МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ СЕРВИСА С ОБРАЗОВАТЕЛЬНЫМИ ИГРАМИ ДЛЯ ДЕТЕЙ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 411 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета КНиИТ
Карасева Арсения Михайловича

Научный руководитель
доцент, к.т.н. _____ М. В. Хамутова

Заведующий кафедрой
к.ф.-м.н., доцент _____ С. В. Миронов

Саратов 15 июня 2024 г.

ВВЕДЕНИЕ

Образовательные игры предоставляют уникальную возможность интегрировать знания и навыки в интерактивной и захватывающей форме, что способствует лучшему усвоению материала и развитию критического мышления. В данной работе фокус сосредотачивается на создании серверной части сервиса, обеспечивающей стабильное и эффективное взаимодействие между клиентскими приложениями и серверной частью приложения.

Развитие области образовательных технологий и игрового обучения предоставляет возможность создания инновационных решений, способствующих разностороннему развитию детей. Серверная часть представляет собой основной каркас системы, обеспечивая обработку данных, безопасность и устойчивость работы. Разработка этой части сервиса является неотъемлемой частью создания интегрированной и эффективной платформы образовательных игр.

Цели данной работы — обучение различным вариантам построения архитектуры приложений, изучение и совершенствование знаний о технологиях, которые предназначены для разработки серверной части веб-приложения, получение навыков интеграции со сторонними сервисами.

Задачи данной работы — спроектировать и разработать backend-часть приложения сервиса детских образовательных игр «EduPlay». Требуется реализовать структуру базы данных приложения, слой доступа к данным, слой бизнес-логики, слой WebAPI, а также развернуть данную часть приложения на удалённом сервере.

Структура и объём работы.

Для решения поставленных задач выполнена выпускная квалификационная работа, включающая в себя введение, 2 основные главы, заключение, список использованных источников из 20 наименований и 5 приложений. Работа изложена на 82 страницах, содержит 13 рисунков.

Первая глава имеет название «Анализ используемых технологий и подходов» и содержит основную теоретическую информацию о технологиях и подходах, которые использовались при разработке серверного приложения.

Вторая глава имеет название «Разработка серверного приложения», данная глава содержит подробное описание процесса выполнения работы.

Выпускная квалификационная работа заканчивается заключением, списком использованных источников, а также приложениями с кодом А-Д.

1 Основное содержание работы

1.1 Архитектура приложения

Монолитная архитектура

Монолитная архитектура представляет собой подход к организации приложения, в котором все компоненты интегрированы в одно целое. В такой архитектуре, всё приложение разворачивается как единое целое и масштабируется или обновляется в целом. При изменении или обновлении одной части приложения, необходимо перекомпилировать всё приложение, что может быть сложным и затратным процессом.

В монолитной архитектуре, все компоненты приложения обмениваются данными напрямую без использования сетевых протоколов или промежуточного слоя. Это означает, что один компонент может быть зависим от других, и изменение в одном компоненте может приводить к необходимости изменений в других компонентах.

Также, монолитная архитектура не предоставляет гибкости в использовании разных технологий и языков программирования для разных компонентов приложения. Все компоненты должны быть написаны на одном языке программирования и использовать одну технологию.

Монолитная архитектура всё ещё широко используется, особенно в небольших проектах, где сложность приложения не настолько высока. Также, монолитная архитектура может быть полезна в случаях, когда приложение работает с небольшим объемом данных и не требует высокой масштабируемости.

Трёхслойная архитектура

Трёхслойная архитектура приложения — это подход к построению приложений, который разделяет приложение на три уровня: клиентский, бизнес-логики и уровень базы данных.

- Клиентский уровень — это интерфейс, который предоставляет пользователю доступ к функциональности приложения. Этот уровень может содержать веб-страницы, мобильные приложения или другие пользовательские интерфейсы;
- Уровень бизнес-логики — это уровень, который содержит бизнес-логику приложения. Здесь расположены все компоненты, которые отвечают за основную функциональность приложения, такие как обработка запросов, проверка прав доступа и так далее;

— Уровень базы данных — это уровень, который содержит данные, необходимые приложению. Здесь располагаются базы данных и другие инструменты для управления данными, такие как системы управления базами данных, хранилища данных и так далее.

Эта архитектура позволяет отделить слой пользовательского интерфейса от слоя бизнес-логики и базы данных, что упрощает процесс разработки, поддержки и масштабирования приложения.

1.2 СУБД PostgreSQL

PostgreSQL — это мощная объектно-реляционная система управления базами данных (СУБД), которая широко используется для хранения, управления и обработки структурированных данных.

