# РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ СОВМЕСТНОГО ИСПОЛЬЗОВАНИЯ ЧАСТНОГО АВТОМОБИЛЯ

## 3.1 Разработка структуры информационной системы

Для разработки проекта была выбрана многоуровневая архитектура. Основные достоинства многоуровневой архитектуры Web-приложений:

* разгрузка Web-сервера от выполнения части операций, перенесенных на сер­вер приложений, и уменьшение размера модулей расширения сервера вслед­ствие их освобождения от лишнего кода;
* обеспечение более гибкого межплатформенного управления между Web-сервером и сервером БД;
* упрощение администрирования и настройки параметров сети – при внесе­нии изменений в программное обеспечение или конфигурацию сервера БД не нужно вносить изменения в программное обеспечение Web-сервера.

На конечном этапе проект будет иметь 4 слоя работы с данными:

1. Presentation layer (уровень представления): это тот уровень, с которым непосредственно взаимодействует пользователь. Этот уровень включает компоненты пользовательского интерфейса, механизм получения ввода от пользователя. Применительно к ASP.NET MVC на данном уровне расположены представления и все те компоненты, который составляют пользовательский интерфейс (стили, статичные страницы html, javascript), а также модели представлений, контроллеры, объекты контекста запроса.
2. Business layer (уровень бизнес-логики): содержит набор компонентов, которые отвечают за обработку полученных от уровня представлений данных, реализует всю необходимую логику приложения, все вычисления, взаимодействует с базой данных и передает уровню представления результат обработки.
3. Data Access layer (уровень доступа к данным): хранит модели, описывающие используемые сущности, также здесь размещаются специфичные классы для работы с разными технологиями доступа к данным, например, класс контекста данных Entity Framework. Здесь также хранятся репозитории, через которые уровень бизнес-логики взаимодействует с базой данных.
4. Common layer (уровень помощников): хранит различные перечисления, списки, классы помощники, которые могут использоваться на каждом из слоев.

Схема архитектуры представлена на рисунке 3.1.

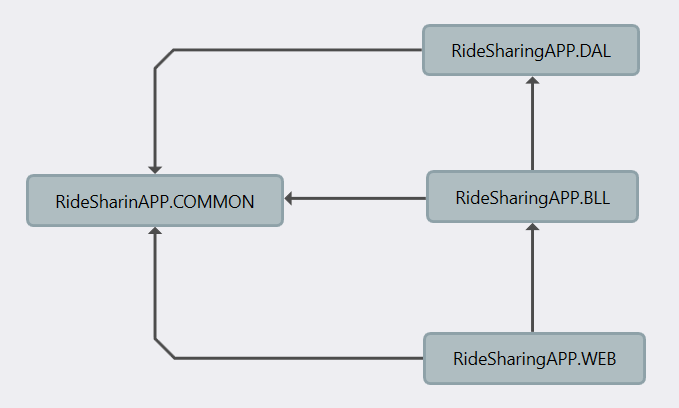


Рисунок 3.1 – Схема архитектуры приложения

Чтобы четко разделять ответственность за различное функционирование в приложении был использован паттерн Model-View-Controller (MVC). Данный паттерн позволяет отделить графический интерфейс от бизнес логики, а бизнес логику от данных.

Таким образом, в классическом варианте, MVC состоит из трех частей, которые и дали ему название:

* модель – этот компонент отвечает за данные, а также определяет структуру приложения (Model);
* представление – этот компонент определяет внешний вид приложения и способы его использования (View);
* контроллер – этот компонент отвечает за связь между моделью и представлением. Определяет, как сайт реагирует на действия пользователя (Controller).

Для облегчения понимания принципов работы приложения, рассмотрим алгоритм работы каждого модуля на примере регистрации в системе.

В качестве модели данного алгоритма использован   
класс – «InputViewModel». Класс представлен на рисунке 3.2.

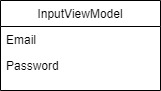


Рисунок 3.2 – Класс модели

В качестве представления используется cshtml страница, на которой отрисована форма авторизации, форма авторизации представлена   
на рисунке 3.3.

