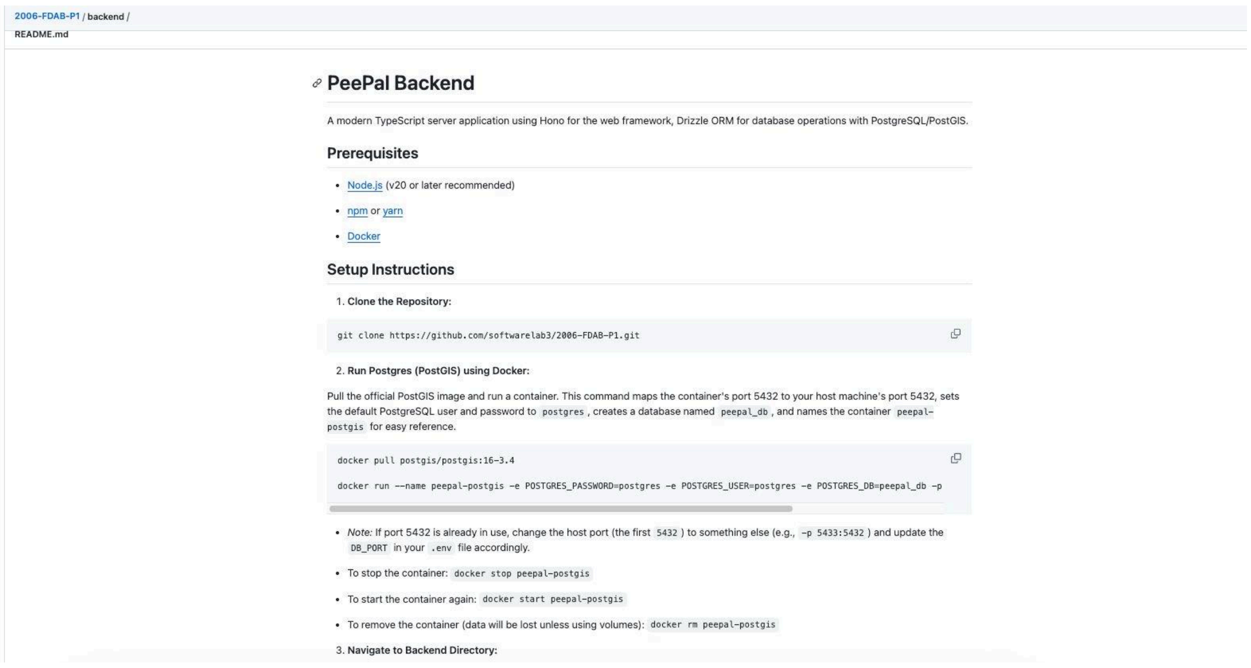
Documentations

Table of Contents

README

Our repository features detailed and instructive readme for frontend and backend components. These documents act as comprehensive guides, assisting future developers in effectively setting up, developing and maintaining the project. The documents are important in maintaining long-term code consistency and fostering robust development practices.

To access the complete documentation, please visit our GitHub pages for the main repository, frontend and backend.



2006-FDAB-P1 / backend /
README.md

PeePal Backend

A modern TypeScript server application using Hono for the web framework, Drizzle ORM for database operations with PostgreSQL/PostGIS.

Prerequisites

- Node.js (v20 or later recommended)
- npm or yarn
- Docker

Setup Instructions

1. **Clone the Repository:**

```
git clone https://github.com/softwarelab3/2006-FDAB-P1.git
```

2. **Run Postgres (PostGIS) using Docker:**

Pull the official PostGIS image and run a container. This command maps the container's port 5432 to your host machine's port 5432, sets the default PostgreSQL user and password to `postgres`, creates a database named `peepal_db`, and names the container `peepal-postgis` for easy reference.