PostgreSQL обладает следующими преимуществами:

— PostgreSQL полностью поддерживает стандарт SQL и предоставляет богатый набор расширенных возможностей, включая сложные запросы, оконные функции, подзапросы;

— Возможность создания собственных типов данных, функций, операторов и языков программирования делает PostgreSQL высоко расширяемым и позволяет адаптировать его под конкретные потребности проекта;

— PostgreSQL гарантирует ACID-свойства (Атомарность, Согласованность, Изолированность, Долговечность) транзакций, что обеспечивает целостность данных и предотвращает их потерю в случае сбоев;

1.3 Платформа .NET Core и язык программирования C#

.NET Core - это кросс-платформенная, открытая платформа, разрабатываемая корпорацией Microsoft. Она представляет собой современную, высокопроизводительную платформу для создания приложений на языках программирования C#, F# и VB.NET. Она поддерживает работу на операционных системах Linux, macOS и Windows. Составные части платформы .NET Core включают в себя такие элементы, как исполняющая среда Common Language Runtime (CLR), библиотеки классов .NET (BCL), уровень доступа к данным ADO.NET и многие другие.

C# — Объектно-ориентированный язык программирования, разработанный в 2001 году инженерами компании Microsoft. Язык разрабатывался в качестве языка для создания приложения для платформ .NET Framework и .NET

Core, то есть C# относится к языкам из семейства .NET. Поскольку C# является C-подобным языком, то его синтаксис близок к синтаксису таких языков, как C++ и Java.

1.4 ASP.NET Core

ASP.NET Core — кроссплатформенный фреймворк для разработки современных веб-приложений на платформе .NET. Важно сказать, что эта технология разрабатывается компанией Microsoft совместно с сообществом, так как у технологии ASP.NET Core открытый исходный код. ASP.NET Core совместим с такими операционными системами, как Windows, Linux и macOS, в то время, как ASP.NET был совместим только с Windows.

ASP.NET Core позволяет разрабатывать веб-приложения с помощью различных моделей разработки, среди которых есть модель Web API, которая применялась при разработке данного приложения. Также доступны такие модели разработки, как Minimal API, ASP.NET Core MVC, Razor Pages и Blazor. Размещать ASP.NET Core приложение можно на таких веб-серверах, как IIS, Kestrel, Apache, Nginx.

1.5 Entity Framework Core

Entity Framework Core — это типовая система объектно-реляционного отображения (ORM), которая позволяет разрабатывать приложения, использующие базы данных, с помощью объектно-ориентированного подхода. Она предоставляет доступ к данным, хранящимся в базах данных, представляя их в виде объектов .NET. На физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает Entity Framework Core, мы уже работаем с объектами. Entity Framework Core обеспечивает ряд преимуществ, таких как упрощение кода, уменьшение количества необходимой кодировки, повышение безопасности, повышение производительности, удобство тестирования и многое другое.

1.6 IdentityServer4

IdentityServer4 — это фреймворк с открытым исходным кодом для реализации сервера аутентификации и авторизации в приложении. Он реализует самые популярные протоколы аутентификации и авторизации, такие как OAuth 2.0 и OpenID Connect, что позволяет разработчикам создавать безопасные при-

ложения, защищенные паролем и предоставляющие доступ к ресурсам только авторизованным пользователям.

IdentityServer4 работает на платформе .NET Core и может быть использован для разработки различных приложений, таких как веб-сайты, мобильные приложения и даже десктопные приложения. Он позволяет настроить множество параметров и настроек, таких как конфигурация клиента и ресурса, а также может быть расширен с помощью пользовательских хранилищ данных и произвольной логики обработки запросов.

1.7 Docker

Docker — это платформа для разработки, доставки и запуска контейнерных приложений. Docker позволяет создавать контейнеры, автоматизировать их запуск и развертывание, управляет жизненным циклом. Он позволяет запускать множество контейнеров на одной хост-машине.

Контейнеры Docker являются изолированными средами выполнения, которые содержат все необходимые для работы приложения файлы, библиотеки и зависимости. Они могут быть созданы и развернуты на любом компьютере, который поддерживает Docker, без необходимости установки всех зависимостей и конфигурации окружения.

1.8 Программа «Стартап как диплом»

Серверное приложение, разработанное в рамках данной бакалаврской работы, предназначено для университетского проекта «EduPlay», который принимает участие в программе «Стартап как диплом»

«Стартап как диплом» — это программа, которая направлена на вовлечение талантливых студентов в развитие экосистемы технологического предпринимательства, а также на поддержку бизнеса, находящегося на начальной стадии. Разработка и реализация программы обучения абитуриентов и преподавателей университетов в подготовке стартапов в качестве ВКР предусмотрены программой «Цифровая экономика».

