = courseWords[courseId] || []
const currentLevelWords = wordsData.filter(w => w.level ===
level)

const learningWordsList =
currentLevelWords.slice(currentCompletedWords)
const totalPages = Math.ceil(learningWordsList.length /
wordsPerPage)
const currentWords = learningWordsList.slice(page *
wordsPerPage, (page + 1) * wordsPerPage)

// ♦ Загружаем прогресс пользователя один раз при заходе
useEffect(() => {
  async function fetchProgress() {
    try {
      const res = await
fetch(`http://localhost:5000/api/course/${userId}/${courseId}`)
      const data = await res.json()
      const userLevel = data?.levelCode || 'A1'
      setCurrentCompletedWords(data?.completedWords || 0)
      setLevel(userLevel)

      const unlockedIndex = allLevels.indexOf(userLevel)
      const unlockedLevels = allLevels.slice(0, unlockedIndex
+ 1)

      setOpenLevels(unlockedLevels)
    } catch (err) {
      console.error('Ошибка при загрузке прогресса:', err)
    }
  }

  fetchProgress()
}, [userId, courseId])

const handleLevelClick = (lvl) => {
  if (!openLevels.includes(lvl)) {
    setShowLockMessage(true)
  } else {
    setLevel(lvl)
    setPage(0)
    setLearnedWords([])
    setShowLockMessage(false)
  }
}

```

```

    }

    const markAsKnown = (word) => {
      if (!learnedWords.includes(word)) {
        setLearnedWords([...learnedWords, word])
      }
    }

    const markAsUnknown = (word) => {
      setLearnedWords(learnedWords.filter(w => w !== word))
    }

    const handleNextPage = async () => {
      try {
        const response = await
fetch(`http://localhost:5000/api/course/${userId}/update`, {
  method: 'PUT',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    courseName: courseId,
    updates: {
      completedWords: currentCompletedWords + wordsPerPage
    }
  })
})

        const data = await response.json()
        if (data.success) {
          console.log('Прогресс обновлён: +', wordsPerPage, 'слов')

          const freshData = await
fetch(`http://localhost:5000/api/course/${userId}/${courseId}`)
            .then(res => res.json())
          setCurrentCompletedWords(freshData?.completedWords)
          setLevel(freshData.levelCode)

          // ♦ сбрасываем страницу на 0
          setPage(0)
        } else {
          alert('Ошибка обновления прогресса')
        }
      } catch (err) {
        console.error('Ошибка:', err)
        alert('Сервер недоступен')
      }
    }

    return (
      <div className="container">
        <h2 className="heading">Словарь</h2>

        { /* Тумблер режима */ }
      </div>
    )
  }
}

```

```

    <div style={{ display: 'flex', gap: '1rem', marginBottom:
'1rem' }}>
      <button
        className="button"
        onClick={() => setShowLearned(false)}
        style={{ backgroundColor: !showLearned ? '#3b82f6' :
'#cbd5e1' }}
      >
        Изучаемые слова
      </button>
      <button
        className="button"
        onClick={() => setShowLearned(true)}
        style={{ backgroundColor: showLearned ? '#3b82f6' :
'#cbd5e1' }}
      >
        Изученные слова
      </button>
    </div>

    {/* Кнопки уровней */}
    <div style={{ display: 'flex', gap: '1rem', marginBottom:
'2rem' }}>
      {allLevels.map(lvl => (
        <button
          key={lvl}
          className="button"
          onClick={() => handleLevelClick(lvl)}
          style={{
            backgroundColor: level === lvl ? '#2563eb' :
openLevels.includes(lvl) ? '#3b82f6' : '#cbd5e1',
            cursor: openLevels.includes(lvl) ? 'pointer' :
'not-allowed'
          }}
        >
          {lvl}
        </button>
      ))}
    </div>

    {showLockMessage && (
      <p style={{ color: 'red', marginBottom: '1rem' }}>
        Уровень закрыт. Пройди тест на скорость, чтобы открыть
его.
      </p>
    )}

    {showLearned ? (
      currentLevelWords.slice(0, currentCompletedWords).length
> 0 ? (
        currentLevelWords.slice(0,
currentCompletedWords).map((item, index) => (
          <div

```

```

        key={index}
        style={{
          border: '1px solid #ddd',
          borderRadius: '12px',
          padding: '1rem',
          background: '#ecfdf5',
          marginBottom: '1rem'
        }}
      >
      <p style={{ fontSize: '1.2rem', marginBottom:
'0.5rem' }}>
        <strong>{item.word}</strong> - {item.correct}
      </p>
    </div>
  ))
) : (
  <p className="subtext">Нет изученных слов.</p>
)
) : (
  <>
    {currentWords.map((item, index) => (
      <div
        key={index}
        style={{
          border: '1px solid #ddd',
          borderRadius: '12px',
          padding: '1rem',
          background: learnedWords.includes(item.word) ?
'#ecfdf5' : '#fff',
          marginBottom: '1rem'
        }}
      >
      <p style={{ fontSize: '1.2rem', marginBottom:
'0.5rem' }}>
        <strong>{item.word}</strong> - {item.correct}
      </p>
      {learnedWords.includes(item.word) ? (
        <button className="buttonDictionary" onClick={()
=> markAsUnknown(item.word)}>
          Убрать из изученных
        </button>
      ) : (
        <button className="buttonDictionary" onClick={()
=> markAsKnown(item.word)}>
          Отметить как выученное
        </button>
      )}
    </div>
  )}
)

{page < totalPages - 1 && (
  <button className="button" onClick={handleNextPage}>
    Следующая страница
  </button>
)}

```

```

        </button>
      )}
    </>
  )}
</div>
)
}

```

Листинг A.17 – Код модуля «Home.jsx»

```

import { useNavigate } from 'react-router-dom'
import logo from '../assets/logo.svg'

export default function Home() {
  const navigate = useNavigate()

  return (
    <div className="container">
      <div style={{
        display: 'flex',
        alignItems: 'center',
        gap: '1rem',
        marginBottom: '0.5rem'
      }}>
        <img
          src={logo}
          alt="Logo"
          style={{ width: '64px', height: '64px' }}
        />
        <h1 className="heading" style={{ margin: 0
}}>Polyglot</h1>
      </div>
      <div className="home-description">
        <h3>Открой мир языков вместе с Polyglot</h3>
        <p>
          Polyglot – это твой личный помощник в изучении иностранных
          языков. Учись в удобном темпе,
          играй, запоминай и отслеживай прогресс – всё в одном
          приложении.
        </p>

