OTUS

Специальная цена

C# ASP.NET Core разработчик

Курс для опытных C#-разработчиков по созданию сайтов на ASP.NET Core

Длительность курса: 160 академических часов

1 Введение в ASP.NET Core и WebApi

1	**Вводное занятие в курс**	познакомиться с преподавателями курса; определиться с персональными целями; обсудить формат Scrum-команд. Домашние задания 1 Знакомство с курсом Цель: В этом занятии вы сделаете первые шаги к вашей работе над курсовым проектом, а также познакомитесь с командой. Для всех: 1) Если у вас нет опыта в JS, приступить к самостоятельному изучению видеокурса "Javascript-для начинающих разработчиков". Если у вас нет к нему доступа, пожалуйста, напишите запрос на help@otus.ru. Вам нужно освоить его до начала модуля "Frontend-технологии". 2) Зарегистрироваться на Гитлабе и сделать коммит в общий проект, при сдаче ДЗ необходимо приложить ссылку на коммит в gitlab репозитории; [Если присоединились к курсу после первого занятия, то нужно написать в Slack руководителю программы Алексею Ягур (Aleksey Yagur) и он определит вас в подходящую команду.] 3) Написать “О себе” в закрытой группе команды, при сдаче ДЗ также приложить ссылку на сообщение в Slack; 4) Выбрать название для команды; 5) Выбрать тему для командного проекта; 6) Написать три способа траты баллов.

Для SCRUM-мастеров:
7) Создать закрытый чат команды в слаке (добавить в него @Alex Yagur);
8) Создать проект на GitLab (https://about.gitlab.com/) или Azure (https://azure.microsoft.com/ru-ru/services/devops/);
9) Добавить в корень проекта открытую лицензию (https://opensource.org/licenses/MIT);
10) Создать доску задач с колонками "Беклог", "В работе", "Сделано";
11) Наполнить беклог верхнеуровневыми задачами.

Пример домашней работы на проверку:
1) К курсу приступил/Я уже JS-гуру
2) <ссылка на коммит в Гитлабе>
3) <ссылка на сообщение "О себе">
4) Название команды: <Название команды>
5) Название проекта: <Тема для проекта>
6) Баллы предлагаю потратить на X, Y и Z.

Дополнительные пункты для SCRUM-мастеров:
7) Чат создан и Алексей добавлен
8) Общедоступный проект создан
9) <Ссылка на файл лицензии>
10) <Ссылка на доску задач>
11) Верхнеуровневые задачи по проекту на доску задач добавлены.

---

2 **Знакомимся с ASP.NET Core**

объяснить, что такое ASP.NET Core и какую проблему он решает;
рассказать какие функции и модули есть внутри фреймворка (MVC, Model Binding, Request LifeCycle);
использовать шаблон для домашних работ:
Asp.net Core vs Asp.net 5;
Request LifeCycle.

Домашние задания

1 Подготовка инфраструктуры обучения и проекта по шаблону

Цель: На протяжении всего курса мы будем с вами улучшать проект, рассылающий промо-коды. Репозиторий доступен по адресу https://gitlab.com/devgrav/otus.teaching.promocodefactory/-/wikis/Home.
В этом задании вам необходимо будет запустить проект и добавить методы в контроллер EmployeesController.

1) Сделать форк репозитория для домашнего задания №1 в свой личный репозиторий (см. Описание, ссылка на репозиторий https://gitlab.com/devgrav/otus.teaching.promocodefactory.homework.base);
2) Убедиться, что проект запускается. Для этого достаточно проверить работу шаблона, вызвав метод API через Swagger. Проект использует .NET Core 3.1, так что надо убедиться, что установлено SDK https://dotnet.microsoft.com/download/dotnet-core/3.1 и все

собирается и запускается на вашей машине.
3) Реализовать Create/Delete/Update методы в EmployeesController на основе материалов ближайших уроков модуля и дополнительных материалов. Методы должны также использовать репозиторий с данными в памяти, как в базовом примере, при этом состояние списка сотрудников должно сохраняться между разными запросами.

В качестве результата необходимо отправить в чат с преподавателем ссылку на свою домашнюю работу.

