МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»
Кафедра математической кибернетики и компьютерных наук

РЕАЛИЗАЦИЯ КЛИЕНТСКОЙ ЧАСТИ УРБАНИСТИЧЕСКОЙ ВЕБ-ИГРЫ С ИСПОЛЬЗОВАНИЕМ HTTP И ПРОТОКОЛА WEBSOCKET

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 5 курса 551 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Рогова Максима Игоревича

Научный руководитель
Старший преподаватель _____ М. И. Сафрончик

Заведующий кафедрой
к. ф.-м. н. доцент _____ С. В. Миронов

Саратов 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3
1 Общая информация об игре 5
1.1 Правила игры 5
1.2 Клиентские модули игры 7
2 Анализ технологий 8
3 Реализация игры 9
3.1 Структура клиентской части 9
3.2 Контроль состояния приложения 9
3.3 Реализация методов для взаимодействия с сервером 10
3.4 Реализация роутера 10
3.5 Реализация модулей 11
3.5.1 Модуль авторизации 11
3.5.2 Модуль списка игр 11
3.5.3 Модуль игры 12
3.5.4 Модуль городского совета 12
ЗАКЛЮЧЕНИЕ 13
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 14

ВВЕДЕНИЕ

Для того чтобы объяснить городские процессы и подчинить их развитие интересам человека, появилась новая наука — урбанистика. Урбанистика — это наука, изучающая развитие городских сообществ и систем. Она аккумулирует достижения гуманитарных, социальных и технических наук для того, чтобы развивать города, делать их комфортнее и удобнее для жизни.

Целью настоящей работы является разработка клиентской части учебно-деловой урбанистической веб-игры для обучения игроков в данной сфере. Необходимо реализовать клиентскую часть приложения, состоящую из:

- модуля авторизации;
- модуля списка игр с возможностью регистрации и входа на конкретную игру;
- модуля игры.

Необходимо решить следующие подзадачи:

- решить задачу двустороннего обмена данных с сервером;
- решить задачу актуальности данных для всех участников игры;
- вышеописанные задачи должны решаться в рамках наличия API на .NET в full-REST архитектуре.

На клиенте используется следующий набор технологий:

- язык JavaScript;
- библиотека для создания SPA приложений ReactJs;
- библиотека ReactRouter для создания маршрутизации по приложению;
- библиотека Axios для написания запросов на сервер;
- библиотека moment.js для работы с датами;
- библиотека ASP .NET SignalR для работы с Websocket;
- библиотека Redux — менеджер состояния приложения;
- библиотека redux-saga — менеджер побочных эффектов.

Бакалаврская работа состоит из введения, трех разделов, заключения, списка использованных источников и пяти приложений. В первом разделе «Общая информация об игре» рассмотрена теоретическая часть о подобных играх, правила игры и основная информация о модулях игры. Во втором разделе «Анализ технологий» описаны использованные технологии, их преимущества и недостатки . Третий раздел «Реализация игры» содержит практическую часть работы: общую структуру, реализацию модулей, описание архитектурных ре-

шений.

1 Общая информация об игре

Существует много разновидностей учебно-деловых игр [1]. В основном это экономические и бизнес деловые игры, но ни одной урбанистической найти не удалось. Так же проблемой является то, что большинство экономических и бизнес игр не перенесены в доступный легковесный формат и большинство из них существует только в формате настольных игр. Поэтому возникла идея создать уникальную урбанистическую игру и при этом решить проблему доступности и удобства использования [2].

В качестве примеров таких игр можно привести:

- «Тренинг по ощущению бизнеса» — настольная игра и ее электронная версия «Собственный КАПИТАЛ». Они представляют экономико-управленческую модель организации производства и сбыта продукции.
- Деловые игры серии «БИЗНЕС-КУРС» — суть игр в управлении предприятием.

Было решено использовать формат веб-игры [3], для того чтобы игроку не нужно было устанавливать игру на компьютер. Формат игры на принятие решений не требует сложных 3d моделей и сценарных скриптов, в браузере будет проще работать с ее дизайном. В качестве основы для игры были взяты деловые бизнес игры, основанные на простом влиянии игрока на общую среду посредством пошагового принятия решений, но поскольку влияние на благоустройство города оказывает каждый его житель, то игра в отличии от бизнес аналогов переведена в командный формат. Благодаря этому был достигнут эффект влияния одного игрока на весь город [4].