        <div className="features-grid">
          <div className="features-column">
            <div className="feature-item"><span>📖</span> 5 уровней
            сложности – от A1 до C1</div>
            <div className="feature-item"><span>⚡</span> Спринт-игры на
            скорость запоминания</div>
            <div className="feature-item"><span>🎯</span> Индивидуальный
            путь обучения</div>
          </div>

          <div className="features-divider"></div>

```

```

    <div className="features-column">
      <div className="feature-item"><span>📖</span> Интерактивный
словарь с тренировками</div>
      <div className="feature-item"><span>📈</span> Прогресс по
каждому курсу и уровню</div>
      <div className="feature-item"><span>👤</span> Аватар и
настройка профиля</div>
    </div>
  </div>

  <p style={{ marginTop: '3rem' }}>
    Начни изучать новый язык уже сегодня — это просто, бесплатно
и весело!
  </p>
</div>

  <button data-testid="home-login" className="button"
onClick={() => navigate('/login')}>
    Войти
  </button>

  <svg
    viewBox="0 0 1440 320"
    style={{
      position: 'absolute',
      bottom: 0,
      left: 0,
      width: '100%',
      zIndex: -1,
    }}
  >
    <path
      fill="#e0f2fe"
      fillOpacity="0.8"
      d="M0,224L30,202.7C60,181,120,139,180,149.3C240,160,300,160,360,250.7C420,277,480,267,540,229.3C600,192,660,128,720,101.3C780,75,840,85,900,90.7C960,96,1020,96,1080,90.7C1140,85,1200,75,1260,74.7C1320,75,1380,85,1410,90.7L1440"
    ></path>
  </svg>
</div>
)
}

```

Листинг А.18 – Код модуля «Login.jsx»

```

import { useState } from 'react'
import { useNavigate } from 'react-router-dom'
import RegisterForm from '../components/registerForm'
import LogoHeader from '../components/logoHeader'

export default function Login() {
  const navigate = useNavigate()

```

```

const [username, setUsername] = useState('')
const [password, setPassword] = useState('')
const [error, setError] = useState('')
const [showRegister, setShowRegister] = useState(false)

const handleLogin = async (e) => {
  e.preventDefault()
  setError('')

  try {
    const response = await
fetch('http://localhost:5000/api/auth/login', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    login: username,
    password: password
  })
})

    const data = await response.json()

    if (data.success) {
      localStorage.setItem('user',
JSON.stringify(data.user)) // ← сохраняем весь объект
      navigate('/profile')
    } else {
      setError(data.message || 'Ошибка входа')
    }
  } catch (err) {
    setError('Ошибка подключения к серверу')
  }
}

return (
  <div className="container">
    <LogoHeader />
    <h2 className="heading">{showRegister ? 'Регистрация' :
'Вход в Polyglot'}</h2>

    {showRegister ? (
      <RegisterForm onCancel={() => setShowRegister(false)} />
    ) : (
      <>
        <form
          onSubmit={handleLogin}
          style={{ display: 'flex', flexDirection: 'column',
width: '300px' }}
        >
          <input
            type="text"

```

```

        placeholder="Логин"
        value={username}
        onChange={ (e) => setUsername(e.target.value) }
        className="input"
        required
    />
    <input
        type="password"
        placeholder="Пароль"
        value={password}
        onChange={ (e) => setPassword(e.target.value) }
        className="input"
        required
    />
    <button className="button" type="submit" data-
testid="login-submit">Войти</button>
    </form>

    {error && <p data-testid="login-error" style={{ color:
'red', marginTop: '1rem' }}>{error}</p>}

    <p style={{ marginTop: '1.5rem' }}>
        Нет аккаунта?{' '}
        <span
            onClick={() => {
                setShowRegister(true)
                setError('')
            }}
            style={{ color: '#3b82f6', cursor: 'pointer',
textDecoration: 'underline' }}
        >
            Зарегистрироваться
        </span>
    </p>
</>
    )}
</div>
)
}

```

Листинг A.19 – Код модуля «Profile.jsx»

```

import { useEffect, useState } from 'react'
import { useNavigate } from 'react-router-dom'
import LogoHeader from '../components/logoHeader'
import avatarPlaceholder from '../assets/avatar-placeholder.jpg'
import { FaEdit } from 'react-icons/fa'

export default function Profile() {
    const navigate = useNavigate()
    const user = JSON.parse(localStorage.getItem('user'))

```

```

    const [avatar, setAvatar] = useState(user?.avatarBase64 ||
avatarPlaceholder)
    const [nickname, setNickname] = useState(user?.nickname)
    const [editing, setEditing] = useState(false)
    const [newNickname, setNewNickname] = useState(nickname)
    const [hovered, setHovered] = useState(false)
    const userId = user?.id

    const [userCourses, setUserCourses] = useState([])

    useEffect(() => {
        async function fetchCourses() {
            const courseList = ['english', 'french', 'spanish']
            const results = await Promise.all(
                courseList.map(async (course) => {
                    const res = await
fetch(`http://localhost:5000/api/course/${userId}/${course}`)
                    const data = await res.json()
                    return {
                        course,
                        level: data?.levelCode || 'A1'
                    }
                })
            )
            setUserCourses(results)
        }

        fetchCourses()
    }, [userId])

    const handleSave = async () => {
        const trimmed = newNickname.trim()
        if (!trimmed) return

        try {
            const response = await
fetch(`http://localhost:5000/api/user/${userId}/nickname`, {
                method: 'PUT',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify({ nickname: trimmed })
            })

            const data = await response.json()

            if (data.success) {
                setNickname(trimmed)
                setNewNickname('')
                setEditing(false)
                localStorage.setItem('nickname', trimmed) // обновим в
хранилище
            } else {
                alert(data.message || 'Ошибка при обновлении ника')
            }
        }
    }

```