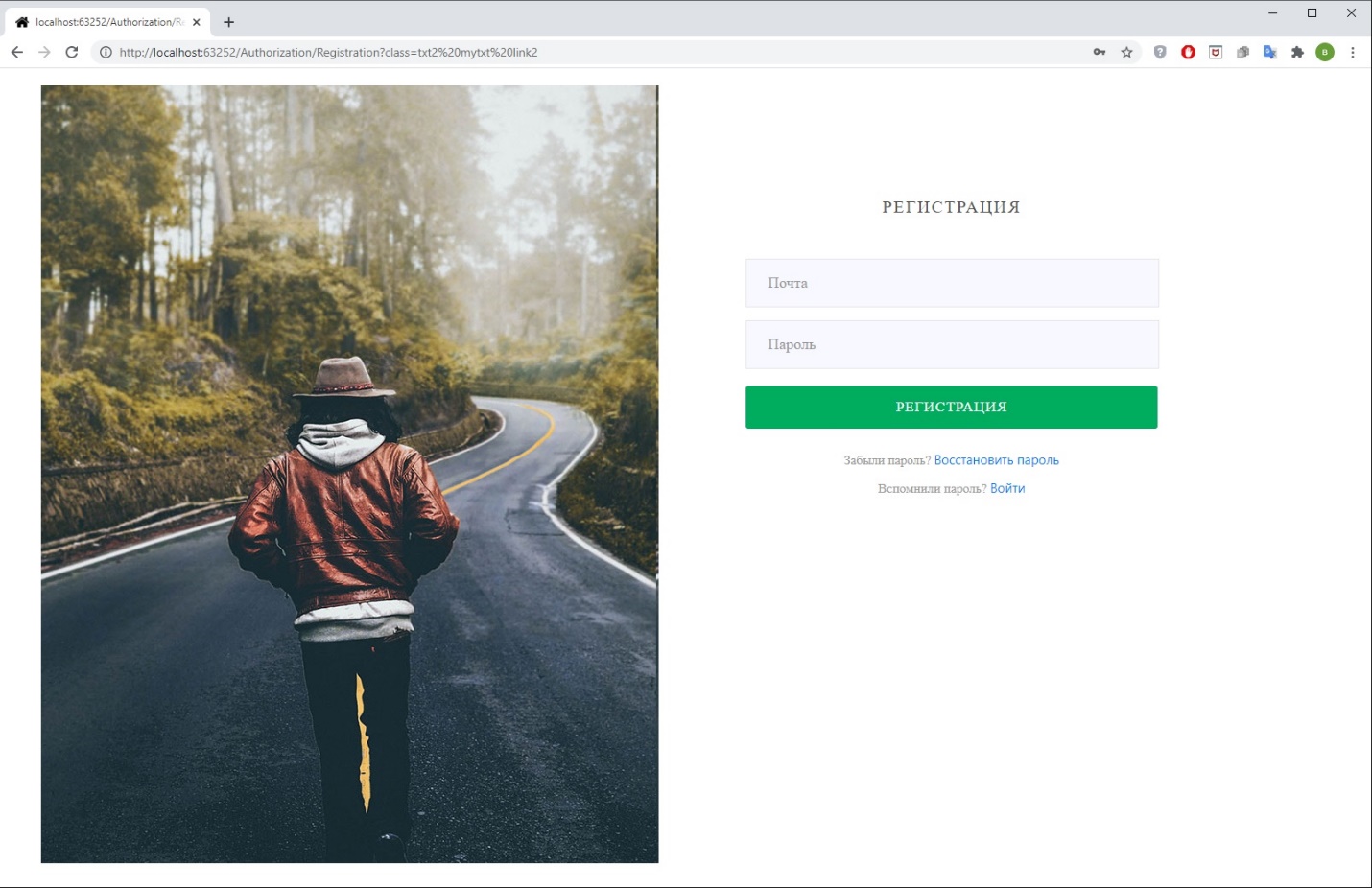


Рисунок 3.3 – Класс модели

В качестве контролера используется класс «AuthorizationController», класс представлен на рисунке 3.4.