3	**Конфигурирование приложения, разработка контроллеров и подключение Swagger-а**	конфигурировать запуск приложения; настраивать маршрутизацию запросов; создавать свои Контроллеры и Действия; использовать Swagger для описания своего API и генерации клиентов.

4	**Стандартный и нестандартные DI контейнеры: что и когда использовать**	объяснить что означает D в аббревиатуре SOLID; использовать стандартный DI-контейнер; подключать нестандартный DI-контейнер.

5	**Работа с базой данных с помощью Entity Framework Core**	настроить работу с реляционной БД через EF; объяснить, что представляет из себя паттерн Репозиторий и паттерн Unit Of Work. Домашние задания 1 Создать базу данных и сконфигурировать EF Цель: Описание домашнего задания можно найти в репозитории https://gitlab.com/devgrav/otus.teaching.promocodefactory.homework.ef Если у вас нет опыта в JS, приступить к самостоятельному изучению видеокурса "Javascript-для начинающих разработчиков". Если у вас нет к нему доступа, пожалуйста, напишите запрос на help@otus.ru. Вам нужно освоить его до начала модуля "Frontend-технологии".

2 Разворачивание приложения в облаке

1	**Способы размещения приложения, Kestrel, IIS**	различать способы размещения приложений для ASP.NET Core; назвать различия, плюсы и минусы использования различных web-серверов для ASP.NET Core и конфигурировать их; использовать Generic Host и Web Host, писать свои Hosted Service для фоновых задач;
2	**Введение в docker, обзор docker compose**	объяснить, что такое Docker и зачем нужна контейнеризация приложений; составить Dockerfile для ASP.NET Core приложения на основе примера; читать и понимать содержимое docker-compose файлов; составлять docker-compose файлы самостоятельно и настраивать конфигурацию ASP.NET Core сервиса по переменным окружения. Домашние задания 1 Сделать docker compose file + Настроить CI для своего проекта Цель: В этом ДЗ вы сделаете docker compose file + настроите CI для своего проекта Описание домашнего задания можно найти в репозитории https://gitlab.com/devgrav/otus.teaching.promocodefactory.homework.dockergitlab
3	**Использование Kubernetes в качестве системы оркестрации контейнеров**	объяснить, что из себя представляет система оркестрации контейнеров; назвать плюсы и минусы Kubernetes по сравнению с остальными.
4	**Введение в CI/CD и настройка на практике**	объяснить, что представляет собой понятие DevOps и зачем нужно; объяснить, что из себя представляет процесс CI/CD; сформулировать основные инструменты для CI и CD, их отличия; настроить собственный CI/CD в GitLab для ASP.NET Core сервиса;
5	**Разворачивание ASP.NET Core приложения в облаке**	выбрать подходящую площадку для разворачивания; разворачивать свои приложения на выбранной площадке.
6	**Ретроспектива и планирование**	уверенно участвовать в ретроспективах; планировать объём работ на будущее.

3 Прокачиваем проект

1	**Конвейеры ПО промежуточного слоя (Middleware)**	объяснить, как работает конвейер запроса в ASP.NET Core; проанализировать в каком порядке вызываются middleware в цикле запрос/ответ; объяснить, как написать свой middleware и в каком случае это может быть нужно; получить примеры кастомных middleware для разных задач и разобрать какой-то стандартный middleware из ASP.NET Core.
2	**Юнит тестирование: фреймворки и инструменты**	изучить основные отличия между популярными фреймворками для тестирования на .NET и посмотреть возможности XUnit; разобраться с шаблонами написания поддерживаемых тестов (AAA, именование, создание тестовых данных через фабричные методы, билдеры и Autofixture); использовать Fluent Assertion для написания удобных утверждений в тестах; разобраться, как тестировать контроллеры и почему тесты на состояние бывают предпочтительнее тестов на поведение, поработать с Moq; написать несколько Unit-тестов на предложенные сценарии на XUnit. Домашние задания 1 Написать тесты к своему проекту и добавить их прогон в CI Цель: Задание поможет отработать навыки написания тестов в ASP.NET Core проектах на xUnit. Новые требования: 1. Для нормальной работы с партнерами решили добавить в систему список партнеров и вести их активность, также будет сохраняться количество промокодов, которые выдал партнер; 2. Приняли решение для части партнеров в ручном порядке продавать подписку на использование API. Для контроля подписки вводим лимиты на выдачу промокодов для партнеров и при попытке выдать промокод будем контролировать лимиты; 3. Лимиты будем задавать и отменять через API. Лимиты должны иметь дату создания, окончания и если отменим до даты окончания, то дату отмены, а также количество промокодов в лимите. В домашнем задании нужно протестировать установку лимитов партнеру. Тесты можно писать в файле SetPartnerPromoCodeLimitAsyncTests в проекте UnitTests. Сделать форк репозитория Homework 4 (https://gitlab.com/devgrav/otus.teaching.promocodefactory.homework.unittests) домашнего задания и реализовать пункты в нем.