EduPlay — это веб-портал, который разрабатывается командой студентов для обучения детей дошкольного возраста, учеников младших классов и родителей, желающих, чтобы обучение приносило их детям удовольствие.

1.9 Реализация серверной части сервиса с образовательными играми для детей

В ходе работы была спроектирована и реализована база данных приложения, а также было разработано само серверное приложение. В качестве СУБД использовалась PostgreSQL, а серверное приложение было разработано на языке C# платформы .NET Core.

Архитектура разработанной системы является монолитной, в то время как архитектура решения представляет из себя трёхслойное приложение — слой доступа к данным (DAL), слой бизнес-логики (BLL), слой клиентской части (PL). В качестве клиентской части серверного приложения был сделан ASP.NET Core WebAPI проект, это удобно для интеграции с клиентской частью самого сервиса.

1.10 Доступ к данным

В разработанном приложении, следуя принципам трёхслойной архитектуры, был сделан отдельный проект для доступа к данным. Слой доступа к данным (DAL — Data Access Layer) состоит из двух проектов (сборок) — проект с интерфейсами (контрактами) для доступа к данным и проект с реализацией этих интерфейсов. Именно из этого слоя приложения взаимодействует с базой данных, это единственная ответственность данного компонента приложения.

Для обеспечения быстрого и лёгкого взаимодействия с базой данных было принято решение реализовать работу с ней с помощью фреймворка «Entity Framework Core». Данная технология позволяет не писать SQL-запросы самостоятельно, а оперирует объектами на уровне серверного приложения для генерации запросов к базе данных.

В папке «DbContexts» находится файл «EduPlayContext.cs» — это контекст базы данных. Контекст базы данных — один из инструментов технологии Entity Framework Core, с помощью которых он организует работу с данными. Entity Framework Core с помощью контекста базы данных реализует шаблон программирования «Unit Of Work» — объединение нескольких репозиториев в один класс для более лёгкой работы с ними. В качестве репозиториев же технология Entity Framework предлагает использовать класс DbSet — он реализует шаблон программирования «Repository», который инкапсулирует работу с базой данных.

1.11 Авторизация пользователей

Регистрация, аутенфикация и авторизация пользователей в разработанном приложении реализована с помощью технологии «IdentityServer4», которая внутри себя использует фреймворк «Entity Framework Core». В целях разделения ответственности, для работы IdentityServer4 был реализован отдельный контекст базы данных «AuthDbContext.cs», в котором хранятся лишь необходимые для авторизации проекции таблиц базы данных.

В приложении реализована авторизация на основе JWT (Json Web Token). Это стандарт для безопасной передачи информации в формате JSON между двумя сторонами. Он используется для токенов аутентификации и авторизации, которые позволяют пользователю получать доступ к защищенным ресурсам. Использование JWT в качестве авторизации подразумевает указание двух полея — «issuer» и «audience». В поле «issuer» указывается источник, который создал токен. В поле audience указывается список получателей, для которых предназначен токен. Это может быть один получатель или список получателей.

JWT состоит из трех частей: заголовка (header), полезной нагрузки (payload) и подписи (signature). Заголовок содержит информацию о типе токена и используемом алгоритме шифрования. Полезная нагрузка содержит данные, которые должны быть переданы. Подпись создается на основе заголовка, полезной нагрузки и секретного ключа, чтобы обеспечить целостность токена.

Конфигурация того, как будет работать и верифицироваться JWT, находится в настройке промежуточного ПО для аутенфикации, которое предоставляет фреймворк «ASP.NET Core».

1.12 Профиль пользователя

В приложении был реализован функционал, обеспечивающий взаимодействие пользователя с его данными профиля на сайте. Все конечные точки, которые отвечают за это взаимодействие, находятся в контроллере «ProfileController». Этот контроллер защищён атрибутом «[Authorize]», поэтому доступ к нему будет иметь только авторизованный пользователь.

Набор действий с данными пользователя, которые доступны в контроллере «ProfileController» представлены ниже:

— Получение данных пользователя — такие конечные точки, как «GetCurrentUser», «GetUserPassedGames», «GetUserThemeProgress» позволяют полу-

чить данные пользователя о его профиле, пройденных играх, прогрессу по тематике игры соответственно;

— Обновление данных пользователя — такие конечные точки, как «UpdateUserLastName», «UpdateUserPassword» позволяют обновить пользователю его фамилию и пароль соответственно;
— Взаимодействие с маскотом — конечные точки «GetAllCompanions» и «UpdateUserCompanion» позволяют получить всех доступных маскотов и обновить маскот пользователя соотвественно;
— Загрузка аватара — конечная точка «UpdateUserProfilePicture» позволяет пользователю загрузить своё изображение в качестве аватара в своём профиле.

