LAB1

Цель работы:

- [ ] Выбор варианта-темы на весь курс
- [ ] Знакомство с разработкой бэкенда
- [ ] Разработка дизайна для 3 страниц

Задание:

- [ ] Создание дизайна приложения в `figma`
- [ ] Базовая шаблонизация в Django для `услуг`
- [ ] Развертывание `Minio`

Контрольные вопросы:

- [ ] MVT
- [ ] Компоненты MVC в лабораторной
- [ ] Django
- [ ] Шаблонизация
- [ ] HTTP и модель OSI
- [ ] Web
- [ ] HTML

Подробно:

Требуется разработать дизайн трех страниц приложения в `figma`, повторив при этом основную стилистику уже существующего популярного ресурса (3 цвета с точными кодами, hoover, форму карточек), связанного с вашей темой. Данные по теме также нужно искать реальные.

Создание базового интерфейса, состоящего из трех страниц. Первая для просмотра списка `услуг` (отели, товары, рейсы и тд) в виде карточек (плиткой в 2-4 столбца) с наименованием, ценой и картинкой, а также отдельным элементом-карточкой текущей `заявки` (корзина) с указанием количества `услуг` в ней. Это количество вычисляется в контроллере-обработчике. При клике по карточке происходит переход на вторую страницу с подробной информацией об `услуге` (даты, описание и тд). А при нажатии на текущую `заявку` по ее id открывается страница просмотра состава `заявки`, в которой есть поля самой `заявки`, каждая ее `услуга` отображается ниже отдельной карточкой друг под другом, поля `услуги` вывести слева направо в 1 строку и справа для каждой из них поле м-м (количества/порядка/главный/другое). В `header` приложения добавить только кнопку/иконку `Домой`, она для навигации на страницу `услуг`.

Добавить поле input для фильтрации на сервере списка `услуг` по одному из полей (наименование, цена, дата события), отображаемых на странице (по умолчанию отображать все). Поле поиска должно сохраняться после запроса. Всего в приложении должно быть 3 GET запроса и две модели-коллекции. Все данные для обеих страниц нужно брать прямо из двух коллекций (массива услуг и словаря заявки), без использования БД. Никаких массивов внутри отдельной `услуги` (соблюдать атомарность и 1НФ), но можно и нужно писать большие текстовые описания. Никакого редактирования, только поиск и просмотр. Без `JavaScript`

В приложении должны быть использованы стили, для каждого элемента списка подгружается свое изображение. Изображения `услуг` отображатся на всех трех страницах. Изображения хранятся в `Minio`, наименование изображение совпадает с id `услуги`, но хранится в модели отдельным полем. Разработать стиль приложения, который будет применяться далее в последующих лабораторных по фронтенду. `CSS` вынести в отдельный файл.

LAB2

Цель работы:

- [ ] Разработка структуры базы данных
- [ ] Ее подключение к бэкенду

Задание:

- [ ] Создание базы данных `PostgreSQL` по теме работы

- [ ] Подключение БД к созданным трем шаблонам

Диаграмма:

- [ ] **ER диаграмма**: сделать в StarUML; таблицы, связи, столбцы, типы столбцов и их длина, первичные, внешние ключи

Контрольные вопросы:

- [ ] Виды БД
- [ ] SQL запросы
- [ ] Курсоры
- [ ] ORM
- [ ] Модель и миграции
- [ ] Чистая архитектура

Подробно:

Необходимо разработать структуру БД по выбранной теме и ее реализовать с учетом требований ниже. Использовать таблицу `услуг` в страницах разработанного приложения. Наполнить таблицы БД данными через `админку Django` или `Adminer`.

Получение и поиск `услуг`, добавление `услуги` в текущую `заявку` (статус черновик она же "корзина"), а также просмотр этой `заявки` сделать через `ORM`. Для страницы текущей `заявки` добавить кнопку логического удаления `заявки` (статус меняется на удален) с помощью выполнения SQL запроса `UPDATE`, без ORM. Всего пять HTTP методов: три GET и один POST добавления в заявку через `ORM`, еще один POST для удаления заявки через SQL.

