**Решение представлено в виде двух связанных проектов:**

1. Parrot Wings ASP.NET Core
2. Parrot Wings ASP.NET Framework API

Parrot Wings ASP.NET Core – React приложение, осуществляющее CORS-запросы к серверу API обрабатывающее их.

Parrot Wings ASP.NET Framework API – ASP RESTfull API, настроенный на прием CORS-запросов от сервера сконфигурированного в Parrot Wings ASP.NET Core (разрешение с указанием url прописано в контроллере). Проект сконфигурирован на ожидание запроса извне без показа “главной страницы”.

Parrot Wings ASP.NET Core сконфигурирован как зависимый от Parrot Wings ASP.NET Framework API проект, и при запуске солюшена происходит автоматический старт Parrot Wings ASP.NET Framework API перед началом выполнения Parrot Wings ASP.NET Core.

**Используемый стек технологий:**

Сервер: MS SQL Server, IIS, Web API, Entity framework

Клиент: HTML5, Javascript, React, bootstrap (частично, больше привык к четвертому, но его внедрить в проект быстро не вышло, воевать с VS не стал)

**Структура и особенности Parrot Wings ASP.NET Core**

Проект был настроен на связь с локальной базой данных Parrot Wings с использованием EF, строка подключения врезана в конфиг проекта.

ERD представлена в файле erd.jpg

Бэкап БД представлен в файле бэкапа ParrotWingsBackup.bak и скрипте ParrotWingsScript.sql

При сериализации модели данных в JSON для ответа сервера был риск возникновения петель, в связи с чем ленивая загрузка контекста была отключена.

Про настройки CORS уже говорилось ранее.

В проекте были разработаны модели данных, репозиторий включающий реализацию доступа к данным, контроллер для обработки HTTP запросов.

Проект сконфигурирован на использование HTTPS.

**Структура и особенности Parrot Wings ASP.NET Core**

Проект представляет собой React приложение. Все данные приложение получает c помощью XMLHttpRequest, адрес сервера храниться в отдельном “конфигурационном” файле (AppSettings.js) как константа и конкатенируется с адресами команд в компонентах. Во время ожидания ответа от сервера пользователю демонстрируется “окно” загрузки, чтобы он не гадал “живо” приложение или нет (а также не имел возможности отправить сто запросов во время ожидания). Все поля для ввода имеют валидацию, по возможности использовались средства HTML, когда их ответ был не совсем корректен, или логика проверки была слишком сложной – использовалась кастомная валидация средствами js, с последующим выводом комментария об ошибке в соответствующее поле (например, дублирующийся email). Некоторая информация доводится до пользователя с помощью показа сервисных “окон” (например, провал попытки авторизации).

В запросах трансфера PW между пользователями предусмотрено “фоновое подтверждение” пользователя паролем, для невозможности перевода средств потенциальными “левыми” запросами к серверу.

Код проектов максимально прокомментирован, чтобы лучше понять мою логику реализации.