		Нужно протестировать следующие Test Cases для установки партнеру (класс Partner) лимита (класс PartnerLimit) метод SetPartnerPromoCodeLimitAsync в PartnersController): 1. Если партнер не найден, то также нужно выдать ошибку 404; 2. Если партнер заблокирован, то есть поле IsActive=false в классе Partner, то также нужно выдать ошибку 400; 3. Если партнеру выставляется лимит, то мы должны обнулить количество промокодов, которые партнер выдал NumberIssuedPromoCodes, если лимит закончился, то количество не обнуляется; 4. При установке лимита нужно отключить предыдущий лимит; 5. Лимит должен быть больше 0; 6. Нужно убедиться, что сохранили новый лимит в базу данных (это нужно проверить Unit-тестом); Если в текущей реализации найдутся ошибки, то их нужно исправить и желательно написать тест, чтобы они больше не повторялись. Тесты должны соответствовать следующим условиям: 1. Использовать именование: ИмяЕдиницыТестирования_Условие_ОжидаемыйРезультат; 2. Для Arrange этапа должен использоваться фабричный метод при определении данных; 3. Для Stub и Mock использовать Moq; 4. Для создания тестируемого класса, например, PartnersController можно использовать AutoFixture (https://habr.com/ru/post/262435/), чтобы избежать ошибок компиляции при добавлении новых зависимостей в конструктор или также использовать фабричный метод; 5. Для проверки утверждения в тестах должен использоваться FluentAssertions; 6. В качестве дополнительного условия тестовые данные должны формироваться с помощью Builder, если не будет использоваться AutoFixture, то Mock и Stub также должны настраиваться через Builder; 7. В качестве дополнительного условия можно провести рефакторинг PartnersController или только метода SetPartnerPromoCodeLimitAsync на свое усмотрение, может быть ввести дополнительные классы для повышения удобства тестирования и сопровождения, если появятся дополнительные классы и методы, то они тоже должны быть протестированы, принцип расположения тестов в проекте соответствующий тестам в примере (Например, WebHost/Controllers/PartnersControllerTests); Если будет сделан рефакторинг, то при сдаче ДЗ поясните какие классы были введены; 8. В качестве дополнительного условия можно добавить прогон тестов CI в GitLab и при сдаче ДЗ прислать ссылку на прогон.
3	**Как устроено логгирование в ASP.NET Core**	проанализировать принцип устройства ILogger и уровни логирования; проанализировать куда можно писать логи в ASP.NET Core и с помощью чего; получить пример текущих Best Practices для логирования,

		Pull/Push модели логирования для средств просмотра логов (ELK, GreyLog); настроить логирование на Serilog.
4	**Различные способы валидации данных**	проанализировать, как работает ModelBinding, как и зачем; реализовать свой код и как связан с сериализацией данных; проанализировать работу со встроенными механизмами валидации модели (атрибутами и устройством model state) и как писать свои атрибуты валидации; написать валидатор через Fluent Validation и чем удобна эта библиотека.
5	**Кеширование: внутреннее и внешнее**	сформулировать причины зачем нужно кеширование и какие данные подходят для кеша, а также получить способы инвалидации кеша; проанализировать как устроено кеширование на уровне HTTP, кеш ответов через UseResponseCaching; разобраться, когда достаточно внутреннего кеширования, а когда нужно внешнее; посмотреть использование IMemoryCache и как масштабировать сервисы с кешем; посмотреть использование IDistributedCache на примере Redis и сравнение с Memcached.
6	**Пишем свой API: REST vs OData**	разобраться с REST-стилем для описания API в ASP.NET Core и уровнями зрелости REST; посмотреть примеры разработки REST API и лучшие практики (методы версионирования, описание через Swagger и т.д.); получить представление и примеры работы с протоколом Odata и его преимущствах по сравнению с REST на ASP.NET Core.
7	**Пишем свой API: GraphQL**	проанализировать почему появился и как может быть полезен; GraphQl для разработки API; разобрать пример реализации GraphQl Api на ASP.NET Core.
8	**Пишем свой API: gRPC, SignalR Core**	провести обзор gRPC протокола и работы с ним из ASP.NET Core на примерах; объяснить взаимодействие в режиме реального времени через SignalR и Websocket. Домашние задания 1 Добавить один из инструментов к своему проекту Цель: Данное задание позволит попробовать технологии альтернативные REST-like подходу для формирования API сервисов. Сделать форк репозитория Homework 5 (https://gitlab.com/devgrav/otus.teaching.promocodefactory/-/wikis/Homework-5) домашнего задания и реализовать

