SET09103
Advanced Web Technologies

Coursework Part #2

Vinh Phat Tu
40507973

December 2023

Contents

1 Introduction

This coursework part will discuss the implementation of recreating /r/place, an event and project made by Reddit.com, a discussion board website.

2 Implementation

Squarish is a React application that presents a 1000x1000 grid of squares which registered users can modify. The user selects a color from the preset colors on the left hand side and then select the square they wish to color. Each user can only change a square every five minutes, allowing for collaboration or competition amongst the player base.

2.1 Link

The website is currently deployed on the dedicated university VM on:

`http://webtech-2324-57.napier.ac.uk/` and `http://146.176.251.242`.

The repository is available on GitHub:

`https://github.com/lawclaw/squarish`

3 Comparison from outlined plan

3.1 Features

All the planned features from part 1 of the coursework have been implemented including the additional ones (refer to 2, 3). The application includes an interactive 1000 x 1000 grid. Users can change squares that are sent to the server using Socket.io and the WebSocket protocol. For additional features, a basic login system using email and password has been implemented along with the timeout functionality.

New features in the final product was as followed:

Feature	Description	Reason
Color palette	Set color palette for users to pick	Essential functionality
'About us'	'About' section describing the purpose of the website and controls	Explanations for user
Alert banner	Alert banner for conveying information like errors	Make information visible and accessible
Database	Database	Storing information of users and grid
Panning	Navigating the canvas	User controls

Table 1: New features in final product

3.2 Wireframe

As for the initial wireframe (refer to 1), the design for the navigation bar remained the same. One notable change in the final version is the removal of the quick actions. This change came about after analyzing the users' use cases of the page. For most users, the main action outside of clicking on the canvas, was changing their selected color. With one primary action there was little justification to spend development time on quick actions, and thus it was removed and replaced with the color palette.

3.3 Navigation tree

As for navigation tree (refer to 2), the design has remained relatively the same. The most significant change is that the React frontend serves three pages: home, login and signup. The 'about_us' page was put in the home page as a smaller section.

3.4 Architecture

As shown in 4, the architecture of the application can be broken down into three tiers: frontend, backend and database. The addition of a database became apparent as the authentication mechanism was developed. User data and accounts needed to be stored at a central place where it could be easily retrieved and modified.

Unlike the traditional Jinja templates (which are served by the Flask application), the React frontend operates on its own and renders websites. The backend is therefore a JSON-RPC API that handles incoming requests from HTTP and Websocket.

3.4.1 PocketBase

PocketBase is both a backend and database service, all ready to go with a single executable. It allows for developers to quickly have a database without needing to install and configure unlike other database engines (MySQL, SQLite and so on). PocketBase is based on SQLite and has a UI (admin dashboard) where developers and admins can manage the database.

The main reason for choosing PocketBase comes from the ease of use and the developer experience. For a small project like this coursework, being able to quickly create to modify collections and entries, was the winning factor. PocketBase also offers ready-to-go endpoints and configuration to abstract from using direct queries when not needed 3. There are pain points when dealing with databases and configuration which again would take time away from development and thus other options such as MySQL and native SQLite were discarded.

3.5 Data

The application stores two types of data in two collections. The first collection: 'users', store the registered users. The second colletion: 'grids', store the different grids or canvases.

4 Reflection

4.1 WebSocket and Socket.IO

This coursework gave me an opportunity to explore the WebSocket protocol and the Socket.IO library. Unlike HTTP, this protocol allows for real-time communication and broadcasting which again in some use cases is critical.

4.2 JSON Web Token (JWT)

Going deep into authentication and learning about JWT has been also interesting. Although quite painful at times (such as having to choose between cookies vs. localStorage).

4.3 Grid rendering and controls

During development, understanding how to render one million squares (1000 x 1000 grid) was the most time-consuming task. Initially a regular <canvas> element was used to display the grid which proved to have a significant slow-down on any grid larger than 100 x 100.

After due research, the main strategy for rendering large canvases was to only render what the user could see and thus implement a camera functionality For example, if the client was zoomed in on a specific area, then it would be more efficient to only render those specific squares rather than loading everything. Creating a camera or viewport created very much complexity and this idea was in the end put on-hold.

The current solution was to use a library called 'react-virtualized'. The library offers a Grid component which only renders visible squares as a grid, improving both performance and load time.

4.4 Microservice Architecture

By developing the backend as an API rather than a webpage serving application, there is a separation of concerns. For example, having an API allows for any HTTP compatible client to request the information they need. The backend simply responds with data which any client can use to present to their liking. If another developer wishes to write an Android application based of the data, the API allows them to retrieve the data without needing to refactor the code.