1.1 Правила игры

Для реализации урбанистической деловой игры были разработаны правила. Одна игра будет состоять из нескольких городов, которые будут представлять из себя команды сформированные из заранее зарегистрированных игроков. Каждый город будет иметь набор следующих характеристик:

- Казна — общий бюджет города.
- Налог — вид налога, который взимается с игрока каждый цикл игры и уходит в бюджет города.
- Население — количество игроков в данном городе.
- Мэр — выбирается вначале игры и несет набор положительных/отрица-

тельных эффектов, сказывающихся на городе.

— Рейтинг — динамически меняется на основание всех характеристик игры, в соответствие с действиями игроков.

Так же каждый игрок имеет личные характеристики изначально равные для всех:

— Бюджет — текущие, личные накопления игрока в рублях.

— Налог в процентах или фиксированная сумма, который игрок будет платить в казну города каждый цикл игры в соответствие с налоговой политикой его города.

— Зарплата — в рублях, суммируется к личному бюджету игрока каждый цикл игры.

Цикл игры представляет из себя набор повторяющихся и не повторяющихся стадий, на каждую из которых отведен таймер, по истечению которого игра переходит в следующую стадию. Стадии игры:

— Ожидает начала — игра создана, но дата начала игры не наступила — таймер до начала игры.

— Начало игры — время после старта игры, отведенное для того, чтобы игрок ознакомился с правилами и для события выдачи первой зарплаты всем игрокам.

— Выбор мэра — время, отведенное для выбора мэра для каждой команды, путем голосования всех игроков за одного из кандидатов из заданного списка.

— Ход — происходит бросание кубика, и каждый игрок, в соответствие со своим случайным числом, передвигается на соответствующую клетку карты игры, так же на стадии хода происходит событие клетки. Каждой клетке соответствует уникальное событие, где игрок принимает личное решение, которое может повлиять на позицию игрока на карте и траты из личного бюджета. Стадия циклично повторяется заданное количество раз.

— Стадия городского совета — наступает после прохождения заданного количества ходов, на которой происходит решение ограниченного количества проблем города из списка, посредством голосования игроков. Для каждой проблемы предлагается заданный список решений, каждое из которых оказывает влияние на благоустройство города. Совет цикличен

в зависимости от общего количества ходов на игру.

— Конец игры — по истечении заданного для игры количества ходов и советов, побеждает город, который имеет больший рейтинг, исходя из принятых решений на стадиях ход и совет.

Карта игры представляет из себя 24 клетки, по которым игрок будет передвигаться на стадии хода, описанной выше. На карте игры будут присутствовать несколько клеток особенных событий, не перечисленных выше:

— Клетка старта — выдача зарплаты.

— Клетка переезда — попадая на которую игрок имеет возможность бесплатного переезда в город-оппонент. В игре будет возможность переезда на любой из стадий, но за плату из личного бюджета.

— Клетка налога — попадая на которую игрок платит налог, в соответствие с вышеописанной налоговой системой, в казну города.

1.2 Клиентские модули игры

В соответствии с описанием и правилами, игра была поделена [5] на соответствующие модули [6]:

— Модуль авторизации и регистрации — нужен для возможности зарегистрировать учетную запись и войти по ней [7] для регистрации на игру [8].

— Модуль списка игр — список игр, где каждый элемент является отдельной игрой с данными о ней и возможностью регистрации и входа.

— Модуль игры — основная страница игры.

— Модуль городского совета — отдельная стадия игры со списком проблем, списком решений на каждую из проблем, и возможностью голосовать за решения.

2 Анализ технологий

Выбор технологий для реализации клиентской части основан на технических особенностях игры. При выборе основного фреймворка/библиотеки рассматривается два фактора, которые больше всего интересуют в данном проекте: высокая производительность при высокой динамике обновления данных и обратная совместимость, т.к. игру планируется активно развивать далее [9]. При рассмотрении трех самых популярных фреймворков Angular, React и View выбор пал на React [10], так как в Angular плохо реализована обратная совместимость, это полноценный фреймворк у которого свой CLI и его нельзя обновить без миграции, а у View не смотря на более высокую производительность загрузки, производительность перерисовок проигрывает React.