		пункты в нем; Реализуем API аналогичный Customers с помощью 2-х разных подходов из технологий: gRPC, GraphQL или SignalR; Например, GRPC и GraphQL.
9	**Интеграционное, нагрузочное тестирование**	сформулировать подходы к интеграционному тестированию в ASP.NET Core; посмотреть, как можно тестировать ASP.NET Core через WebApplicationFactory и xUnit; сформировать представление о BDD подходе к тестированию и посмотреть пример тестирования через SpecFlow; посмотреть, как можно провести нагрузочное тестирования сервиса на ASP.NET.
10	**Ретроспектива и планирование**	уверенно участвовать в ретроспективах; планировать объём работ на будущее.

4 Микросервисная архитектура

1 **Монолиты и микросервисы: что есть что**

отличать монолит от микросервиса;
объяснить, что и в каких случаях лучше использовать;
перечислить плюсы и минусы различных способов построения систем.

---

2 **Как превратить монолит в микросервисы**

объяснить в каких случаях стоит, а в каких - не стоит разделять систему на более мелкие части;
проводить декомпозицию большой монолитной системы на подсистемы;
выделять микросервисы по функциям и доменам (DDD) системы;
планировать разделение большой системы на подсистемы с помощью шаблонов интеграции приложений.

Домашние задания

1 Выделить часть функционала монолита в микросервис

Цель: Подготовить документ-план распила монолита на микросервисы.

0. Выбрать крупную, известную вам информационную систему, в примере по ссылке https://docs.google.com/spreadsheets/d/1Y-0ExAXsmt-tyQOgjAZv43ReCvMUR5kVcpGN1ppQWGw/edit?usp=sharing представлено Яндекс.Такси. Описать ее назначение и один или несколько ее бизнес процессов
1. Определить необходимые микросервисы, и их назначение (должно быть не менее 5)
2. Описать взаимодействие сервисов в рамках одного из бизнес-сценариев. Т.е. какой микросервис с кем взаимодействует, какие данные передает, какие ожидает и способ взаимодействия (синхронно (какая технология?) или асинхронно (какая технология?)) Например, при запросе от пользователя нам необходимо:
проверить не исключен ли он,
затем построить маршрут
на основе маршрута у тарификатора получить данные о стоимости
и т.д.
3. Использовать паттер strangler для одного из микросервисов

---

3 **Работа с данными в микросервисах: Архитектура**

объяснить различные способы организации согласованности данных в микросервисах;
разделять потоки данных между микросервисами через CQRS;
аргументировать за и против использования синхронных и асинхронных каналов взаимодействия;
использовать MassTransit и RabbitMq для реализации асинхронного обмена данными между микросервисами;
объяснить в каких случаях и как использовать CQRS и Event Sourcing, саги и распределенные транзакции;