При отсутствии `заявки` в статусе черновик у пользователя, она создается только при добавлении `услуги` в `заявку`. Если такая `заявка` уже есть, то `услуга` добавляется сразу в нее. Удаленные заявки просматривать нельзя, если у пользователя нет текущей `заявки`, ее карточка на странице услуг не активна.

Обязательно наличие 4 таблиц (каскадное удаление запрещено!):

- `услуг` (наименование, описание, статус удален/действует, url к изображению `Nullable`, поля по предметной области)
- `заявок` (только 4 поля `NotNull`: id, статус, дата создания datetime, `создатель`; дата формирования (2 действия `создателя`) datetime, дата завершения (2 действия `модератора`) datetime и `модератор`, поля по предметной области).
- `м-м заявки-услуги` (составной уникальный ключ, дополнительные поля количества/порядка/главный/другое)
- `пользователей` (для `Django` обязательно использовать системную таблицу, НЕ свою: логин, пароль, признак модератора)

Обязательно наличие 5 или более статусов `заявок`: черновик, удалён, сформирован, завершён, отклонён. У каждого пользователя не более одной заявки в статусе `черновик`. Названия таблиц и их полей должны соответствовать предметной области. Одно из доп. полей `заявки` или `м-м` рассчитывается при завершении заявки.

LAB3

Цель работы:

- [ ] Создание веб-сервиса в бэкенде нашей системы для использования его в `SPA`

Задание:

- [ ] Создание веб-сервиса со всей итоговой бизнес логикой (кроме авторизации)
- [ ] Подключение его к БД и тестирование в `postman`

Диаграмма:

- [ ] **Диаграмма классов** с детализацией бэкенда (домены методов по `url` с интерфейсами, перечислить все методы, модели, таблицы БД) + insomnia/postman. Связи у моделей сделать по коду, 3 варианта: методы используют разные модели, модели используют другие модели, модели используют несколько таблиц

Контрольные вопросы:

- [ ] Веб-сервис
- [ ] REST
- [ ] RPC
- [ ] Заголовки и методы HTTP

- [ ] Версии HTTP
- [ ] HTTPS
- [ ] OSI ISO

Подробно:

Создание **веб-сервиса** для получения/редактирования данных из вашей БД. Требуется разработать все методы для реализации итоговой бизнес логики вашего приложения. Для изображений `услуг` использовать `Minio`.

Требования к веб-сервису:
Методы и `url` в `API` должны соответствовать `REST`. Для списка `услуг` (как в 1 лабораторной) и `заявок` (по статусу и диапазону даты формирования) нужно предусмотреть фильтрацию на бэкенде. Взаимодействие с БД через `ORM`. Не делать `POST` `заявки`. Записи в статусе `удален` на клиент не передаются.

Статусы нельзя менять с любого на любой: `создатель` удаляет и формирует черновик заявки, а `модератор` отклоняет и завершает сформированную заявку. В данной лабораторной пользователь `создатель` зафиксирован во всех методах - укажите его константой через функцию- `singleton`. Системные поля (ид, статусы, создатель и модератор, даты создания, формирования и завершения) передавать с клиента для изменения запрещено. Они вычисляются на бэкенде через авторизацию и бизнес-логику.

Домены:

- [ ] Домен услуги
- [ ] Домен заявки
- [ ] Домен м-м
- [ ] Домен пользователь

LAB4

Цель работы:

- [ ] Завершение бэкенда для `SPA`

Задание:

- [ ] Добавление авторизации
- [ ] `Swagger` в веб-сервис
- [ ] Подготовка ТЗ

Диаграмма:

- [ ] **Sequence диаграмма**: весь набор `HTTP` запросов по бизнес-процессу без БД и нативного приложения: аутентификация, список услуг без черновика, добавление услуги в заявку, еще раз список услуг с черновиком, просмотр черновой заявки, редактирование заявки, формирование заявки, список заявок, завершение модератором из второго фронтенда, список заявок с расчетом. Добавить домены в качестве `Lifeline`, при добавлении сообщений выбирать методы доменов из диаграммы классов, передавать ключевые входные и выходные данные через `arguments` в скобках у `Message`