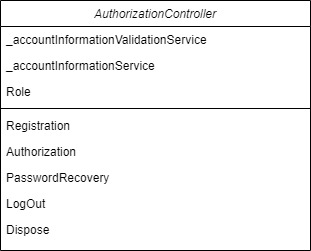


Рисунок 3.4 – Класс контролера

Таким образом, при переходе на страницу, пользователю отрисовывается форма – View, на форме могут быть отображены какие-либо данные модели, в нашем случае модель требуется заполнить, то есть ввести почту и пароль.

После заполнения данных, при нажатии кнопки «Регистрация» сработает метод контроллера – «Registration». Контроллер проверит корректность введенных данных на своем уровне и передаст управление уровню для работы с бизнес логикой, классам «AccountInformationService» и «AccountInformationValidation».

Так как проверка на корректность введенных данных на стороне клиента производилась с помощью JavaScript, который могут не поддерживать некоторые браузеры, в программе реализован второй уровень проверки данных, в классах «Validation» на уровне бизнес логики.

Таким образом класс «AccountInformationValidation» осуществит проверку на корректность введенных данных, на стороне сервера. Класс «AccountInformationValidation» представлен на рисунке 3.4.

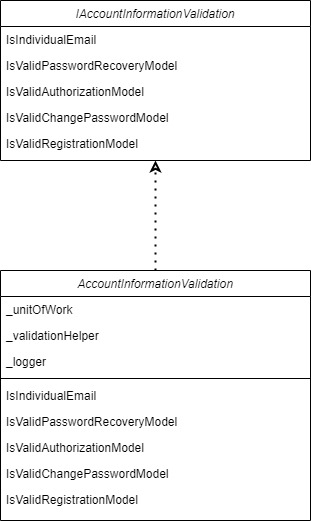


Рисунок 3.4 – Класс «AccountInformationValidation»

Все вычисления, запись, корректировка, сортировка и прочие операции с данными выполняются в сервисах (классах «Service»). В нашем случае сервис «AccountInformationService» для сохранения данных в базе данных обратиться к классу «UnitOfWork» уровня доступа к данным, который имеет прямой доступ к базе данных. Класс «AccountInformationService» представлен на рисунке 3.6.

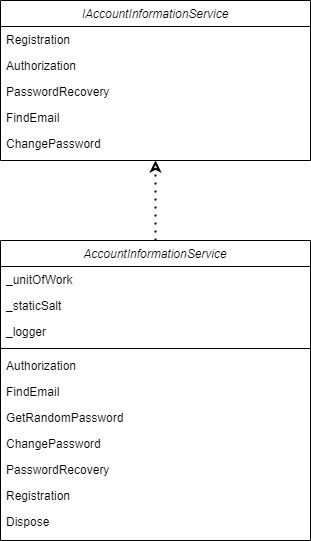


Рисунок 3.6 – Класс «AccountInformationService»

Класс «UnitOfWork» представляет собой паттерн Unit Of Work. Паттерн Unit of Work позволяет упростить работу с различными репозиториями и дает уверенность, что все репозитории будут использовать один и тот же контекст данных, что в свою очередь позволяет без особого труда, в случае надобности, сменить СУБД приложения. Класс «UnitOfWork» представлен на рисунке 3.7.

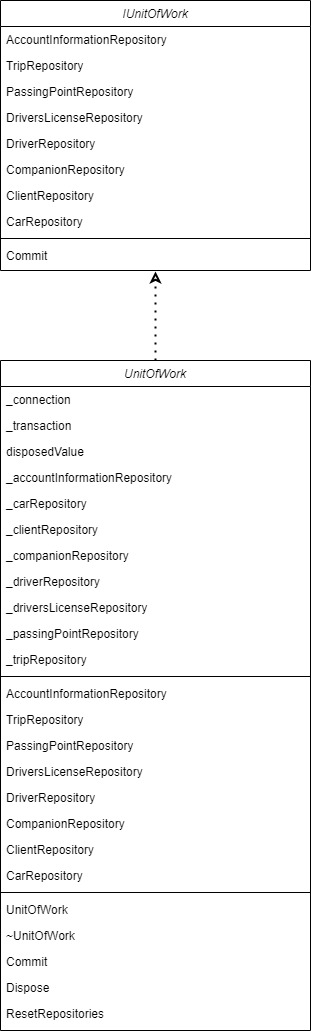


Рисунок 3.7 – Класс «UnitOfWork»

После сохранения данных, уровень доступа к данным возвращает результат обратно на уровень бизнес логики, тот обрабатывает данные и возвращает их на уровень представления, на котором контролер в свою очередь также обрабатывает данные и с помощью представления (View), отображает ошибку на форме, как показано на рисунке 3.8. Если были введены корректные данные – контроллер отобразит пользователю другое представление, авторизации, которое представлено на рисунке 3.9.