4	**Работа с данными в микросервисах: Работа с реляционными БД**	объяснить основную идею реляционных баз данных и необходимость их использования, индексы и уровни изоляции транзакций; использовать ADO.NET для работы с базой данных в приложении; использовать Dapper для упрощения работы с объектами, используемыми при взаимодействии с базой данных.
5	**Работа с данными в микросервисах: Работа с NoSQL**	объяснить в чём сходство и отличие SQL и NoSQL; выделить подходящие данные для хранения в NoSQL базе данных; использовать NoSQL базу данных в своём проекте на примере MongoDb или Cassandra. Домашние задания 1 Добавить NoSQL базу в микросервис Цель: Домашнее задание поможет получить навыки работы с NoSQL базами данных и распределенным кэшем в микросервисном варианте проекта. Подробно про его архитектуру можно почитать здесь: https://gitlab.com/devgrav/otus.teaching.promocodefactory/-/wikis/Home Сделать форк репозитория Homework 6 (https://gitlab.com/devgrav/otus.teaching.promocodefactory.homework.nosql) и реализовать ДЗ в нем. Необходимо подключить NoSql базу данных к одному из микросервисов или распределенный кеш на Redis, на выбор. Mongo В качестве NoSQL базы можно использовать Mongo и перенести туда данные и работу с ними для одного из микросервисов на выбор: модуля Администрирование либо модуля Предложение промокодов клиентам. Mongo образ нужно будет добавить в docker-compose: image: "mongo:4.2.3" Модуль "Администрирование" В модуле Администрирование (Otus.Teaching.Pcf.Administration) нужно изменить хранение сущностей Employee и Role в MongoDb, реализовать новый репозиторий для работы с Mongo, плюс обеспечить работу текущих контроллеров с новым хранилищем. Модуль "Предложение промокодов клиентам" Если хочется лучше познакомиться с Mongo, то можно реализовать более сложный функционал из микросервиса Предложение промокодов клиентам (Otus.Teaching.Pcf.GivingToCustomer). Необходимо перенести хранение сущностей в Mongo вместо Postgress и реализовать функционал, приведенный в контроллерах на новом хранилище. Redis

Если вместо Mongo хочется поработать с Redis, то в Redis кэш можно перенести справочник предпочтений Preference и во всех микросервисах обращаться к этим данным через Redis кэш и интерфейс IDistributedCache.
Так как предпочтения сейчас нужны в двух микросервисах, то при получении промокода от партнера и при выдаче клиентам выглядит правильным реализовать микросервис со справочной информацией, где сохранять предпочтений в СУБД, а для быстрого доступа использовать распределенный кеш или локальный кеш микросервиса.
Тогда сервис справочной информации нужно будет добавить в docker-compose и сделать его сборку в Dockerfile по аналогии с другими сервисами, плюс при получении данных предпочтений в Otus.Teaching.Pcf.GivingToCustomer и Otus.Teaching.Pcf.ReceivingFromPartner вызывать новый микросервис, который будет использовать кеш.
Однако, для простоты реализации кеш можно добавить только в один из текущих микросервисов и вместо базы получать предпочтения из него.

Инструкция по запуску:
Проект состоит из трех микросервисов и их баз данных, настройка Posgress баз для них приведена в docker-compose файле в корне репозитория, чтобы запустить только базы данных выполняем команду:
docker-compose up promocode-factory-administration-db promocode-factory-receiving-from-partner-db promocode-factory-giving-to-customer-db
Сами сервисы доступны в общем solution:
Otus.Teaching.Pcf.sln
Если базы данных запущены, то в Visual Studio или Rider настраиваем запуск нескольких проектов сразу и работаем с API через Swagger, для API в Swagger добавлены примеры данных для вызова и тестирования.

---

6 **Обзор популярных брокеров сообщений и работа с RabbitMQ**

рассказать в чём отличия и сходства популярных брокеров сообщений;
создавать собственные облачные сервера с настроенным для работы RabbitMQ;
писать приложения, использующие облачный сервер RabbitMQ для общения друг с другом.

Домашние задания

1 Подключить RabbitMQ к своему проекту

Цель: Домашнее задание поможет изучить на практике декомпозицию микросервисов и организацию взаимодействия между ними через RabbitMq.

Сделать форк репозитория Homework 7 (https://gitlab.com/devgrav/otus.teaching.promocodefactory.homework.messagebroker) и реализовать ДЗ в нем.