1.13 Система друзей

В разработанном приложении была реализована система друзей — она позволяет пользователям добавлять друг друга в друзья, что делает процесс отслеживания прогресса других пользователей легче и удобнее.

Друзья пользователей хранятся в базе данных в таблице «friends», где реализована связь «Многие-Ко-Многим» — это значит, что у одного пользователя может быть один и более друг. Данная таблица содержит в себе три поля: Id пользователя, Id его друга и статус дружбы. Статус дружбы — самостоятельно написанное перечисление на языке «plpgsql», которое имеет 3 возможных значения: «incoming» (входящий), «outgoing» (исходящий) и «accepted» (принятый).

Все конечные точки API системы друзей находятся в «FriendController» контроллере, который доступен только для авторизованных пользователей. Список доступных конечных точек с их кратким описанием представлен ниже:

— «AddFriend» — HTTP-Post конечная точка, позволяющая одному пользователю отправить запрос на дружбу другому пользователю;
— SearchUsers — HTTP-Get конечная точка, которая предоставляет возможность поиска пользователей по их никнейму;
— GetUserFriends — HTTP-Get конечная точка, возвращающая список друзей пользователя. Бизнес-логика этого метода использует фильтр по статусу дружбы и пагинацию для более удобной выборки данных;
— GetFriendship — HTTP-Get конечная точка, позволяющая узнать статус дружбы между двумя пользователями;

— AcceptFriend — HTTP-Patch конечная точка, с помощью которой один пользователь может подтвердить добавление к себе нового друга. Успешное выполнение этой конечной точки меняет статус дружбы двух пользователей на «accepted»;
— RemoveFriend — HTTP-Delete конечная точка, которая позволяет пользователю удалить из своих друзей другого пользователя. При успешном выполнении данной конечной точки данные о дружбе двух пользователей удаляются из таблицы «friends».

1.14 Работа с играми

Сущность игры является одной из доменных сущностей приложения. Данные об играх хранятся в таблице «games» базы данных, столбцы этой таблицы содержат информацию о названии и описании игры, её сложности и тематике, возрастном ограничении, адресе игры на стороннем VPS сервере. Конечные точки API для получения данных об играх находятся в контроллере «GameController», который содержит как защищённые конечные точки, так и нет.

Набор действий с данными игр, которые доступны в контроллере «ProfileController» представлены ниже:

— Получение данных об играх — такие конечные точки, как «GetAllGames» и «GetGameByLinkName» позволяют получить список всех доступных игр и игру по её ссылке соответственно;
— Обновление результатов игры — HTTP-Put конечная точка «GameRecord» позволяет обновлять результат игры у пользователя;
— Получение сопроводительных данных об играх — контроллеры «ThemeController» и «DifficultyController» содержат необходимые конечные точки для получения всех тематик игр и их сложностей соответственно;
— Получение данных о таблице лидеров — в контроллере «LeaderboardController» находится конечная точка «GetUserGameRecordsByGameLinkName», которая позволяет получить список лидеров по очкам в определённой игре.

1.15 Панель администратора

В разработанном приложении был реализован API для панели администратора. С помощью этого API можно добавлять, редактировать или удалять игры, тематики и сложности игр. Также данный API позволяет добавить нового маско-

та, заблокировать пользователя, добавить новую роль пользователю, а также получить список всех пользователей сайта. Конечные точки этого API находятся в контроллере «AdminPanelController», который защищён фильтром авторизации «[Authorize(Roles = UserRoles.Admin)]» — все конечные точки этого контроллера доступны только пользователям, которые являются администраторами в системе.

ЗАКЛЮЧЕНИЕ

Результатом дипломной работы является функционирующая серверная часть сервиса с образовательными играми для детей. Разработанное серверное приложение обеспечивает удобную работу клиентской части приложения с его API, которое включает в себя функционал регистрации, авторизации, получения и обновления данных пользователя, работы с системой друзей, взаимодействия с данными о результатах игр пользователей, получения и обновления данных об играх, взаимодействия с панелью администратора.

В ходе выполнения дипломной работы были получены и усовершествованы знания об архитектуре, проектировании и разработке серверной части приложений, развёртывании их в среде Docker. В течение создания приложения были усовершествованы умения разработки серверной части веб-приложения с помощью фреймворка ASP.NET Core и вспомогательных технологиях, таких как Entity Framework Core и IdentityServer4. Был приобретён навык интеграции со сторонними сервисами для обеспечения распределённости данных приложения. Также были получены знания и опыт в построении приложений, основанных на различных архитектурах (трёхслойная, монолитная). Цели дипломной работы можно считать достигнутыми.