```

    }
  } catch (err) {
    alert('Ошибка соединения с сервером')
  }
}

const handleAvatarChange = () => {
  const input = document.createElement('input')
  input.type = 'file'
  input.accept = 'image/*'

  input.onChange = async (e) => {
    const file = e.target.files[0]
    if (!file) return

    const reader = new FileReader()
    reader.onload = async () => {
      const base64 = reader.result
      const user = JSON.parse(localStorage.getItem('user'))
      const userId = user?.id

      try {
        const response = await
fetch(`http://localhost:5000/api/user/${userId}/avatar`, {
  method: 'PUT',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ avatarBase64: base64 })
})

        const data = await response.json()

        if (data.success) {
          setAvatar(base64)
          const updatedUser = { ...user, avatarBase64: base64
}
          localStorage.setItem('user',
JSON.stringify(updatedUser))
        } else {
          alert(data.message || 'Ошибка при загрузке аватара')
        }
      } catch (err) {
        alert('Сервер недоступен')
      }
    }

    reader.readAsDataURL(file)
  }

  input.click()
}

return (

```

```

<div className="container" >
  <LogoHeader />
  <div
    style={{ position: 'relative', marginBottom: '1rem' }}
    onMouseEnter={() => setHovered(true)}
    onMouseLeave={() => setHovered(false)}
  >
    <img
      src={avatar}
      alt="Аватар"
      style={{
        width: '120px',
        height: '120px',
        borderRadius: '50%',
        objectFit: 'cover',
        border: '3px solid #3b82f6'
      }}
    />
    {hovered && (
      <button
        data-testid="save-nickname"
        onClick={handleAvatarChange}
        style={{
          position: 'absolute',
          bottom: '-10px',
          left: '50%',
          transform: 'translateX(-50%)',
          background: '#3274D3',
          color: '#fff',
          padding: '0.3rem 0.7rem',
          fontSize: '1rem',
          border: 'none',
          borderRadius: '8px',
          cursor: 'pointer',
          boxShadow: '0 2px 6px rgba(0,0,0,0.15)',
          whiteSpace: 'nowrap'
        }}
      >
        Изменить аватар
      </button>
    )}
  </div>

  {/* Никнейм и редактирование */}
  <div style={{ display: 'flex', alignItems: 'center', gap:
'0.5rem', marginBottom: '1rem' }}>
    {editing ? (
      <>
        <input
          type="text"
          className="input"
          value={newNickname}
          onChange={(e) => setNewNickname(e.target.value)}

```

```

        />
        <button data-testid="save-nickname"
className="button" onClick={handleSave}>Сохранить</button>
    </>
    ) : (
    <>
        <h2 className="heading" style={{ margin: 0
}}>{nickname}</h2>
        <span data-testid="edit-nickname">
            <FaEdit
                onClick={() => setEditing(true)}
                style={{ cursor: 'pointer', color: '#3274D3',
width: '30px', height: '30px' }}
                title="Редактировать"
            />
        </span>
    </>
    )}
</div>

<div
    style={{
        backgroundColor: '#fff',
        padding: '1rem 2rem',
        borderRadius: '12px',
        boxShadow: '0 2px 8px rgba(0,0,0,0.05)',
        marginBottom: '2rem',
        width: '300px'
    }}
>
    <h3 style={{ marginBottom: '0.5rem' }}>Статистика по
курсам</h3>
    {userCourses.map((entry) => {
        const allLevels = ['A1', 'A2', 'B1', 'B2', 'C1']
        const currentIndex = allLevels.indexOf(entry.level)
        const progressPercent = Math.round(((currentIndex) /
allLevels.length) * 100)

        return (
            <div key={entry.course} style={{ marginBottom:
'1rem' }}>
                <p style={{ marginBottom: '0.25rem'
}}>{entry.course.toUpperCase()}</p>
                <div style={{ background: '#e5e7eb', height:
'8px', borderRadius: '4px' }}>
                    <div
                        style={{
                            background: '#3b82f6',
                            width: `${progressPercent}%`,
                            height: '100%',
                            borderRadius: '4px'
                        }}
                    ></div>
                </div>
            </div>
        )
    })}

```

```

        </div>
        <p style={{ fontSize: '0.8rem', color: '#6b7280'
    }}>
        Уровень: {entry.level} из C1
    </p>
</div>
    )
    }
}
</div>

    <button className="button" onClick={() =>
navigate('/courses')}>
        Перейти к курсам
    </button>
</div>
)
}

```

Листинг А.20 – Код модуля «SprintGame.jsx»

```

import { useEffect, useState } from 'react'
import { useNavigate } from 'react-router-dom'
import wordsEn from '../assets/wordsForEnglishCourse.json'
import wordsFr from '../assets/wordsForFrenchCourse.json'
import wordsEs from '../assets/wordsForSpanishCourse.json'

const courseWords = {
  english: wordsEn,
  french: wordsFr,
  spanish: wordsEs
}

export default function SprintGame({ courseId, level,
refreshLevel }) {
  const navigate = useNavigate()
  const [currentIndex, setCurrentIndex] = useState(0)
  const [score, setScore] = useState(0)
  const [timeLeft, setTimeLeft] = useState(30)
  const [isFinished, setIsFinished] = useState(false)
  const [nextLevelUnlocked, setNextLevelUnlocked] =
useState(false)

  const wordsData = courseWords[courseId] || []
  const levelWords = wordsData.filter(w => w.level === level)
  const currentWord = levelWords[currentIndex] || null

  useEffect(() => {
    if (timeLeft <= 0 || currentIndex >= levelWords.length) {
      setIsFinished(true)
    }

    if (!isFinished) {

```

```

        const timer = setTimeout(() => {
            setTimeLeft(prev => prev - 1)
        }, 1000)
        return () => clearTimeout(timer)
    }
}, [timeLeft, currentIndex, isFinished])

useEffect(() => {
    if (timeLeft <= 0 || currentIndex >= levelWords.length) {
        setIsFinished(true)

        const successRate = score / levelWords.length
        const user = JSON.parse(localStorage.getItem('user'))
        const userId = user?.id
        const nextLevel = getNextLevel(level)

        if (userId) {
            const updates = {
                sprintScore: successRate >= 0.95 ? 0 :
Math.round(successRate * 100)
            }

            if (successRate >= 0.95 && nextLevel) {
                updates.levelCode = nextLevel
                updates.completedWords = 0
                setNextLevelUnlocked(true)
            }

            fetch(`http://localhost:5000/api/course/${userId}/update`,
{
            method: 'PUT',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({
                courseName: courseId,
                updates
            })
        })
        .then(res => res.json())
        .then(data => {
            if (data.success) {
                console.log('Данные обновлены:', data)
            } else {
                console.error('Ошибка при обновлении:',
data.message)
            }
        })
        .catch(err => {
            console.error('Ошибка запроса:', err)
        })
    }
})