1. Добавляем RabbitMq в docker-compose файл.
2. Подключаем RabbitMq в микросервисы Выдача промокода клиенту (Otus.Teaching.Pcf.GivingToCustomer), Администрирование (Otus.Teaching.Pcf.Administration) и по

		желанию в Получение промокода от партнера (Otus.Teaching.Pcf.ReceivingFromPartner). Можно использовать драйвер RabbitMq для .NET, тогда вероятно нужно будет использовать Hosted Service для прослушивания очереди, также можно использовать MassTransit или NServiceBus. 3. При получении промокода от партнера в микросервисе Otus.Teaching.Pcf.ReceivingFromPartner в методе ReceivePromoCodeFromPartnerWithPreferenceAsync контроллера PartnersController в конце выполнения сейчас происходят вызовы _givingPromoCodeToCustomerGateway и _administrationGateway, где через HTTPClient изменяются данных в других микросервисах, такой способ является синхронным и при росте нагрузке или отказе одного из сервисов приведет к отказу всей операции. 4. Для повышения надежности вместо синхронных вызовов нужно отправлять одно событие в RabbitMq, тогда в микросервисах Otus.Teaching.Pcf.Administration и по желанию в Otus.Teaching.Pcf.GivingToCustomer нужно сделать подписку на данное событие и реализовать аналог текущих синхронных операций, вызываемых через API. 5. Для того, чтобы ту же логику можно было вызвать из консюмера очереди ее надо перенести из контроллера в класс-сервис, который нужно разместить в проекте Core, в контроллере и консюмере очереди надо вызывать метод этого сервиса. Инструкция по запуску: Проект состоит из трех микросервисов и их баз данных, настройка Posgress баз для них приведена в docker-compose файле в корне репозитория, чтобы запустить только базы данных выполняем команду: docker-compose up promocode-factory-administration-db promocode-factory-receiving-from-partner-db promocode-factory-giving-to-customer-db Сами сервисы доступны в общем solution: Otus.Teaching.Pcf.sln Если базы данных запущены, то в Visual Studio или Rider настраиваем запуск нескольких проектов сразу и работаем с API через Swagger, для API в Swagger добавлены примеры данных для вызова и тестирования.
7	**Как создать хороший шаблон для микросервисов**	снизить дублирование кода в микросервисах; объяснить плюсы и минусы использования общего и раздельного кода в микросервисах для одних и тех же функций; перечислить желательные свойства и характеристики "правильного" шаблона для микросервисов.
8	**Ретроспектива и планирование**	уверенно участвовать в ретроспективах; планировать объём работ на будущее.

5 Frontend-технологии

1 **Базовые элементы фронтенда: Razor + jQuery + MVC**

создавать веб-страницы с использованием Razor;
подключать и использовать jQuery на этих страницах;
объяснить как на практике работает схема разделения данных MVC.

Домашние задания

1 Добавить UI на Razor к проекту

Цель: В этом домашнем задании вы научитесь использовать Razor-страницы для создания веб-интерфейса к вашему приложению.

Добавить админку используя Razor page. Админка должнав позволять редактировать добавлять удалять и просматривать:
1. Предпочтения
2. Промокоды
3. Пользователей
4. Дополнительные таблицы по желанию

---

2 **Современный фронтенд**

различать современный JS синтаксис (JS, ES5/ES6/ES8);
использовать CSS-preprocessors для стилизации;
создавать JS проект с NPM;
получать данные с AJAX в браузере;
расширять webpack config.

---

3 **Введение в React**

провести обзор современных JS-фреймворков;
развернуть react приложение;
сделать eject;
написать разметку компонента на JSX;
добавить валидацию props.

Домашние задания

1 Базовый сетап фронтенд части проекта с React

Цель: Научиться разворачивать фронтенд проекты с npm, webpack, React;
Разобраться с настройками webpack;
Подключить axios.

1. Создать проект с create-react-app;
2. Расширить конфиги webpack добавив проксирование вызова АПИ;
3. Написать компонент с формой авторизации отправляющий запрос.

---

4 **React**

добавить stat к компоненту;
создать обмен данными parent-child components;
рассмотреть очередность и условия вызова lifecycle hooks;
добавить многостраничность с React Router;

добавить работу с стейтом через Redux.

---

5 **React Router & Redux**

добавить многостраничность с React Router;
добавить работу с стейтом через Redux.

Домашние задания

1 Роутинг и управление стейтом с React

Цель: Научиться писать сложный фронтенд с роутингом и управлением стейтом
Разобраться с подключением сторонних плагинов и UI компонентов

1. Установить react-router;
2. Добавить отдельные компоненты страниц - Login / Register / HomePage / 404;
3. Добавить стейт-менеджмент с Redux;
4. Найти возможное дублирование кода и применить HOC паттерн.