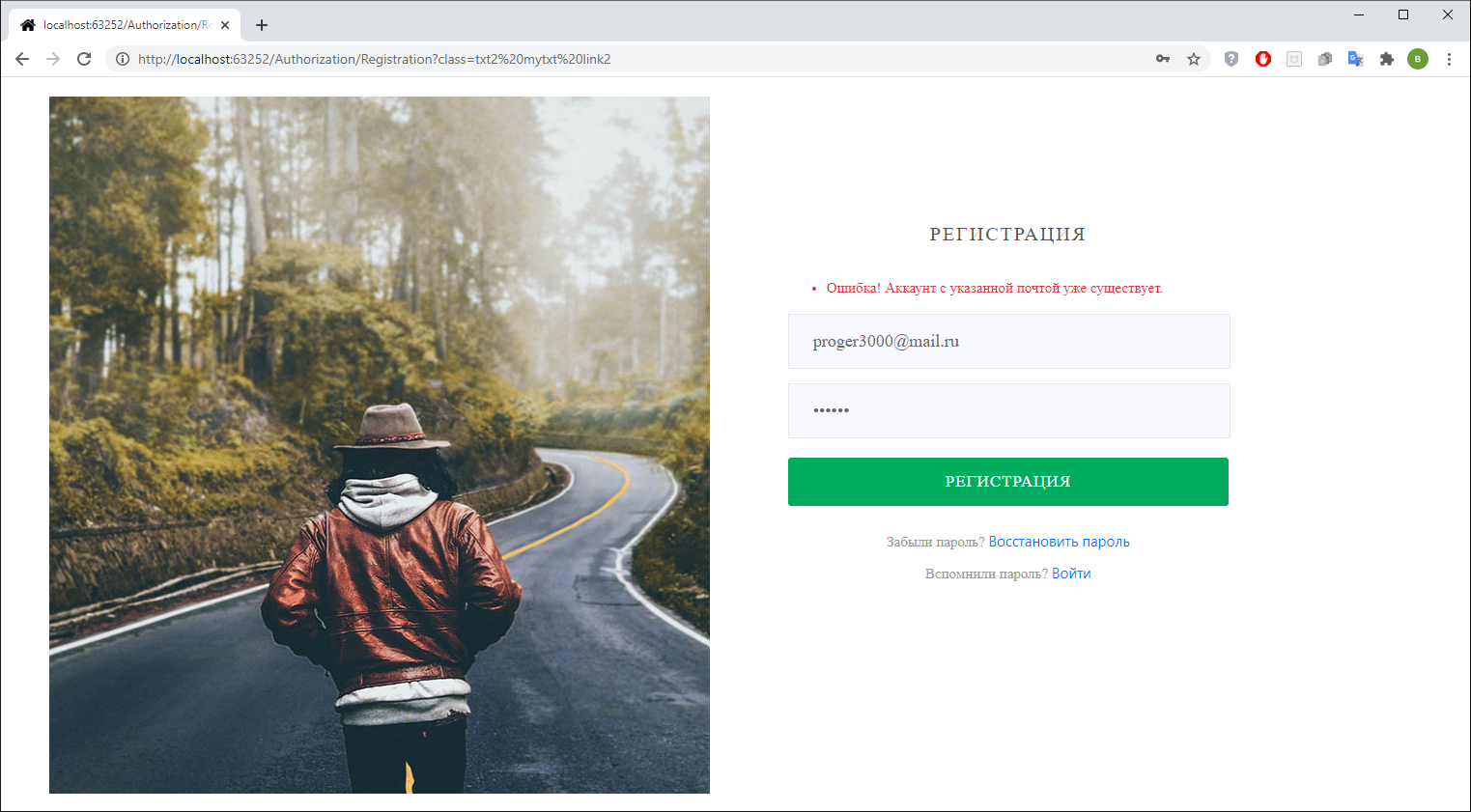


Рисунок 3.8 – Страница регистрации, с ошибкой

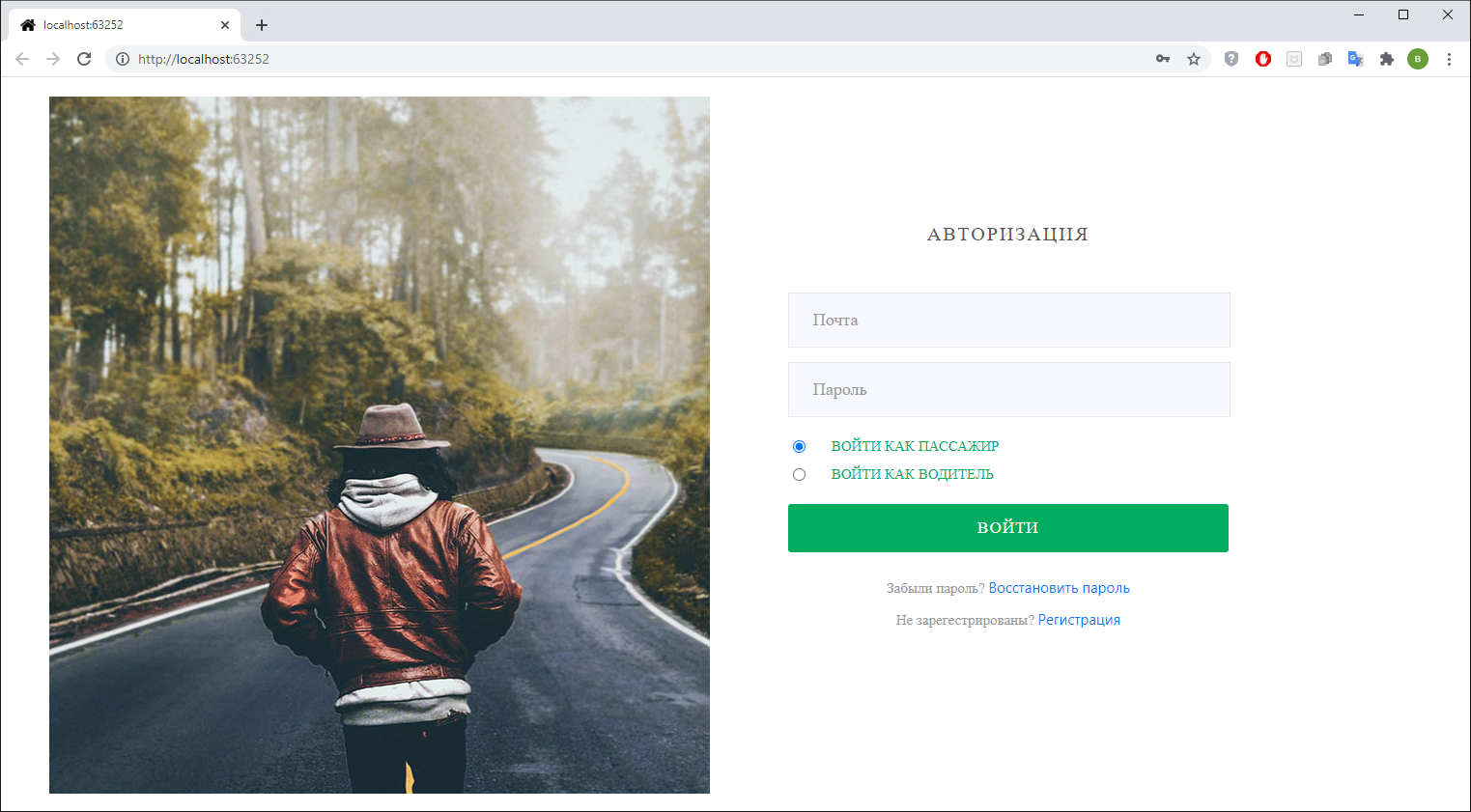


Рисунок 3.9 – Страница авторизации

Структура приложения находится в графической части дипломного проекта. В ней также приведены блок-схемы алгоритма авторизации и создания маршрута.