if (!isFinished) {

```

```

    const timer = setTimeout(() => {
      setTimeLeft(prev => prev - 1)
    }, 1000)
    return () => clearTimeout(timer)
  }
}, [timeLeft, currentIndex, isFinished])

const handleAnswer = (answer) => {
  if (answer === currentWord.correct) {
    setScore(score + 1)
  }
  setCurrentIndex(currentIndex + 1)
}

const restartGame = async () => {
  await refreshLevel() // вызываем функцию из родителя

  setCurrentIndex(0)
  setScore(0)
  setTimeLeft(30)
  setIsFinished(false)
  setNextLevelUnlocked(false)
}

const getNextLevel = (level) => {
  const levels = ['A1', 'A2', 'B1', 'B2', 'C1']
  const index = levels.indexOf(level)
  return index < levels.length - 1 ? levels[index + 1] : null
}

if (isFinished) {
  return (
    <div className="container">
      <h2 className="heading">Результат</h2>
      <p className="subtext">
        Вы правильно перевели {score} из {levelWords.length}
        слов
      </p>

      {nextLevelUnlocked ? (
        <p className="subtext" style={{ color: 'green',
marginTop: '1rem' }}>
 Поздравляем! Уровень {getNextLevel(level)}
разблокирован.
        </p>
      ) : (
        <p className="subtext" style={{ color: 'gray',
marginTop: '1rem' }}>
        Для открытия следующего уровня нужно не менее 95%
        правильных ответов : (
        </p>
      )
    )
  )
}

```

```

    })

    <div style={{ display: 'flex', gap: '1rem' }}>
      <button
        className="button"
        onClick={() => navigate(`/courses/${courseId}`)}
      >
        ← Вернуться в курс
      </button>
      <button className="button"
onClick={restartGame}>Начать заново</button>

    </div>
  </div>
)
}

return (
  <div className="container">
    <h2 className="heading">Спринт — уровень {level}</h2>
    <p style={{ fontSize: '1.2rem', marginBottom: '1rem' }}>
      Время: <strong>{timeLeft}</strong>s</p>

    {currentWord && (
      <>
        <p style={{ fontSize: '1.5rem', marginBottom: '1rem' }}>
          Слово: <strong>{currentWord.word}</strong>
        </p>
        <div style={{ display: 'grid', gap: '1rem', maxWidth:
'300px', width: '100%' }}>
          {currentWord.options.map((option, index) => (
            <button
              key={index}
              className="button"
              style={{ padding: '0.6rem' }}
              onClick={() => handleAnswer(option)}
            >
              {option}
            </button>
          ))}
        </div>
      </>
    )}

    <p style={{ marginTop: '2rem', color: '#6b7280' }}>
      Правильных ответов: {score}
    </p>
  </div>
)
}

```

Листинг A.21 – Код модуля «DictionaryRoute.jsx»

```
import { useParams, useNavigate } from 'react-router-dom'
import DictionaryPage from '../pages/DictionaryPage'
import LogoHeader from '../components/logoHeader'

export default function DictionaryRoute() {
  const { lang } = useParams()
  const navigate = useNavigate()

  return (
    <div className="container">
      <LogoHeader />
      <DictionaryPage courseId={lang} />
      <button
        className="button"
        style={{ margin_bottom: "2rem" }}
        onClick={() => navigate(`/courses/${lang}`)}
      >
        ← Вернуться в курс
      </button>
    </div>
  )
}
```

Листинг A.22 – Код модуля «SprintRoutes.jsx»

```
import { useParams, useLocation } from 'react-router-dom'
import { useState } from 'react'
import SprintGame from '../pages/SprintGame'
import LogoHeader from '../components/logoHeader'

export default function SprintRoute() {
  const { lang } = useParams()
  const location = useLocation()
  const [level, setLevel] = useState(location.state?.levelCode || 'A1')
  const user = JSON.parse(localStorage.getItem('user'))
  const userId = user?.id

  const refreshLevel = async () => {
    try {
      const response = await
fetch(`http://localhost:5000/api/course/${userId}/${lang}`)
      const data = await response.json()
      setLevel(data?.levelCode || 'A1')
    } catch (err) {
      console.error('Ошибка при обновлении уровня:', err)
    }
  }

  return (
    <div className="container">
```

```

        <LogoHeader />
        <SprintGame courseId={lang} level={level}
refreshLevel={refreshLevel} />
    </div>

    )
}

```

Листинг А.23 – Код модуля «main.jsx»

```

import React from 'react'
import ReactDOM from 'react-dom/client'
import { RouterProvider } from 'react-router-dom'
import router from './router'
import './App.css'

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <RouterProvider router={router} />
  </React.StrictMode>
)

```

Листинг А.24 – Код модуля «router.jsx»

```

import { createBrowserRouter } from 'react-router-dom'
import Home from './pages/Home'
import Courses from './pages/Courses'
import Login from './pages/Login'
import Profile from './pages/Profile'

import CoursePage from './pages/CoursePage'
import AITrainer from './pages/AITrainer'
import DictionaryRoute from './routes/DictionaryRoute'
import SprintRoute from './routes/SprintRoutes'
const router = createBrowserRouter([
  { path: '/', element: <Home /> },
  { path: '/login', element: <Login /> },
  { path: '/profile', element: <Profile /> },
  { path: '/courses', element: <Courses /> },
  { path: '/courses/:lang', element: <CoursePage /> },
  { path: '/courses/:lang/dictionary', element:
<DictionaryRoute /> },
  { path: '/courses/:lang/sprint', element: <SprintRoute /> },
  { path: '/courses/:lang/ai', element: <AITrainer /> }
],
{
  basename: '/PolyglotApp'
})
export default router

```

Листинг A.25 – Код модуля «index.html»

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/logo.svg" />
    <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
    <title>Polyglot</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

Листинг A.26 – Код модуля «.env»

```
DB_USER=polyglot
DB_PASS=pass
DB_SERVER=localhost
DB_NAME=polyglot
```

Листинг A.27 – Код модуля «App.css»

```
body {
  margin: 0;
  font-family: 'Segoe UI', sans-serif;
  background: linear-gradient(210deg, #dfebf3, #70c4fb);
  color: #1f2937;
  min-height: 100vh;
  overflow-x: hidden;
  position: relative;
}

.container {
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  min-height: 100vh;
  text-align: center;
}