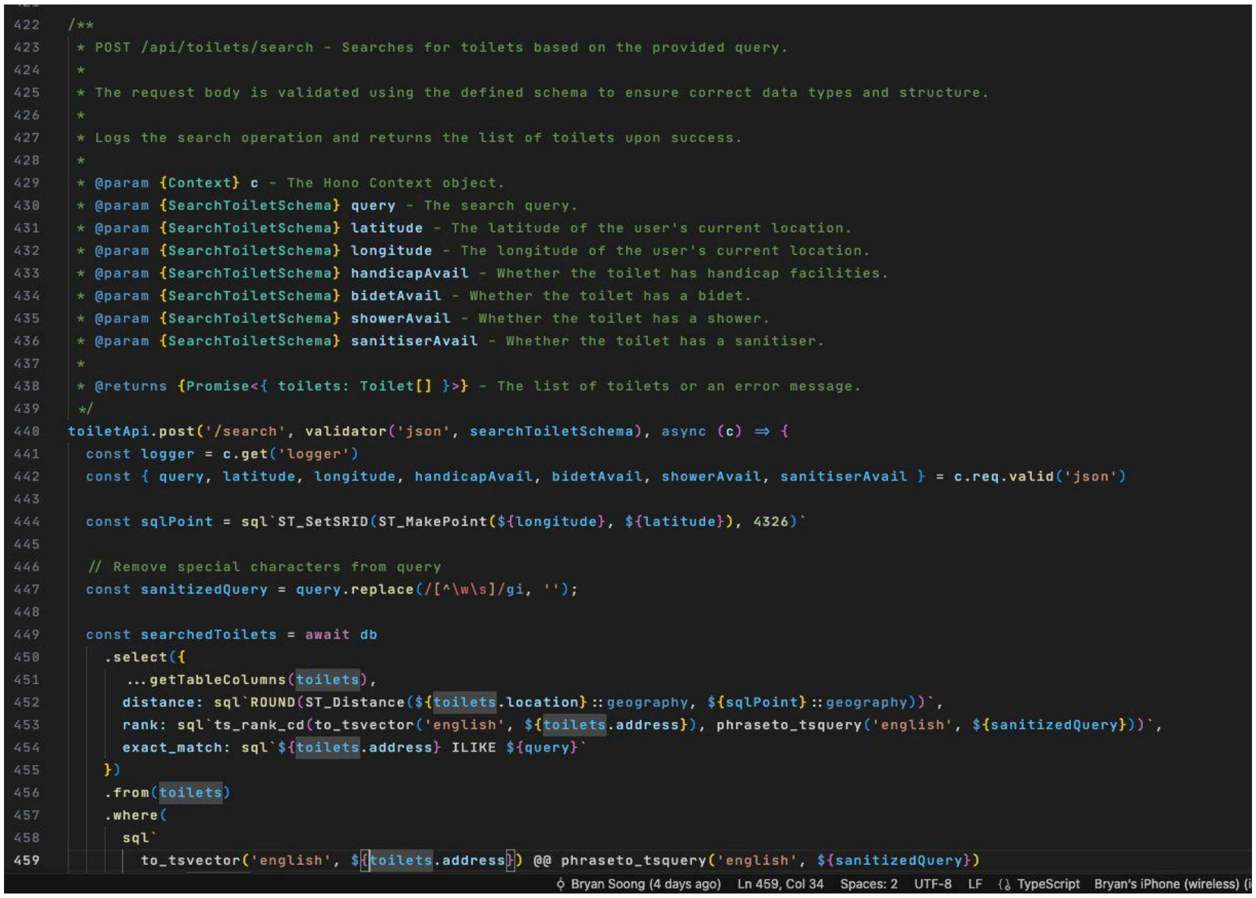
```
docker pull postgis/postgis:16-3.4

docker run --name peepal-postgis -e POSTGRES_PASSWORD=postgres -e POSTGRES_USER=postgres -e POSTGRES_DB=peepal_db -p
```

- *Note:* If port 5432 is already in use, change the host port (the first `5432`) to something else (e.g., `-p 5433:5432`) and update the `DB_PORT` in your `.env` file accordingly.
- To stop the container: `docker stop peepal-postgis`
- To start the container again: `docker start peepal-postgis`
- To remove the container (data will be lost unless using volumes): `docker rm peepal-postgis`

3. **Navigate to Backend Directory:**

Code Comments

We have included comprehensive and insightful comments in our code.



```
/**
 * POST /api/toilets/search - Searches for toilets based on the provided query.
 *
 * The request body is validated using the defined schema to ensure correct data types and structure.
 *
 * Logs the search operation and returns the list of toilets upon success.
 *
 * @param {Context} c - The Hono Context object.
 * @param {SearchToiletSchema} query - The search query.
 * @param {SearchToiletSchema} latitude - The latitude of the user's current location.
 * @param {SearchToiletSchema} longitude - The longitude of the user's current location.
 * @param {SearchToiletSchema} handicapAvail - Whether the toilet has handicap facilities.
 * @param {SearchToiletSchema} bidetAvail - Whether the toilet has a bidet.
 * @param {SearchToiletSchema} showerAvail - Whether the toilet has a shower.
 * @param {SearchToiletSchema} sanitiserAvail - Whether the toilet has a sanitiser.
 *
 * @returns {Promise<{ toilets: Toilet[] }>} - The list of toilets or an error message.
 */
toiletApi.post('/search', validator('json', searchToiletSchema), async (c) => {
  const logger = c.get('logger')
  const { query, latitude, longitude, handicapAvail, bidetAvail, showerAvail, sanitiserAvail } = c.req.valid('json')

  const sqlPoint = sql`ST_SetSRID(ST_MakePoint(${longitude}, ${latitude}), 4326)`

  // Remove special characters from query
  const sanitizedQuery = query.replace(/[^\w\s]/gi, '');

  const searchedToilets = await db
    .select({
      ...getTableColumns(toilets),
      distance: sql`ROUND(ST_Distance(${toilets.location}::geography, ${sqlPoint}::geography))`,
      rank: sql`ts_rank_cd(to_tsvector('english', ${toilets.address}), phraseto_tsquery('english', ${sanitizedQuery}))`,
      exact_match: sql`${toilets.address} ILIKE ${query}`
    })
    .from(toilets)
    .where(
      sql`
        to_tsvector('english', ${toilets.address}) @@ phraseto_tsquery('english', ${sanitizedQuery})
```

Bryan Soong (4 days ago)   Ln 459, Col 34   Spaces: 2   UTF-8   LF   TypeScript   Bryan's iPhone (wireless)