---

6 **Как подружить React.JS с Asp.net Core**

объяснить в каких случаях и как разделять фронтовое и бекенд приложения;
определить плюсы и минусы подобного разделения, а также влияния этого на архитектуру приложения в целом и появляющихся особенностей разворачивания и поддержки.

Домашние задания

1 Создать проект для React.js и связать с проектом по API

Цель: Развернуть ASP.NET Core и React на одном и отдельном хосте.

Развёртывание на одном хосте:
1. Создайте новый проект из шаблона ASP.NET Core Web Application with React.js.
2. Убедитесь, что в файле Startup.cs добавлено SPA-middleware.
3. Запустите ваш проект и убедитесь что фронтенд и бекенд работают правильно.
4. Также можете запустить сборку фронтенда отдельно от процесса сборки бекенда. Для этого используйте метод UseProxyToSpaDevelopmentServer().

Развёртывание на разных хостах:
1. Создайте новый бекенд проект из шаблона ASP.NET Core Web Application with API.
2. Создайте новый фронтенд проект с помощью create-react-app.
3. На бекенде настройте CORS для адреса вашего фронтенд приложения.
4. На фронтенде создайте страницу с отображением погоды. Данные о погоде получайте с вашего бекенд приложения.

7	**Ретроспектива и планирование**	уверенно участвовать в ретроспективах; планировать объём работ на будущее.

6 Управление безопасностью

1	**Модель авторизации / аутентификации в Asp.net Core**	разобраться в том, как устроена аутентификация/ авторизация в ASP.NET Core и чем они отличаются; разобраться в структуре сущностей Identity, Claims, Role; посмотреть какие есть способы передачи данных аутентификации / авторизации, токены (jwt), cookies, windows auth; проанализировать варианты реализации авторизации в микросервисах.
2	**Разграничение прав доступа встроенными средствами**	объяснить, как устроена авторизация на основе Roles, Claims, Policies; рассказать, как сделать императивную авторизацию; реализовать ограничение доступа к данным, чтобы пользователи видели только доступные для них данные.;
3	**Identity Server 4: Обзор и устройство**	посмотреть, что из себя представляет Identity Server и какую проблему решает; разобраться в протоколе OAuth, JWE, JWS, JWT.
4	**Identity Server 4: Вход через внешних провайдеров**	посмотреть примеры реализации логина через внешних провайдеров, Google, Facebook и т.д. на Identity Server. Домашние задания 1 Создать собственный сервис для авторизации + логин через Гугл Цель: В этом домашнем задании вы научитесь создавать собственный сервер авторизации и подключать к нему внешние провайдеры аутентификации. 1. Создать проект из шаблона IdentityServer4 с уже подключенным хранилищем пользователей и пользователями (With In-MemoryStores and Test Users); 2. Создать Web-Api приложение с ресурсом (хотя бы возвращающим "Hello World!"), защищённым [Authorize] и подключить к IdentityServer-y; 3. Создать консольное приложение, для доступа к этому ресурсу с помощью jwt-токена полученного от IdentityServer-a; 4. Вывести на экран содержимое ответа запроса токена и ответа Web-Api; 5. Создать приложение в Google и настроить аутентификацию через него; 6. Залогиниться через Google и сделать скриншот страницы клэймов; 7. Подключить к проекту аутентификацию через Facebook; 8. Залогиниться через Facebook и сделать скриншот страницы клэймов.
5	**Как защититься от**	провести обзор основных механизмов атаки на API и сайты;

	хакерских атак	получить способы борьбы с ними на ASP.NET Core и .NET.
6	**Рефакторинг старого кода**	разобраться с тем, что из себя представляет правильный рефакторинг; узнать основные шаги по устранению технического долга;

7 Проектная работа

1	**Консультация по проектам и домашним заданиям**	получить ответы на вопросы по проекту, ДЗ и по курсу. Домашние задания 1 Итоговый проект Цель: В этом домашнем задании вы продемонстрируете знания и навыки, которые приобрели на курсе. Шаги и их состав зависят от конкретной темы проектной работы.
2	**Защита проектных работ**	защитить проект и получить рекомендации экспертов.