Проектирование программного обеспечения

Перед тем, как приступать к проектированию программного средства, необходимо провести анализ всех поставленных задач и возможных вариантов его использования. Для этой цели следует разработать диаграмму вариантов использования, которая станет основой дальнейшей работы над приложением. Пример такой диаграммы можно увидеть в приложении Д научной работы.

Разрабатываемое клиент-серверное приложение обладает несколькими преимуществами, такими как доступность через браузер на любом устройстве с интернет-соединением, кроссплатформенность и совместимость. Роль пользовательского интерфейса выполняется клиентской частью, которая отображает информацию и предоставляет пользователю возможность взаимодействия с ней.

Клиентская часть разработана с использованием библиотеки React. Эта библиотека позволяет одностраничные приложения. Клиентская часть разрабатываемого приложения будет обмениваться данными с сервером незаметно для пользователя: без перезагрузки страницы или временной остановки возможности использования приложения. В качестве библиотеки, на которой будет основана вся остальная разработка клиентской части, была выбрана именно React, потому что это современная, быстро развивающаяся библиотека, имеющая открытый исходный код, поддерживаемая компанией Meta, использующая язык TypeScript, который в свою очередь является надмножеством языка JavaScript. В отличии от других библиотек и фреймворков, в React используется технология Virtual DOM, использование которой позволяет оптимизировать производительность приложения и ускорить работу с интерфейсом, особенно в случае больших и сложных приложений.

Серверная часть представляет собой ASP.NET Core Web API приложение. Web API это фреймворк для создания веб-сервисов, которые могут общаться через HTTP-протокол. Он позволяет создавать RESTful веб-сервисы, которые могут обмениваться данными с клиентскими приложениями или другими серверами программного обеспечения. Серверная часть приложения разделена на три слоя: PL(Presentation Layer), DAL (Data Access Layer), BL (Business layer). Общая идея слоистой архитектуры заключается в том, чтобы разделить приложение на набор слоев, каждый из которых решает определенную задачу и имеет свои собственные обязанности. Это позволяет легче понимать и изменять код, уменьшает связность между разными частями приложения и упрощает тестирование и развертку приложения.

Для хранения информации пользователя используется система управления базами данных MSSQL. Для взаимодействия Web API и базы данных используется фреймворк Entity Framework Core. EF Core - это средство объектно-реляционного отображения (Object-Relational Mapping, ORM), которое позволяет отображать данные из базы данных в реальные объекты программы и обратно. Использование ORM экономит время разработчиков и позволяет им сосредоточиться на бизнес-логике, а не на управлении базой данных.

Проектирование базы данных

Для проектирования базы данных необходимо изначально проанализировать предметную область и сущности в ней. В итоге мы получим модель данных, показывающая нам как взаимодействовать с сущностями и как они взаимодействуют между собой.

Модели базы данных для программного средства представлены на рисунке 2.1.



Рисунок 2.1 – Структурная диаграмма основных таблиц

База данных состоит из основных таблиц и вспомогательных, которые в основном используются для осуществления связи многие-ко-многим. Список всех основных таблиц базы данных:

* account;
* aim;
* aimRecording;
* aspNetUsers;
* category;
* loans;
* operation;
* subcategory.

В базе данных все уникальные значения первичных ключей, которые используются для идентификации записей, представлены и сохранены в виде 128-битных глобальных идентификаторов GUID, которые состоят из тридцати двух шестнадцатеричных цифр, разделенных на группы дефисами. Этот формат идентификаторов обеспечивает максимальную уникальность каждого ключа и предотвращает коллизии, что позволяет точно идентифицировать каждую запись в базе данных. В результате, надежность и точность работы системы баз данных повышаются, что является критически важным для успешной работы любой организации. Структура всех таблиц представлена в приложении А.

Таблица Account хранит информацию о счетах пользователей. Все поля таблицы Account перечислены в таблице 2.1

Таблица 2.1 ­­­– Свойства таблицы Account

|  |  |
| --- | --- |
| Название поля | Описание |
| р |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Проектирование клиентской части

Клиентская часть приложения написана с использованием библиотеки *React*. Для хранения данных на клиентской стороне используется библиотека *Redux.*

*Redux* – это библиотека для управления состоянием приложения в JavaScript и TypeScript. Она позволяет хранить все данные в едином объекте-хранилище (*store*) и изменять их с помощью действий (*actions*) и редюсеров (*reducers*). Это делает управление состоянием приложения более предсказуемым и удобным, а также улучшает производительность при работе с большим объёмом данных. Redux часто используется вместе с React, но может применяться и с другими фреймворками и библиотеками JavaScript.

Redux позволяет отслеживать неконтролируемые изменения данных, хранимых в приложении, поскольку кроме хранения данных, после каждой операции делает снимок данных лежащих в памяти. Такие мутации в *TypeScript* могут быть опасными, так как они нарушают ограничения типизации и могут привести к непредсказуемому поведению в вашей программе. В частности, мутации могут вызвать ошибки времени выполнения или привести к несогласованности данных. Но это не повод отказываться от *TypeScript* в пользу *JavaScript*, поскольку в *JavaScript* вообще нет типизации. Также *Redux* позволяет избежать дополнительных запросов на сторону сервера за одними и теми же данными, которые используются в разных местах приложения.

Redux-toolkit

Webpack

Redux - thunk

Схема взаимодействия thunk – app – reducer;

Проектирование серверной части