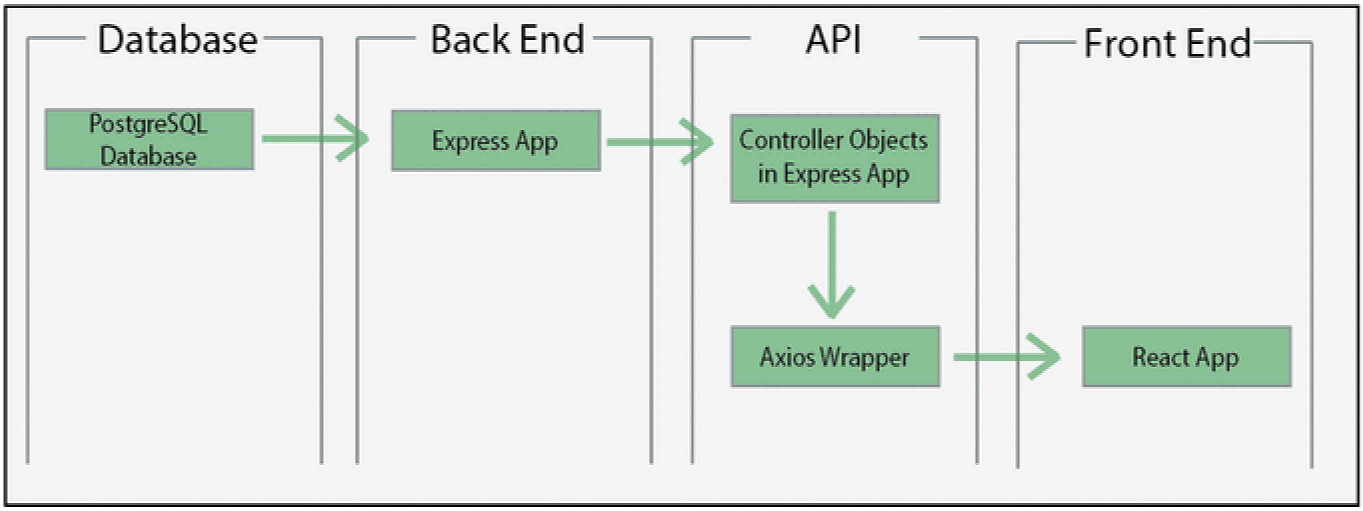
Developer Guide - Getting Started with Potato Server

Welcome to Potato Server! 🥔

https://potato.colab.duke.edu/

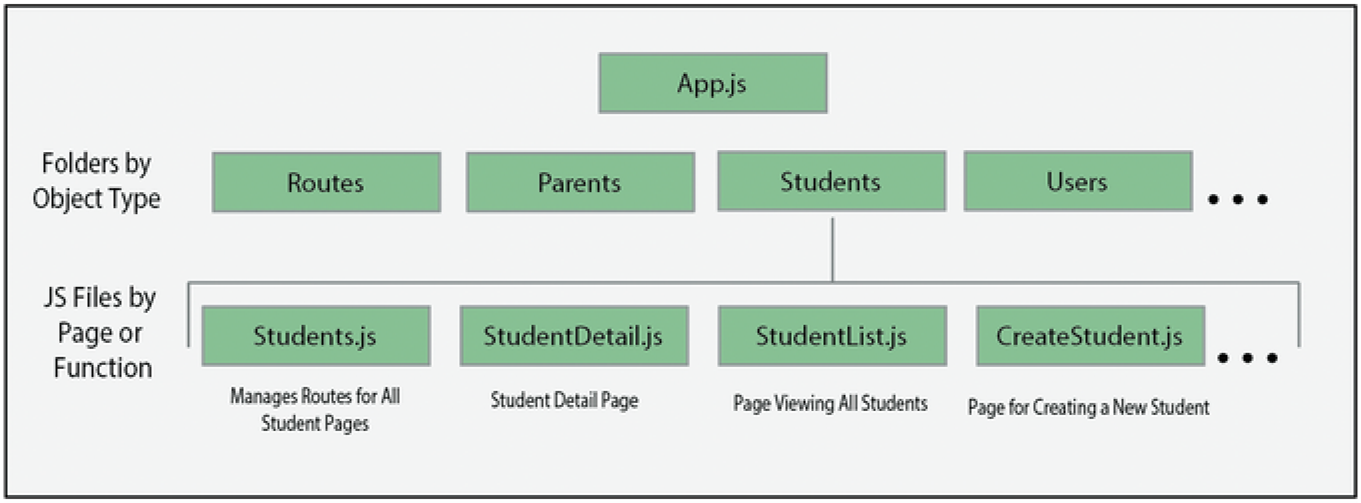
System Architecture Overview



Front End

Our front end uses a Node.js framework is built in React and can be found in the `view/src` folder of the Potato Server Repository here. We use NPM as the package manager (see How to Configure a Development Environment for Start-Up).