This also comes with more complexity over using Jinja templates. But gaining a hands-on experience in developing a microservice architecture has been interesting.

5 Future considerations

The main sore point at this stage is the grid. At the moment, the grid always starts in the top-left corner. The grid has limited x and y panning, and also lacks zooming. Therefore, the main priority for the future is to develop a new grid which can be panned and zoomed. A potential solution explored was PixiJS, a library dedicated to rendering canvases.

There is also missing functionality with the login system such as a recovery mechanism. If a user forgets their password, there should be a recovery step (which doesn't exist at the moment).

Another improvement is handling logout. Since JWT tokens are stored in localStorage, there is no set expiration time unlike cookies. Furthermore, if the user doesn't log out after 24 hours (when token has expired on server-side), there could be unexpected behaviours which have not been tested yet.

A Figures and tables from Part #1

A.1 Features from Part #1

Feature	Description	Reason
Canvas	Implement a pixel canvas that will display the current state	Main feature of the website
Websocket	Using websocket protocol to allow for real-time updates	Give users instant feedback on changes to the canvas

Table 2: Main features

Feature	Description	Reason
Login system	System to track users	Potential personalisation
Timeout	Restrict canvas editing frequency	Reduce bot behavior

Table 3: Additional features

A.2 Wireframe

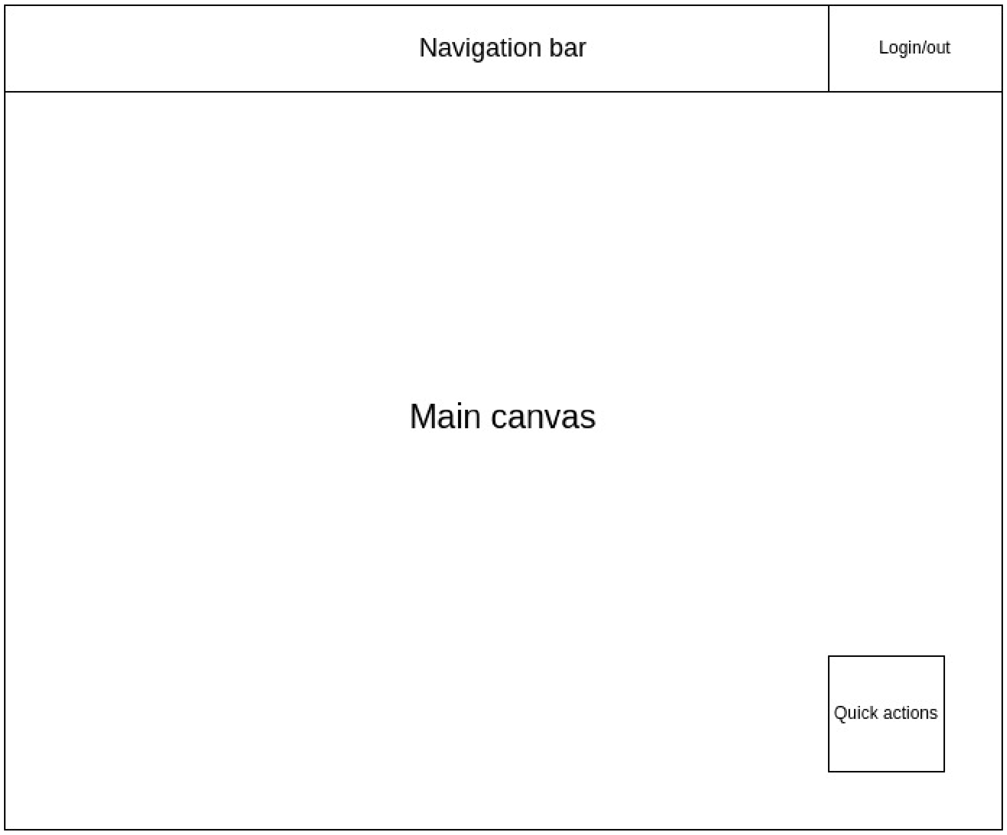


Figure 1: Wireframe for application

A.3 Navigation tree

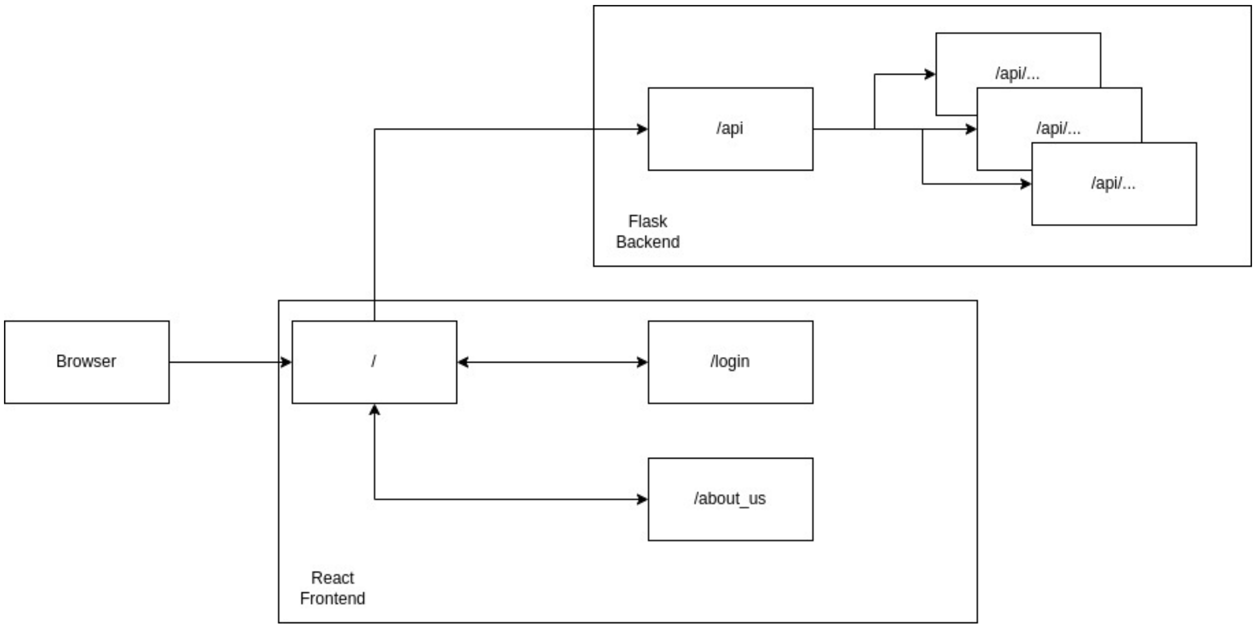
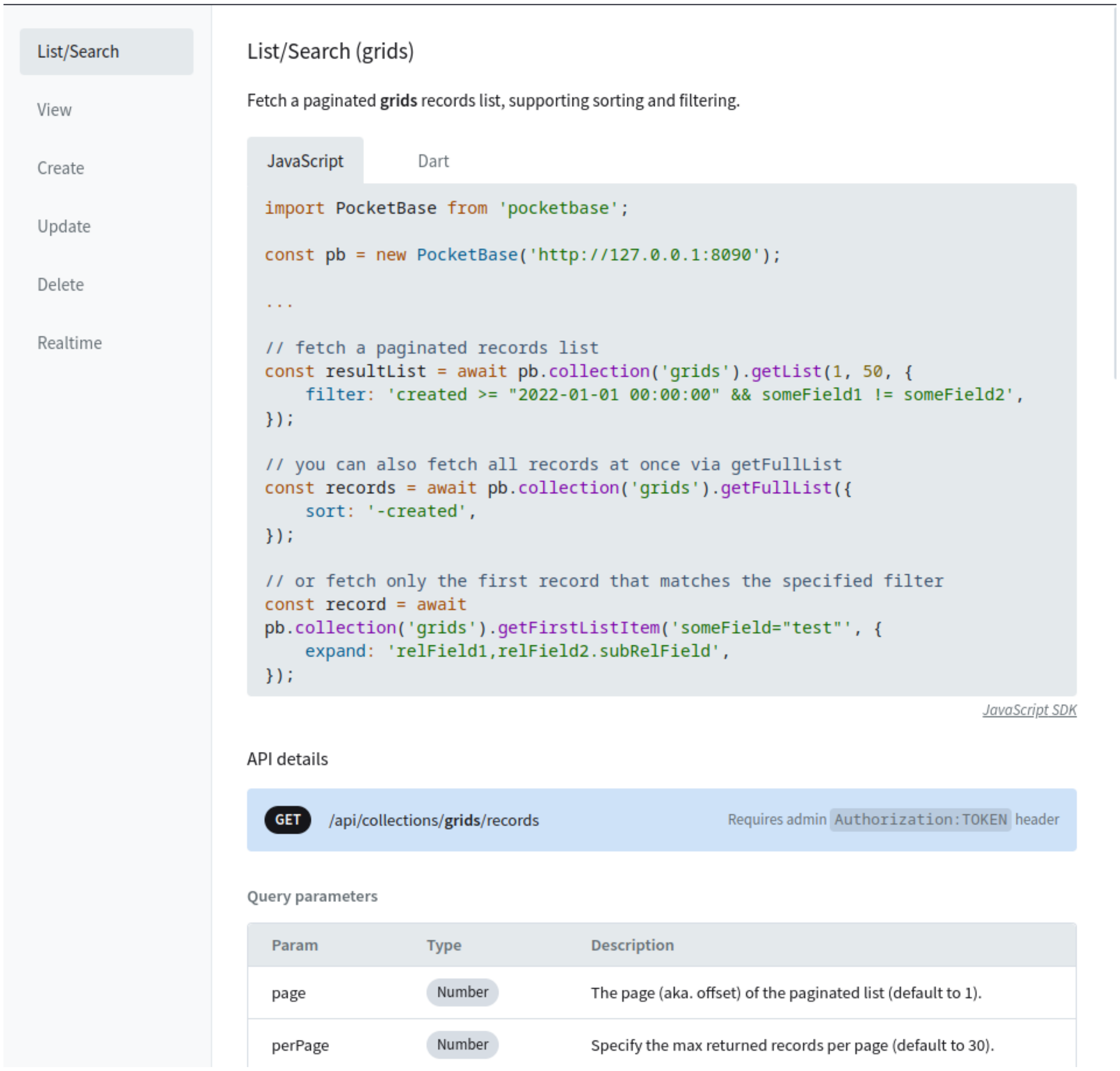


