WebLab#1
HTTP, REST, NGINX
2022

1. Взять за основу большое приложение с базой данных (например, ЛР по ППО, курсовой проект по БД, личный проект и т.д.). Рекомендуется брать уже существующий проект, чтобы не тратить время на разработку бизнес-логики и логики взаимодействия с БД.

   Можно взять любую новую идею, которая предполагает работу с 3-5 сущностями в базе данных (или ином хранилище, в т.ч. удаленном), и 2-3 формы взаимодействия с пользователем.

   Если идея больше, то допускается формирование групп в 2 человека (с кратным увеличением сложности и объема задачи).

   С этим приложением будут связаны еще 2 работы: макетирование и реализация SPA.

   Если на курсовой у вас было десктопное приложение - так даже интереснее - особенно если удастся сохранить поддержку старого UI.
2. В .md файле подготовить, кратко:
   a. Цель работы, решаемая проблема/предоставляемая возможность.
   b. Краткий перечень функциональных требований.
   c. Use-case диаграмма системы.
   d. Экраны будущего приложения на уровне черновых эскизов. Задача данного упражнения - понять, как с приложением должен взаимодействовать пользовать для упрощения проектирования API. Это могут быть классические wireframes, черновики от руки, наброски в PAINT/псевдографике/Figma.

      Примечание: здесь не требуется UI UX дизайн, требуется примерное распределение функциональности по экранам и возможным компонентам управления.
   e. ER-диаграмма сущностей системы.

3. Спроектировать в формате Swagger (https://editor.swagger.io/) внешнее публичное API системы в идеологии REST, дающее доступ ко всем данным и функциям системы, необходимое для внешних интеграций и последующего подключения клиентского SPA-приложения.

   Если система предполагает двунаправленное или реал-тайм взаимодействие допускается вынести часть API из REST (и реализовать его на базе web sockets/grpc/soap). Вынесенное api документировать в отдельном .md файле.

   Предусмотреть как минимум один вызов на базе метода PATCH.

   Примечание 1: сделать сразу по REST, чтобы потом не переделывать много раз. Не как вам кажется должно выглядеть REST API, а так, как оно действительно должно выглядеть (можно посмотреть тут - https://restfulapi.net/)

   Примечание 2: Во время проектирования API внимательно изучить методы HTTP и коды ответов, а также основные заголовки. Эта информация пригодится при сдаче лабораторных.
4. По спроектированному swagger подготовить реализацию в программном коде. К реализации так же подключить swagger, уже для документирования (возможны вариации в зависимости от используемой технологии). Придерживаться подходов “чистой архитектуры”: поддерживать абстрагирование СУБД путем использования паттерна Repository, использовать три модели сущности: сущность БД, сущность системы и DTO для API. Если в проекте уже есть UI, то поддерживать его в рабочем состоянии. Если уже есть api, то оставить его в версии api/v1 и создать новое api/v2.

   Примечание: к приложению предъявляются все те же требования, что предъявлялись к приложениям в рамках курса ППО.
5. Настроить Nginx для работы web-приложения в части маршрутизации (Гайд для начинающих):
   a. Настроить маршрутизацию /api/v1 (/api/v2) на подготовленное REST API
   b. По пути /api/v1 (/api/v2) отдавать swagger
   c. Если у системы был старый MPA или SPA интерфейс - проксировать его на /legacy. Если MPA-интерфейса не было, то

сделать страничку с ссылкой на загрузку десктопной/консольной версии.

d. Настроить / на отдачу статики (в будущем - SPA-приложения). Пока положить приветственный HTML (/static/index.html) с картинкой (static/img/image.jpg).
e. Настроить /test на отдачу той же страницы, что и /
f. Настроить /admin на проксирование в админку базы данных (любую стандартную).
g. Настроить /status на отдачу страницы статуса сервера Nginx (nginx status)

