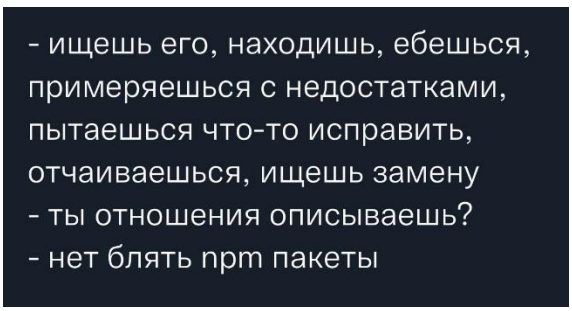
Национальный исследовательский университет компьютерных технологий, механики и оптики

Факультет ПИиКТ

Веб-программирование.
Лабораторная работа №4.

Вариант №30613



Работу выполнил: Кулаков Никита
Группа: P3230
Преподаватель: Каюков И.А.
Город: Санкт-Петербург
2021 год

Задание:

Переписать приложение из предыдущей лабораторной работы с использованием следующих технологий:

- Уровень back-end должен быть основан на Spring.
- Уровень front-end должен быть построен на React ~~+ Redux (необходимо использовать ES6 и JSX) с использованием набора компонентов PrimeReact.~~
- Взаимодействие между уровнями back-end и front-end должно быть организовано посредством REST API.

Поскольку было предоставлено дополнительное задание от практика, то технологии изменились.

- Redux изучить, но в лабе не использовать. Библиотеку компонентов взять https://github.com/JetBrains/ring-ui.
- Вместо redux-а взять effector.
- Create react app не использовать, сделать ssr-ую версию приложения с использованием loadable components.

Приложение по-прежнему должно включать в себя 2 страницы - стартовую и основную страницу приложения. Обе страницы приложения должны быть адаптированы для отображения в 3 режимах:

- "Десктопный" - для устройств, ширина экрана которых равна или превышает 1227 пикселей.
- "Планшетный" - для устройств, ширина экрана которых равна или превышает 787, но меньше 1227 пикселей.
- "Мобильный"- для устройств, ширина экрана которых меньше 787 пикселей.

Стартовая страница должна содержать следующие элементы:

- "Шапку", содержащую ФИО студента, номер группы и номер варианта.
- Форму для ввода логина и пароля. Информация о зарегистрированных в системе пользователях должна храниться в отдельной таблице БД (пароль должен храниться в виде хэш-суммы). Доступ неавторизованных пользователей к основной странице приложения должен быть запрещён.

Основная страница приложения должна содержать следующие элементы:

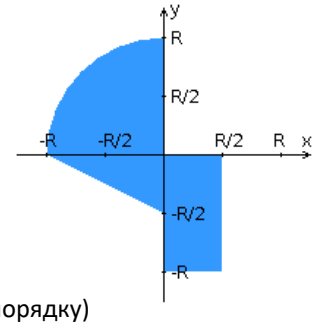
- Набор полей ввода для задания координат точки и радиуса области в соответствии с вариантом задания: Dropdown {'-2','-1.5','-1','-0.5','0','0.5','1','1.5','2'} для координаты по оси X, Text (-3 ... 3) для координаты по оси Y, и Dropdown {'-2','-1.5','-1','-0.5','0','0.5','1','1.5','2'} для задания радиуса области. Если поле ввода допускает ввод заведомо некорректных данных (таких, например, как буквы в координатах точки или отрицательный радиус), то приложение должно осуществлять их валидацию.
- Динамически обновляемую картинку, изображающую область на координатной плоскости в соответствии с номером варианта и точки, координаты которых были заданы пользователем. Клик по картинке должен инициировать сценарий, осуществляющий определение координат новой точки и отправку их на сервер для проверки её попадания в область. Цвет точек должен зависить от факта попадания / непопадания в область. Смена радиуса также должна инициировать перерисовку картинки.
- Таблицу со списком результатов предыдущих проверок.
- Кнопку, по которой аутентифицированный пользователь может закрыть свою сессию и вернуться на стартовую страницу приложения.

Дополнительные требования к приложению:

- Все результаты проверки должны сохраняться в базе данных под управлением СУБД Oracle.
- Для доступа к БД необходимо использовать Spring Data

Ссылки на репозитории:

- Фронтенд: https://github.com/zubrailx/web-lab-4-front
- Бэкенд: https://github.com/zubrailx/web-lab-4-back
- Альтернативная ссылка: https://github.com/zubrailx/university/tree/main/year-2/Web-programming/lab-4



(по порядку)

Вывод:

21 октября я решил сесть и за недельку написать данную лабораторную работу, однако даже не думал, что получится так, что я только одну божью коровку буду фиксить столько.