Figure 2: Navigation tree for single-page React application with Flask backend

B PocketBase screenshots



List/Search (grids)

Fetch a paginated **grids** records list, supporting sorting and filtering.

```
import PocketBase from 'pocketbase';

const pb = new PocketBase('http://127.0.0.1:8090');

...

// fetch a paginated records list
const resultList = await pb.collection('grids').getList(1, 50, {
    filter: 'created >= "2022-01-01 00:00:00" && someField1 != someField2',
});

// you can also fetch all records at once via getFullList
const records = await pb.collection('grids').getFullList({
    sort: '-created',
});

// or fetch only the first record that matches the specified filter
const record = await
pb.collection('grids').getFirstListItem('someField="test"', {
    expand: 'relField1,relField2.subRelField',
});
```

API details

GET /api/collections/**grids**/records Requires admin `Authorization:TOKEN` header

Query parameters

Param	Type	Description
page	Number	The page (aka. offset) of the paginated list (default to 1).
perPage	Number	Specify the max returned records per page (default to 30).

Figure 3: Endpoint preview

C Architecture diagram

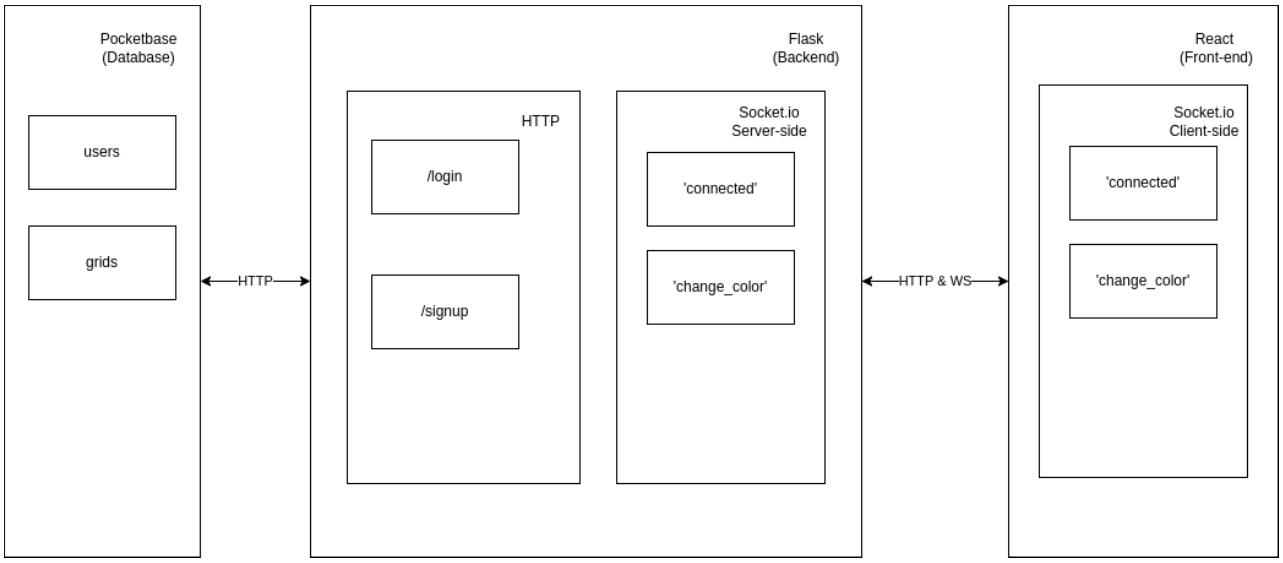


Figure 4: Architeture for Squarish