Контрольные вопросы:

- [ ] Куки
- [ ] Сессия
- [ ] Redis
- [ ] JWT
- [ ] Авторизация и аутентификация
- [ ] SSO
- [ ] Двухфакторная аутентификаци
- [ ] RSA

Подробно:

Реализовать методы бэкенда для `аутентификации` и `регистрации`. Авторизация через хранение сессий и куки. Автозаполнение пользователя в таблице `заявок` при создании новой. Добавить описание методов для `swagger`.

Добавить проверку `Permissions` для методов `модератора`. Без авторизации в `Swagger` должно быть доступно только чтение-получение данных через API, с авторизацией - методы `пользователя`, а для `модератора` доступны все методы.

Вместе с 4 лабораторной сдается текущий комплект документации, который требуется оформить в виде `.docx`

LAB5

Цель работы:

- [ ] Разработка базового SPA на React

Задание:

- [ ] Разработать три страницы фронтенд приложения на `React`, `TS`
- [ ] Подключить его к веб-сервису

Диаграмма:

- [ ] **Диаграмма классов:** с детализацией бэкенда и фронтенда: добавить методы авторизации, фронтенд разделить на страницы, добавить у страниц зависимость от API.

Контрольные вопросы:

- [ ] React
- [ ] Props и состояние
- [ ] Компонент и элемент
- [ ] UseState и useEffect
- [ ] Жизненный цикл компонента
- [ ] CORS и обратный прокси
- [ ] Vite и Babel
- [ ] BFF и GraphQL
- [ ] Next.js и SSG
- [ ] FSD

Подробно:

Разработать базовый интерфейс приложения на `React` для `гостя`, аналогичный двум страницам из лабораторной работы №1 для просмотра `услуг`, а также сделать `главную` (стартовую) страницу со статическим описанием. При этом на странице списка `услуг` должны быть все необходимые фильтры (по диапазону дат, названию, цене) с фильтрацией на бэкенде. Использовать компоненты `React-Bootstrap`. Для карточек предусмотреть изображение по-умолчанию, если поле в `услуге` пустое.

В приложении должны быть навигационная панель `navbar` для списка базовых страниц, а также самописная навигационная цепочка `breadcrumbs`, где отображается путь от базовой страницы к текущей. В этой лабораторной никакого `Redux`, а `Context` вообще в курсе использовать нельзя.

Содержимое карточек получать из вашего веб-сервиса. Ajax-запросы написать самостоятельно через `fetch`. Ограничение с `CORS` решить через проксирование `React`. В методах `fetch` предусмотреть получение данных из коллекции с `mock`-объектами при отсутствии доступа к вашему бэкенду.

LAB6

Цель работы:

- [ ] Внедрение адаптивности
- [ ] Развертывание приложения

Задание:

- [ ] Внедрить менеджер состояний для хранения значений фильтров
- [ ] Добавление адаптивности и PWA

- [ ] Создание Tauri
- [ ] Развертывание в `Pages`

Диаграмма:

- [ ] **Deployment диаграмма:** все узлы и компоненты системы: фронтенда, web-сервера со статикой, веб-сервиса, базы данных и других хранилищ и тд. Узлы соединить протоколами, компоненты фронтенда и бэкенда поместить в узлах, указать API между ними.

Контрольные вопросы:

- [ ] Flux
- [ ] Схема redux (store, reducer, dispatch, action)
- [ ] Виды нативных приложений (PWA, Tauri и тд)
- [ ] Pages

Подробно:

Добавление менеджера состояний `Redux Toolkit` для хранения фильтра услуг. Рекомендуется использовать расширение для браузера redux-devtools. Необходимо развернуть фронтенд на `GitHub Pages` и добавить возможность работы в режиме `PWA`. Добавить адаптивность для трех страниц приложения.