Далее необходимо выбрать способ общения с сервером. Так как игра будет использовать глобальные таймеры, одинаковые для всех игроков, понадобится двусторонний обмен данным, который обеспечивает протокол WebSocket [11]. Но поскольку в игре есть уникальные для каждого пользователя действия, требующие обработки ошибок, необходимо комбинировать WebSocket [12] с обычным Http, так как любая успешная передача данных по WebSocket, никогда не является ошибкой и придется писать на каждую ошибку отдельную обработку, как на сервере так и на клиенте. Необходимо выбрать библиотеку для WebSocket [13], так как у него есть ряд проблем:

- отсутствие кроссбраузерности — не все браузеры поддерживают WebSocket;
- отсутствие каналов — у чистого WebSocket нет инструментов персонализировать данные, он всегда отправляет данные всем подписчикам.

При выборе библиотеки так же необходимо было обратить внимание на совместимость этой библиотеки с сервером. Так как серверная часть реализована на .NET, идеальным вариантом стала SignalR — это библиотека microsoft, для работы с WebSocket [14].

Так же для клиента важную роль играет управление состояниям приложения. Этот выбор очень сильно зависит от совместимости с другими технологиями и личными предпочтениями. В этом проекте необходимо обрабатывать очень много побочных эффектов от WebSocket и HTTP запросов, поэтому решено было использовать Redux-Saga [15], а соответственно сопутствующий менеджер состояния [16] Redux [17].

3 Реализация игры

3.1 Структура клиентской части

Проект состоит из клиента модуля авторизации и самой игры [18], с одинаковой структурой:

- api — папка с основным классом для взаимодействия с сервером через http, и разбиения конкретных методов взаимодействия по сущностям;
- assets — папка для хранения статических ресурсов: картинок, шрифтов, глобальных css стилей;
- components — функциональные компоненты, которые не взаимодействуют напрямую с данными приложения, чье поведение может реагировать на набор аргументов;
- constants — папка для хранения глобальных статических переменных;
- hooks — для хранения функций генераторов;
- modules — папка для хранения основных компонент приложения, которые взаимодействуют с данным через redux, каждый модуль состоит из своей папки components и папки redux, в которой хранятся обработчики данных;
- pages — для хранения контейнеров, каждый из которых представляет собой одну страницу приложения и собирает в себе реализованные модули;
- redux — папка для хранения глобальных обработчиков данных, не относящихся ни к одному модулю приложения;
- utils — для чистых функций обработчиков, например, форматирование дат, чисел и т.д.;
- файл App.js — общая обертка приложения, где реализуется роутер приложения и общий контейнер;
- файл index.js — точка входа приложения.

3.2 Контроль состояния приложения

Для управления состоянием приложения и построения архитектурного решения используется библиотека Redux, основанная на принципе взаимодействия чистых функций и архитектурного подхода Flux [19]. Это библиотека с открытым исходным кодом, написанная Даниилом Абрамовым в 2015г. Реализованная с целью контроля состояния приложений, которые работают с большим количеством динамических данных. Так как Redux базируется на

Flux соответственно обмен данными происходит через один поток и реализует для этого четырехступенчатую систему взаимодействия:

- store — общий контейнер, реализующий в себе хранилище для данных всего приложения;
- reducer — функция, которая хранит в себе логику взаимодействия конкретной логической сущности со store;
- actionCreater — чистая функция, создающая событие с конкретным именем, соответствующим с каким либо ActionType в любом из reducers;
- dispatcher — функция сообщающая reducers, какое событие необходимо воспроизвести в данный момент.

Так же для разработки архитектурного решения воспользуемся библиотекой redux-saga, которая является оберткой над Redux, использующая для этого функции генераторы и предоставляющая методы для прямого взаимодействия с Redux.

3.3 Реализация методов для взаимодействия с сервером

Поскольку было решено в одном приложении использовать и websocket и http, необходимо реализовать для каждого из способов взаимодействия отдельный механизм.

Для обмена данными через http, воспользуемся библиотекой axios, которая обеспечивает более простое взаимодействие со стандартным XmlHttpRequest и напишем на ее основе класс Singlton, который будет подставлять домен API из переменных окружения, текущий токен авторизации из Cookie, обрабатывать общие кейсы ошибок с одинаковым поведением в интерфейсе, например, 401 и 404 ошибки, и будет реализовывать 4 основных REST метода: POST, PUT, GET, DELETE.

Далее реализуется функция для взаимодействия с сервером через WebSocket, воспользовавшись вышеописанные библиотекой SignalR. Функция реализуют экземпляр соединения с определенными заранее настройками и подставленным из переменных окружения адресом api.