6. Настроить Nginx в части балансировки (Гайд по настройке балансировки): запустить еще 2 инстанса бэкенда на других портах с правами доступа в базу данных только на чтение и настроить балансировку GET запросов к /api/v1 (/api/v2) в NGINX на 3 бэкенда в соотношении 2:1:1, где первый - основной бэкенд-сервер.
   Провести нагрузочное тестирование с помощью ApacheBenchmark, результаты оформить в виде отчета в .md.
   Быть готовым показать и доказать, что балансировка действительно работает.
7. Настроить Nginx в части маршрутизации, таким образом, чтобы url /mirror1 вел на отдельно развернутую версию приложения, и при этом, все относительные урлы приложения корректно работали с новым префиксом (/mirror1/api/v1 и др.).
8. Настроить Nginx таким образом, чтобы подменялось имя сервера в заголовках http-ответов (проставлялось название приложения).
9. Настроить кеширование (для всех GET-запросов, кроме /api) и gzip-сжатие в Nginx (Настройка gzip сжатия, Гайд по настройке кеширования).

Дополнительное задание #1

Реализовать простейшую jwt-авторизацию. Обращаться к методам api должен иметь возможность только авторизованный пользователь.

Дополнительное задание #2

Настроить https (Создание сертификата) и http2 для всех запросов, продемонстрировать работу ServerPush или механизм Preload на странице index.html и картинке.

Дополнительное задание #3

Настроить работу сервиса по протоколу http3.

Примечание

Рекомендации на случай выбора новой темы:

Требования к приложению.

1. Обязательно использование классического WEB-стека: http, html, css, ts. При этом Backend может быть реализован на C#, Java, Kotlin, Scala, C++, TypeScript, Delphi, Go, Rust.
2. Помимо классического web-приложения, может быть рассмотрено web-приложение, упаковываемое в десктопное и мобильное, а так же telegram bot с WEB UI.
3. Не менее 2-3 экранов с данными. Под “Экраном” понимается не просто html ”об авторе”, а целостная страница с данными. Пример такого приложения: Интернет магазин (страница списка товаров, детальный просмотр товара, корзина), платформа, маркетплейс, умный дом и тд.
4. На каждом экране должна быть минимум одна пользовательская активность (кроме пассивного просмотра информации). Продолжая пример с интернет-магазином: Поиск товара и добавление в корзину на списке товаров, Добавление в корзину при детальном просмотре, Покупка в корзине.
5. В проекте должны быть данные. Это могут быть данные, хранящиеся в виде файлов в файловом хранилище, может быть SQL/NoSQL база данных, может быть сервис данных, а может быть и внешняя система, с которой ваш проект взаимодействует по некому API (например, vk, twitter и т.д.). В любом случае в приложении выделяется слой по работе с такими данными.

Список тем.

Замечание:Ниже приведен небольшой список тем приложений для выбора. Выбрать можно любую тему из списка приложенных, либо свою собственную. В последнем случае ее необходимо предварительно согласовать с преподавателем.

1. Внутренний кафедральный сервис распределения по учебным проектам. Самым разным: как по групповым курсовым, так и оптимальному поиску научного руководителя (по тегам интересов и введенным ограничениям, спискам тем работ - как у преподавателей, так и у студентов).
2. Внутренний кафедральный сервис с ботвой и полезными материалами (наподобие iu7-world.ru).
3. Внутренний сервис управления кафедральным GitLab. По сути - Web- GUI для заведения студентов, подключения их к учебным группам, выдаче репозиториев, просмотра статистики по сдачам лаб (принятым реквестам). Возможна и выдача лаб автоматическим созданием веток.
4. Внутренний сервис уведомлений, объявлений, расписаний и новостей. Информация о присутствию преподавателей, проведению доп занятий и консультаций, пересдач, собраний, переносов и тд.
5. Небольшая кооперативная игра на JS.
6. Маркетплейс.
7. Платформа для умного дома.
8. Технологический блог.