Создание простого нативного приложения на `Tauri` для интерфейса гостя (без авторизации и редактирования), состоящий из 3 страниц с фильтрацией и картинками. Подключить приложение к разработанному API через IP адрес в локальной сети (не `localhost`).

LAB7

Цель работы:

- [ ] Завершение интерфейса `пользователя` в `React`

Задание:

- [ ] Добавить авторизацию и возможность оформления `заявок` во фронтенд через `Redux Toolkit` `redux-thunk`
- [ ] Кодогенерация взаимодействия с `API` через `Axios`

Диаграмма:

- [ ] **Activity диаграмма/BPMN:** для итогового бизнес-процесса для ДЗ: описание бизнес-процесса, разделение на 3 дорожки по ролям двух пользователей и выделенного сервиса (при наличии), действия соответствуют операциям пользователей в вашей системе.

Контрольные вопросы:

- [ ] Схема redux-toolkit (reducer, store, middleware)
- [ ] UseContext
- [ ] Axios
- [ ] Local storage

Подробно:

Добавить страницы для регистрации и авторизации. Добавить страницу для просмотра списка `заявок` пользователя в виде таблицы. Добавить в меню пункты для новых страниц. Добавление в `Redux Toolkit` состояния интерфейса после авторизации. В приложении должно быть реализовано переключение между интерфейсом гостя и интерфейсом пользователя по кнопке `Вход` / `Выход`. После авторизации в меню должно отображаться Имя/Логин пользователя. При выходе должно сбрасываться содержимое конструктора новой заявки, а также фильтры пользователя. Добавить страницу личного кабинета пользователя для сброса пароля и др полей.

Добавление на странице услуг кнопки `Добавить` для внесения данной услуги в новую заявку. Добавление страницы `заявки`, где в статусе `черновик` можно удалить уже добавленные в заявку услуги, поменять их количество или `подтвердить` заявку. Эта же страница используется для просмотра старых заявок в других статусах, но без возможности редактирования. Переход на страницу `заявки` через специальную кнопку, которая меняет состояние: если черновик есть - кнопка доступна, а если заявки-черновика нет - кнопка отображается с другим стилем и недоступна.

Для обращений к методам веб-сервиса использовать `axios`, кодогенерацию и `redux-thunk middleware`. При выполнении запросов отображать на странице анимацию.

LAB8

Цель работы:

- [ ] Внедрение Real-time web

Задание:

- [ ] Реализовать интерфейс `модератора`

Диаграмма:

- [ ] **Диаграмма состояний** для статусов `заявок`
- [ ] **Диаграмма прецедентов** интерфейса `React`

Контрольные вопросы:

- [ ] Любые вопросы по реализации интерфейса `модератора`
- [ ] Long Polling
- [ ] Agile
- [ ] DevOps
- [ ] UML
- [ ] GitFlow workflow

Подробно:

Необходимо добавить в приложение React интерфейс `модератора`, доступный после его авторизации и имеющий следующие отличия:

- Новое окно редактирования `услуг`, список услуг отображается таблицей. Доступно добавление новых услуг (обязательные и необязательные поля), редактирование, удаление.
- Добавить статические страницы 403 и 404
- В окне списка `заявок` доступны кнопки для смены статуса заявок. Также есть поля фильтрации по диапазону `даты формирования` (только даты, без времени) и статусу заявок на стороне бэкенда. Фильтрация по `создателю заявки` на фронтенде.
- Окно списка `заявок` переделать на `short polling` чтобы отображать актуальные статусы.

Дополнительное задание / Мобилка

Задание:

- [ ] Создание приложения для `гостя` на iOS/Android/Qt/React-native
- [ ] Подключение к веб-сервису

Контрольные вопросы:

- [ ] Виды нативных приложений
- [ ] Отличие нативных приложений от web-приложений
- [ ] React-native
- [ ] PWA
- [ ] Tauri

Подробно:

Создание простого нативного приложения для интерфейса гостя (без авторизации и редактирования), состоящий из 2 страниц с фильтрацией и картинками. Подключить приложение к разработанному API через IP адрес в локальной сети. Доработать диаграмму развертывания.