Итак, сначала решил сделать лабораторную на спринге по гайду от разработчиков, однако вся моя куча не запускалась, потому что нужно было сделать ssr с библиотекой компонентов от JetBrains с (как я уже понял под середину всего времени) модулем npm loadable components. Потом попытался через псевдо-ssr от ребят на medium, которые с помощью ant-run копировали в target статику, все равно не работало. Зато довольно неплохо понял maven: узнал для чего нужны properties, как конфигурировать com.github.eirslett, maven-antrun-plugin, исключать модули через exclude. Затем осознал, что единственный выход – использовать отдельный сервер (express) на нодах. Недельку посидел и позалипал в монитор не понимая, почему не работает babel (постоянно то пишет require или document is not defined, то import out of module), поразбирался с webpack. Через неделю с пустотой на душе данная проблема была решена.

Что узнал, пока работал с NodeJS: как конфигурировать babel (указывание версий поддержки браузерами, плагины для осуществления трансляции), webpack (rules, plugins, optimization, devtools->vendor, loaders, node-externals); отличие js, mjs, cjs; как запускать webpack-dev-server, использовать concurrently для параллельно разработки frontend и backend.

Касаемо технологий, которые были использованы, исключая вышеперечисленное: ignore-styles для исключения inline импорта scss стилей, @jetbrains/ring-ui библиотека компонентов, axios для осуществления ajax, await запросов на сервер express и spring, effector как хорошая альтернатива redux, сервер express, сессии на express-session для хранения контекста авторизации пользователя на нодах, хеширование с помощью js-sha256, moment для парсинга localDateTime, библиотека react для написания компонентов, react-dom, react-dom-router для реализации возможных путей для пользователя, loadable components для получения стилей, скриптов, кода html из компонентов, а также определения типов компонентов: ssr или нет.

Какие библиотеки npm затронул, но не использовал в итоге: express-cookies, isomophic-style-loader, typescript.

Подробнее про библиотеку react: узнал как писать компоненты в формате классов, функций, каковы преимущества одного над другим, как использовать ReactContext (отличие старого от нового, Redux) вместе с Provider, точнее понял, что для авторизации его использовать не очень, потому что он полностью асинхронный, что доставило очень много ненужных проблем, какими образами можно передавать пропсы в компоненты(...rest, {}, props). Разобрался с useState, временем жизни компонентов: рендером, монтированием, размонтированием. Также изучил и применил некоторые хуки: useRef, useEffect, useMemo, а также с помощью useEffect реализовал componentDidMount для отписки эффектов effector (потому что был баг с дублированием). Реализовал routes и сконфигурировал приватные и публичные пути.

Для обработки запросов и обновления токенов написал отдельный axios конфиг, который на каждый запрос с ответом 401, в котором указано, что токены сгорели, посылает async запрос на api/v1/user/refresh запрос с целью получения новых токенов.

Касаемо effector: изучил, что такое state, event, effect, domain какие эффекты можно навешивать на state, event (watch, on, map, prepend, filter, reset). Также узнал про функции combine, sample, restore, хук useStore для получения данных из store.

Из довольно обычного: нормально понял, как работают промисы, что такое async, await, использовал архитектурный шаблон a tomic для разграничения компонентов, узнал, что такое CORS, SSR.

Переходим к Spring: очень понравилось, что если конфигурация pom.xml не работает, то нужно откатить назад версию spring parent. Соответственно, в первый раз поработал на нем, отконфигурировал wildfly, для подключения логгера, чтобы затем перейти на tomcat, сделал так, чтобы пароль и логин для бд читались из файла credentials.txt для некой безопасности от @Fin1199. Сам сервер был построен на Rest API, через JWT были использованы accessToken для получения доступа к ресурсам, refreshToken для получения токенов, если они сгорели (время жизни 5 секунд для наглядности). На Spring Security использовал кастомный encryptor для паролей, OncePerRequestFilter для определения авторизации пользователей. Также разобрался, что сами фильтры влияют на аутентификацию, в то время как в SpringConfig configure() задаются параметры авторизации. Именно поэтому несмотря на то, что стоит permitAll, все равно проверяется хедер Authorization, если таковой указан. Поработал с Spring Data Jpa, внутри Entity сделал довольно интересную логику создания таблиц (попрактиковался на joinColumn, joinTable, manyToMany и т.п в этом роде). Кроме того, реализовал логику ролей (ROLE_ADMIN, ROLE_USER), однако на фронтенде не использовал из-за ненадобности. Подробное описание API можно найти на GitHub по ссылке, указанной выше.

О себе: перестал бояться читать чужой код, поскольку очень много проводил за поисками оптимального решения на GitHub случайных пользователей (честное слово, можете проверить мою лабораторную, там уровень копирайта точно не должен превышать 2% :), добавил 20 звезд). Понял, что мне нужно купить нормальный компьютер.