.heading {
  font-size: 3rem;
  font-weight: 600;
  margin-bottom: 1.5rem;
}

.courseHeading {
  font-size: 3rem;
```

```

    font-weight: 600;
    margin-bottom: 1rem;
    background: #ffffff;
    border-radius: 15px;
    padding: 0.75rem 1rem;
}

.subtext {
    font-size: 1.5rem;
    color: #292c31;
    max-width: 500px;
}

.button {
    padding: 0.75rem 1rem;
    font-size: 1.5rem;
    margin-top: 1rem;
    background-color: rgb(50 116 211);
    color: white;
    border: none;
    border-radius: 8px;
    cursor: pointer;
    box-shadow: 0 4px 8px rgba(109, 95, 224, 0.2);
    transition: background-color 0.3s ease, transform 0.2s ease;
    margin-bottom: 0.5rem;
}

.button:hover {
    background-color: rgb(59 100 159);
    transform: translateY(-2px);
}

.buttonDictionary {
    padding: 0.75rem 1rem;
    font-size: 1.2rem;
    margin-top: 0.5rem;
    margin-bottom: 0.5rem;
    background-color: rgb(50 116 211);
    color: white;
    border: none;
    border-radius: 8px;
    cursor: pointer;
    box-shadow: 0 4px 8px rgba(109, 95, 224, 0.2);
    transition: background-color 0.3s ease, transform 0.2s ease;
}

.buttonDictionary:hover {
    background-color: rgb(59 100 159);
    transform: translateY(-2px);
}

.input {
    padding: 0.6rem 1rem;

```

```

margin-bottom: 1rem;
border: 1px solid #ccc;
border-radius: 10px;
font-size: 1rem;
outline: none;
transition: border-color 0.2s ease, box-shadow 0.2s ease;
}

.input:focus {
border-color: #3b82f6;
box-shadow: 0 0 0 3px rgba(59, 130, 246, 0.2);
}

.home-description {
backdrop-filter: blur(16px) saturate(120%);
-webkit-backdrop-filter: blur(16px) saturate(120%);
background-color: rgba(255, 255, 255, 0.65);
border-radius: 20px;
border: 1px solid rgba(255, 255, 255, 0.3);
padding: 1.35rem;
max-width: 800px;
margin: 1rem auto;
box-shadow: 0 8px 24px rgba(0, 0, 0, 0.08);
text-align: center;
font-size: 1.05rem;
line-height: 1.7;
transition: all 0.3s ease;
}

.home-description h3 {
font-size: 1.7rem;
font-weight: 700;
margin-bottom: 3rem;
color: #1e293b;
}

.home-description ul {
list-style: none;
padding-left: 0;
margin-top: 1rem;
}

.home-description ul li {
display: flex;
align-items: center;
gap: 0.6rem;
margin: 0.6rem 0;
font-size: 1rem;
}

.features-grid {
display: flex;

```

```

    justify-content: center;
    align-items: stretch;
    gap: 2rem;
    margin-top: 2rem;
    flex-wrap: wrap;
    opacity: 0;
    transform: translateY(20px);
    animation: fadeInUp 0.8s ease-out forwards;
}

@keyframes fadeInUp {
  to {
    opacity: 1;
    transform: translateY(0);
  }
}

.features-column {
  display: flex;
  align-items: center;
  flex-direction: column;
  gap: 1rem;
  flex: 1;
  min-width: 240px;
}

.features-divider {
  width: 2px;
  background-color: rgba(0, 0, 0, 0.1);
  border-radius: 1px;
  margin-top: 0.5rem;
  margin-bottom: 0.5rem;
}

.feature-item {
  display: flex;
  align-items: center;
  gap: 0.6rem;
  font-size: 1rem;
  transition: background 0.3s, transform 0.3s;
  padding: 0.3rem 0.6rem;
  border-radius: 8px;
}

.feature-item:hover {
  background-color: rgba(255, 255, 255, 0.3);
  transform: scale(1.05);
  cursor: default;
}

```

Приложение Б

(рекомендуемое)

Результаты тестирования

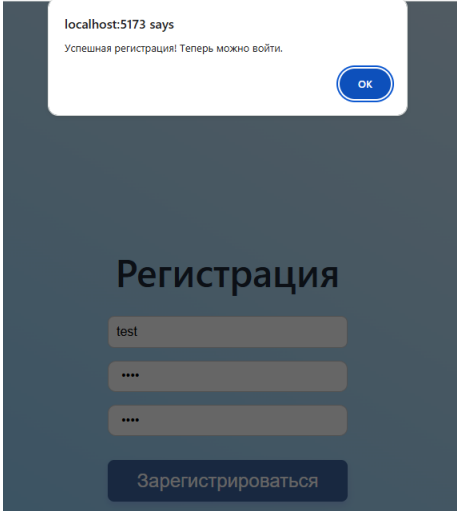
```

Running 5 tests using 3 workers
ok 1 tests\profile.spec.js:13:3 > Профиль > Редактирование никнейма (990ms)
ok 2 tests\auth.spec.js:4:3 > Аутентификация > Успешный логин (583ms)
ok 3 tests\courses.spec.js:14:3 > Курсы > Просмотр курсов (665ms)
ok 4 tests\auth.spec.js:16:3 > Аутентификация > Ошибка логина (435ms)
ok 5 tests\courses.spec.js:19:3 > Курсы > Переход в курс (499ms)

