

# WebLab#1

## HTTP, REST, NGINX

### 2021

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 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. 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.

Предусмотреть как минимум один вызов на базе метода PATCH.

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.
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 МРА-интерфейс - проксировать его на /legacy. Если МРА-интерфейса не было, проксировать стартовую страницу web-приложения или заскаффолдить одну из сущностей системы.
  - 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.
7. Настроить Nginx таким образом, чтобы подменялось имя сервера в заголовках http-ответов (проставлялось название приложения).

8. 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 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, js/ts. При этом Web-сервер и Backend могут быть реализованы на любых технологиях. Наиболее предпочтительно использование компилируемых языков программирования со статической типизацией. Хорошим выбором будет экосистемы языков C# и Java.
2. 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.
3. 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.
4. 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 (возрождение великого 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.