3.4 Реализация роутера

Приложение будет состоять из 3 страниц и модальных окон. Так как доступ к приложению должен быть ограничен авторизацией пользователя, ре-

ализуем компонент приватного роута. Компонент возвращает страницу при наличии токена, иначе возвращает обратно на авторизацию. Так же выполняет get запрос на получение данных о пользователе на каждой странице, дабы иметь всегда актуальный id, и проверять 401 ошибку закончившейся авторизации.

3.5 Реализация модулей

После того как создана основная архитектура приложения и описаны основные многократно используемые сущности, можно приступить к реализации самих модулей игры [20].

3.5.1 Модуль авторизации

В модуле авторизации WebSocket не нужен, поскольку существует всего два запроса с отправкой данных для входа и регистрации. Исходя из того, что модуль авторизации представляет из себя две формы, воспользуемся библиотекой Formik, которая представляет собой контейнер собирающий данные формы и предоставляющий методы get, set для управления данными. Контейнер реализует следующую логику: предоставляет всем потомкам (полям) свои методы, устанавливает серверные ошибки при их наличии, и принимает в себя набор полей. Далее необходимо реализовать универсальные поля, которые при попадании в вышеописанный контейнер подключаются к нему. Поле реализует следующую логику: подключается к контейнеру, выводит ошибку, позволяет вывести при необходимости label. При успешной отправке формы на сервер, соответственно получаем токен, записываем его в cookie и переадресовываем пользователя на игру.

3.5.2 Модуль списка игр

Данный модуль представляет из себя таблицу с созданными играми, где каждая строка хранит информацию о конкретной игре. Модуль состоит из компонента обертки где будет запрашивать список с сервера. Так же компонент, который реализует одну строку таблицы, форматируя пришедшие данные. Для него был написан алгоритм склоняющий существительные под число в русском языке, так как в каждой строке выводится количества ходов и игроков. И

в нем же реализована отправка запроса на регистрацию пользователя в игру, если он еще на нее не зарегистрирован.

3.5.3 Модуль игры

Данный модуль дробиться на множество сущностей каждая из которых имеет свою подписку на определенную WebSocket команду API:

- Компонент карты.
- Компонент информации о городе.
- Компонент заголовков — информация о текущем таймере и ходе.
- Компонент профиля — личная информация игрока (бюджет, зарплата, размер налога, имя).
- Компоненты событий: хода, приветствия, переезда, компонент кубика.

3.5.4 Модуль городского совета

Модуль городского совета будет состоять из следующих компонент:

- Компонент контейнер - принимающий данные с сервера и собирающий весь модуль.
- Компонент одной проблемы.
- Компонент списка решений.

ЗАКЛЮЧЕНИЕ

В ходе баклаврской работы была реализована клиентская часть учебно-деловой веб-игры с использованием:

- ReactJS;
- Websocket;
- SignalR;
- Redux;
- Redux-saga.

Показаны преимущества обмена данными через протокол WebSocket с использованием его в связке с библиотекой SignalR в рамках проекта, где была важна скорость и частота обмена данными, т.к. многое было связано с таймерами, это хорошо видно на примере игры, построенной на единоличных и командных принятых решениях в условиях ограниченного времени. Так же были продемонстрированы преимущества подхода к реализации, где побочные эффекты, например асинхронные операции обрабатываются отдельно от бизнес-логики приложения. Продемонстрировано преимущество Flux подхода к управлению состоянием приложения в клиентской части.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Проектирование пошаговых онлайн-игр. [Электронный ресурс]. — URL: https://cyberleninka.ru/article/n/proektirovanie-poshagovyh-onlayn-igr (Дата обращения: 08.05.2021). Загл. с экр. Яз. рус.

2 *Сысолетин, Е. Г.* Проектирование интернет-приложений. Учебно-методическое пособие / Е. Г. Сысолетин. — Екатеринбург: Уральский федеральный университет, 2015.

3 Разработка браузерной профориентационной игры. [Электронный ресурс]. — URL: https://cyberleninka.ru/article/n/razrabotka-brauzernoy-proforientatsionnoy-igry (Дата обращения: 10.05.2021). Загл. с экр. Яз. рус.

4 Разработка отказоустойчивой распределенной многопользовательской игры. [Электронный ресурс]. — URL: https://cyberleninka.ru/article/n/razrabotka-otkazoustoychivoy-raspredelennoy-mnogopolzovatelskoy-igry (Дата обращения: 11.05.2021). Загл. с экр. Яз. рус.