5 passed (2.3s)
PS D:\dipPolyglot\Polyglot>

```

Рисунок Б.1 – Результаты выполнения автоматических тестов



localhost:5173 says
Успешная регистрация! Теперь можно войти.

Регистрация

test

....

....

Зарегистрироваться

Результаты		Сообщения	
	id	login	password
1	3	WWW	\$2b\$10\$Q9JcYjytiHQIs4pB4UuPOobnYd9Cpy8N8Qw4WAq3h/f6/....
2	4	userTest	\$2b\$10\$G5SwEheVwMBD9L2hptUQEuekeb.PE4/WYP53LH2RP...
3	5	testuser	\$2b\$10\$lkWYVgTv3so94NRTrlXwa.l4lf1twMVViPHP8SwzXI5NFa...
4	6	test	\$2b\$10\$XZWRFGKPD4d.oY9WxEDBEo\$TLS6qLNhexsJf1b44uP...

Рисунок Б.2 – Тестирование процесса регистрации

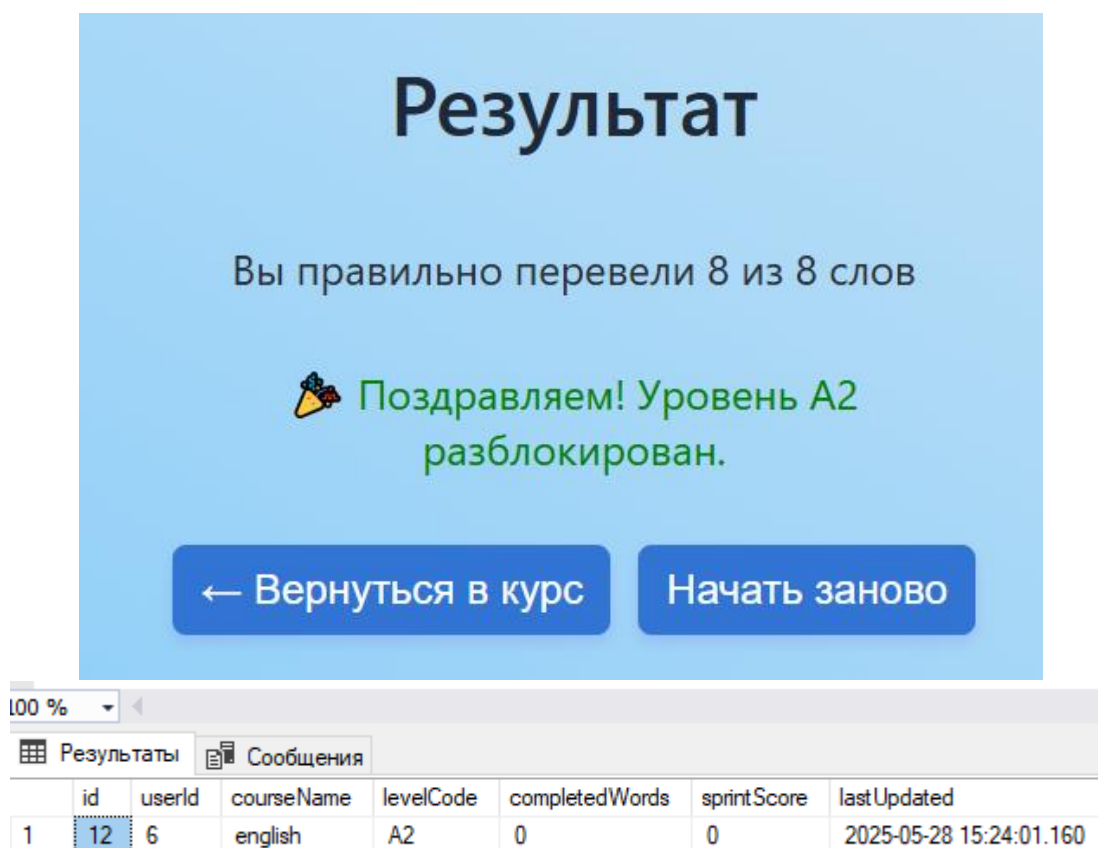


Рисунок Б.3 – Тестирование повышения уровня курса

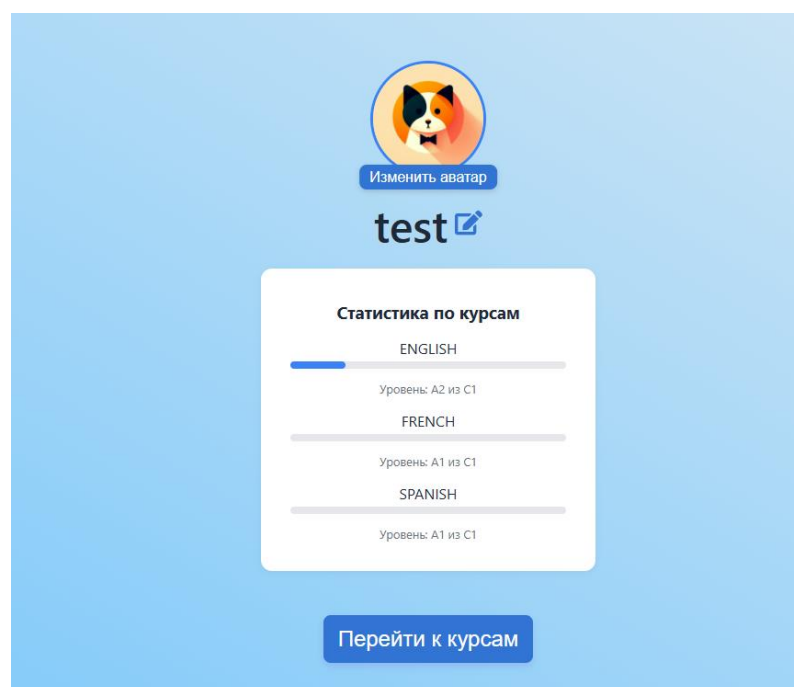


Рисунок Б.4 – Появление кнопки «Изменить аватар» при наведении на аватар профиля

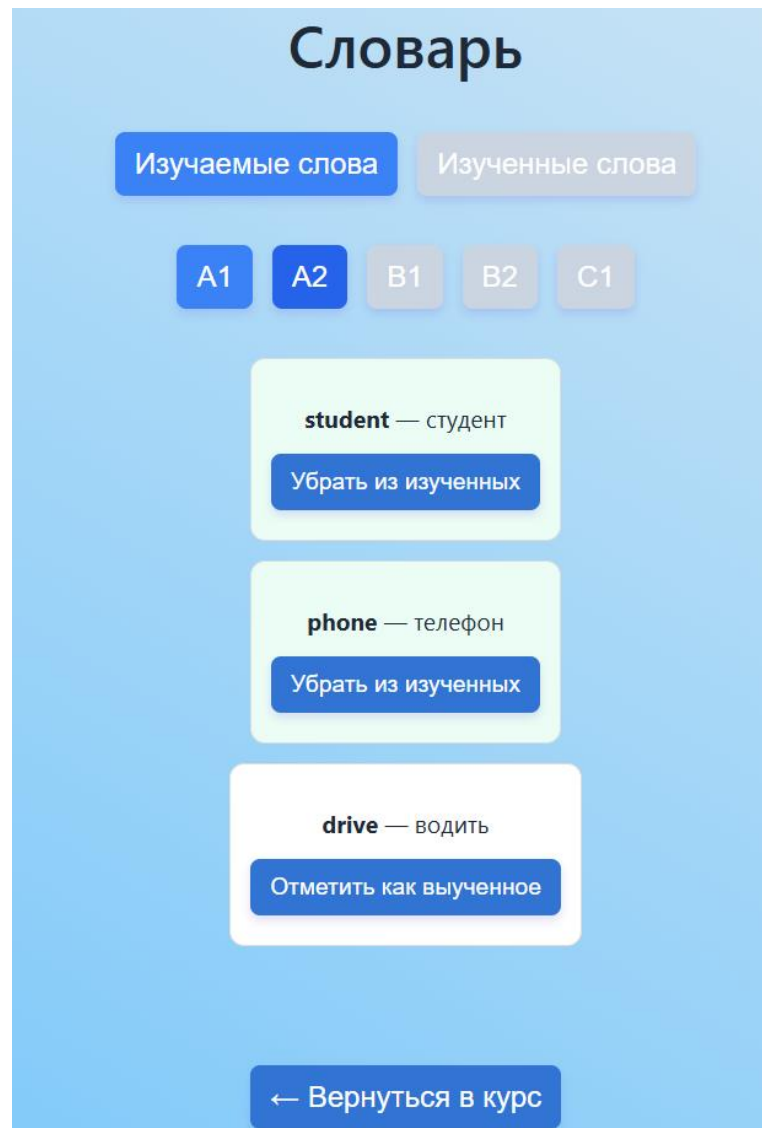


Рисунок Б.5 – Изменение стилей изученных слов

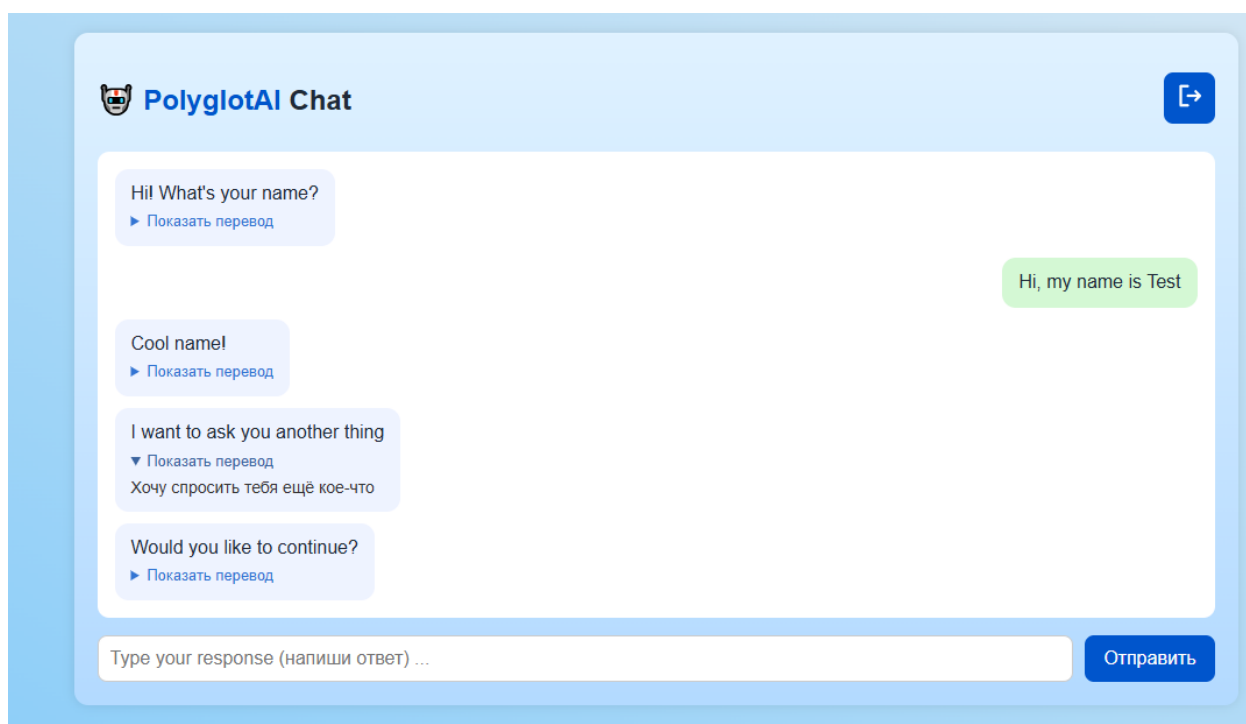


Рисунок Б.6 – Отображение переводов в чате с ИИ

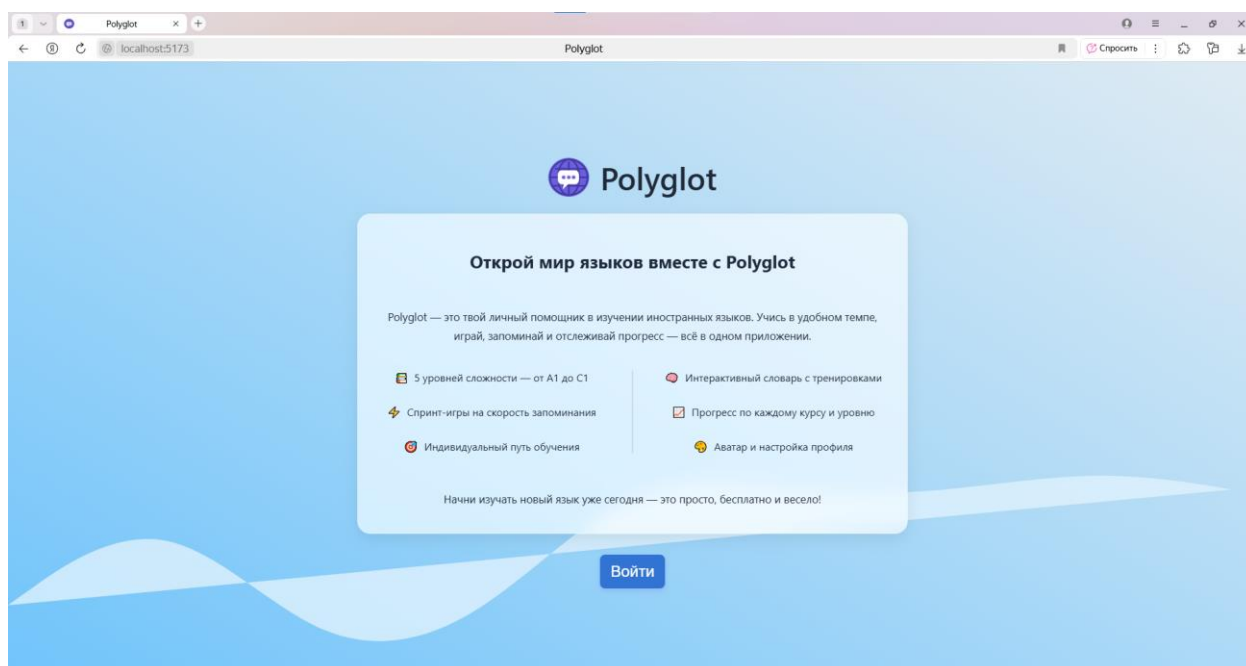


Рисунок Б.7 – Открытие главной страницы в браузере «Яндекс Браузер»

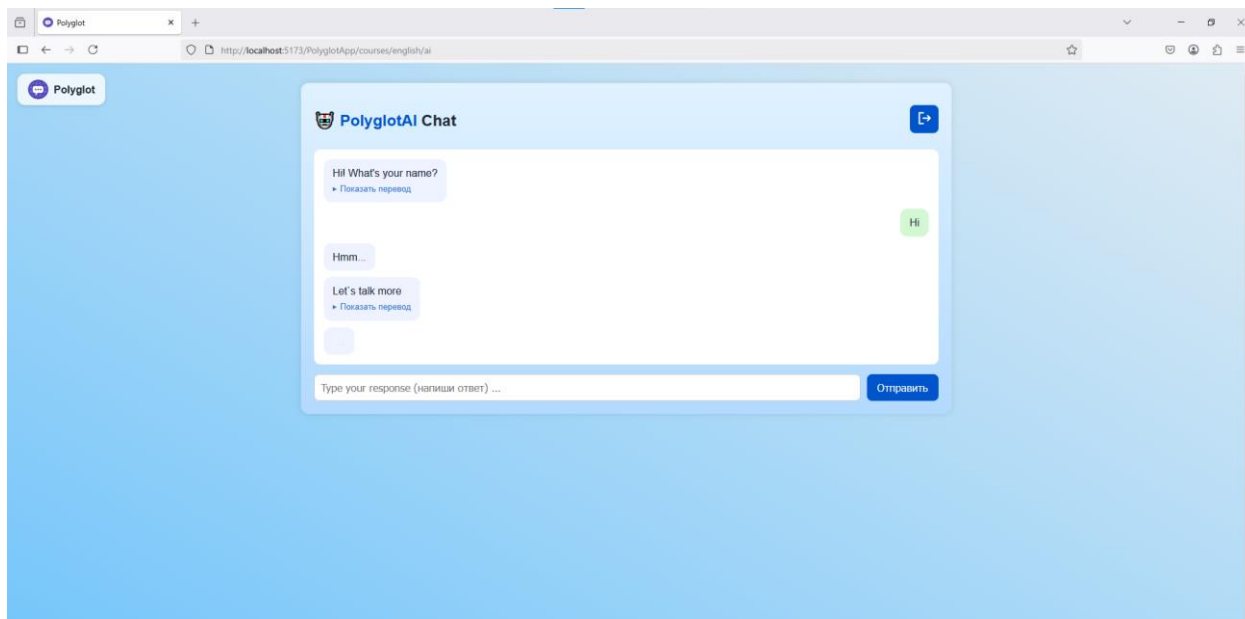


Рисунок Б.8 – Страница чата с ИИ в браузере «Mozilla Firefox»

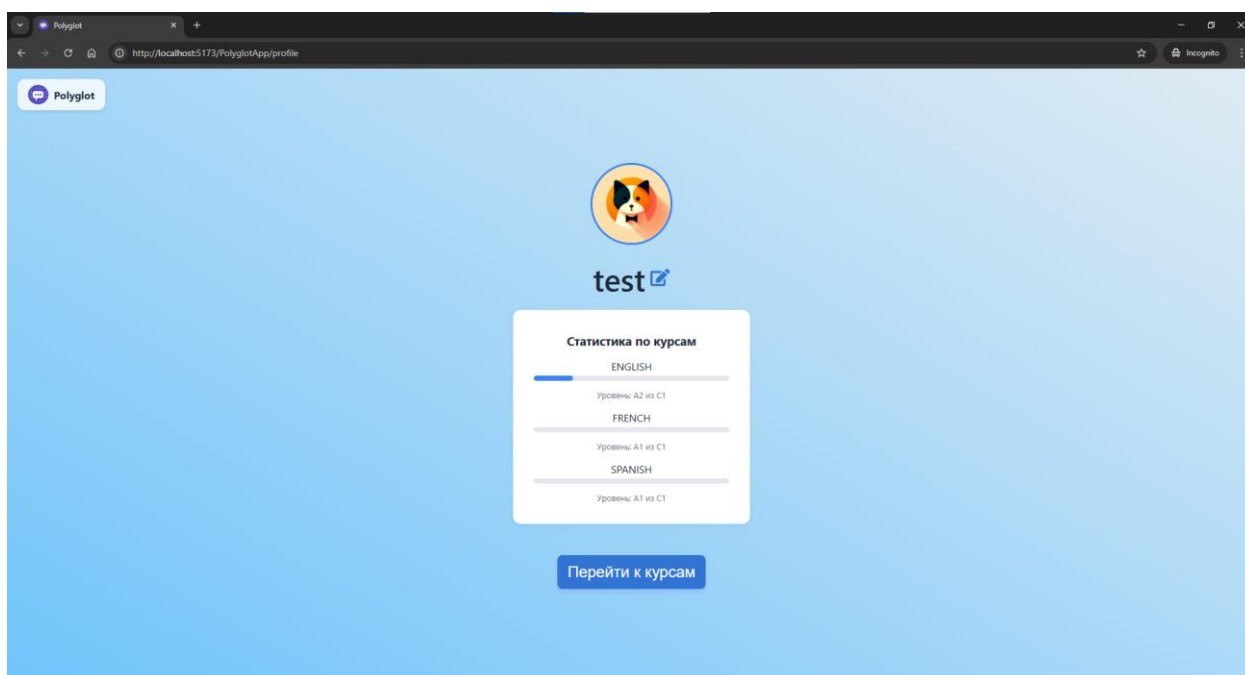


Рисунок Б.8 – Страница профиля в браузере «Google Chrome»