5 Проектирование программного обеспечения с помощью микросервисной архитектуры. [Электронный ресурс]. — URL: https://cyberleninka.ru/article/n/proektirovanie-programmnogo-obespecheniya-s-pomoschyu-mikroservisnoy-arhitektury (Дата обращения: 14.05.2021). Загл. с экр. Яз. рус.

6 *Мартин, Р.* Чистая архитектура. Искусство разработки программного обеспечения / Р. Мартин. — СПб.: Питер, 2018.

7 Защита персональных данных при авторизации пользователя в распределенных информационных системах, построенных на основе Web-технологий. [Электронный ресурс]. — URL: https://cyberleninka.ru/article/n/zaschita-personalnyh-dannyh-pri-avtorizatsii-polzovatelya-v-raspredelennyh-informatsionnyh-sistemah-postroennyh-na-osnove-web (Дата обращения: 15.05.2021). Загл. с экр. Яз. рус.

8 Оценка возможности использования микросервисной архитектуры при разработке пользовательских интерфейсов клиент-серверного программного обеспечения. [Электронный ресурс]. — URL:

https://cyberleninka.ru/article/n/otsenka-vozmozhnosti-ispolzovaniya-mikroservisnoy-arhitektury-pri-razrabotke-polzovatelskih-interfeysov-klient-servernogo (Дата обращения: 13.05.2021). Загл. с экр. Яз. рус.

9 Разработка клиентской части веб-приложения с использованием технологий spa. [Электронный ресурс]. — URL: https://cyberleninka.ru/article/n/razrabotka-klientskoy-chasti-veb-prilozheniya-s-ispolzovaniem-tehnologiy-spa (Дата обращения: 15.05.2021). Загл. с экр. Яз. рус.

10 A Javascript library for building user interfaces ? React. [Электронный ресурс]. — URL: https://reactjs.org/ (Дата обращения: 21.05.2021). Загл. с экр. Яз. англ.

11 websocket MDN [Электронный ресурс]. — URL: https://developer.mozilla.org/ru/docs/WebSockets (Дата обращения: 19.05.2021). Загл. с экр. Яз. рус.

12 RFC websocket [Электронный ресурс]. — URL: https://tools.ietf.org/html/rfc6455 (Дата обращения: 20.05.2021). Загл. с экр. Яз. англ.

13 websocket learnjs [Электронный ресурс]. — URL: https://learn.javascript.ru/websocket (Дата обращения: 20.05.2021). Загл. с экр. Яз. рус.

14 SignalR [Электронный ресурс]. — URL: https://docs.microsoft.com/ru-ru/aspnet/signalr/overview/getting-started/introduction-to-signalr (Дата обращения: 20.05.2021). Загл. с экр. Яз. рус.

15 Redux-saga. [Электронный ресурс]. — URL: https://redux-saga.js.org/docs/About (Дата обращения: 21.05.2021). Загл. с экр. Яз. англ.

16 Принципы и подходы к Frontend архитектуре веб-приложений [Электронный ресурс]. — URL: https://cyberleninka.ru/article/n/sovremennye-printsipy-i-podhody-k-frontend-arhitekture-veb-prilozheniy (Дата обращения: 18.05.2021). Загл. с экр. Яз. рус.

17 Redux. [Электронный ресурс]. — URL: https://redux.js.org/introduction/getting-started (Дата обращения: 21.05.2021). Загл. с экр. Яз. англ.

18 Микросервисная архитектура приложения как система, упрощающая разработку и поддержку кода. [Электронный ресурс]. — URL: https://cyberleninka.ru/article/n/mikroservisnaya-arhitektura-prilozheniya-kak-sistema-uproschayuschaya-razrabotku-i-podderzhku-koda (Дата обращения: 23.05.2021). Загл. с экр. Яз. рус.

19 Сравнение современных реализаций Flux-архитектур разработки веб-приложений. [Электронный ресурс]. — URL: https://cyberleninka.ru/article/n/issledovanie-i-sravnenie-sovremennyh-realizatsiy-flux-arhitektur-razrabotki-veb-prilozheniy (Дата обращения: 22.05.2021). Загл. с экр. Яз. рус.

20 *Грэди, Б.* Объектно-ориентированный анализ и проектирование с примерами приложений / Б. Грэди. — М.: Вильямс, 2017.