The front end follows the following structure, where folders are used to group a particular object and JS files are used to define pages or functions:



Back End

Our back end (`/model/src`) uses Express as the web framework and PostgreSQL for databasing. To interact with the databasing, we utilized TypeORM. There are five core tables in our PostgreSQL database for the app: Users, Students, Students, Routes, and Stops. Details for each table can be found in `model/src/entity/*.ts` while details for each route can be found in `model/src/routes/*.ts`. Below, each specific table key is detailed.

API

In order to make calls to the back end, the front end interacts with React-friendly `src/view/api/axios_wrapper.js`. This wrapper uses Axios to make and return HTTP calls which then calls the routes defined above.

Each asynchronous call for each table is located in its accompanying controller repository (detailed by TypeORM) in `model/src/controller/*.ts`.

Database Table Schema Layout

Users

Column	Type	Required?
uid	number	auto-generated primary key
email	string	yes
fullName	string	yes
address	string	yes
longitude	number	yes
latitude	number	yes
role	string	yes
password	string	yes
students	Relation: Student[]	no
confirmationCode	string	no
studentInfo	Relation: Student	no
runs	Relation: Run[]	

Students

Column	Type	Required?
uid	number	auto-generated primary key
id	string	no
fullName	string	yes
school	Relation: School	yes
route	Relation: Route	no
parentUser	Relation: User	yes
inRangeStops	Relation: Stop[]	no
account	Relation: User	no

Schools

Column	Type	Required?
uid	number	auto-generated primary key
name	string	yes
address	string	yes
longitude	number	yes
latitude	number	yes

`arrivalTime`	`time`	no
`departureTime`	`time`	no
`students`	`Relation: Student[]`	no
`routes`	`Relation: Route[]`	no

Routes

Column	Type	Required?
`uid`	`number`	auto-generated primary key
`name`	`string`	yes
`desciption`	`string`	yes
`longitude`	`number`	yes
`latitutde`	`number`	yes
`polyline`	`string[]`	no
`students`	`Relation: Student[]`	no
`school`	`Relation: School`	yes
`stops`	`Relation: Stop[]`	no
`runs`	`Relation: Run[]`	no

Stop

Column	Type	Required?
`uid`	`number`	auto-generated primary key
`name`	`string`	yes
`location`	`number`	yes
`longitude`	`number`	yes
`latitude`	`number`	yes
`pickupTime`	`time`	yes
`dropoffTime`	`time`	yes
`arrivalIndex`	`number`	no
`route`	`Relation: Route[]`	yes
`inRangeStudents`	`Relation: Student[]`	no

Geo

Column	Type	Required?
`uid`	`number`	auto-generated primary key
`address`	`string`	yes
`longitude`	`number`	yes
`latitude`	`number`	yes
`timeCreated`	`timestamptz`	yes

Runs

Column	Type	Required?
`uid`	`number`	auto-generated primary key
`busNumber`	`string`	yes
`driver`	`Relation: User`	yes
`route`	`Relation: Route`	yes
`timeStarted`	`timestamptz`	yes
`duration`	`number`	no
`timedOut`	`boolean`	no
`direction`	`string`	yes
`longitude`	`number`	no
`latitude`	`number`	no
`TTerroro`	`boolean`	no
`lastFetchTime`	`timestamptz`	no

Emailing

To send bulk email blasts, we chose to use a third-party email provider and create a SMTP transport on the server with Nodemailer. Currently, the production server uses gmail as the email provider. It can be easily switched to a more professional bulk email provider by changing the SMTP credential in `/model/mailConfig.json`. The software also uses RabbitMQ as a queue manager, ready to scale up.

How to Configure a Development Environment

1. Clone the repository

```
git clone https://github.com/cadyzq/PotatoServer.git
```

2. Add secrets

The app uses a couple secrets

- React Google Map API secret

```
cd <path_to_PotatoServer>/view
touch .env
```

Add the secret to the file. It should has the format

```
REACT_APP_GOOGLE_MAPS_API=<the_actual_secret>
```

- JSON Web Token keys

The app uses JWT to authenticate users. We need a pair of keys to make JWTs

```
cd <path_to_PotatoServer>/model
mkdir secrets
cd secrets
openssl genrsa -out jwt_private.key 512
openssl rsa -in jwt_private.key -pubout -out jwt_public.key
```

- Email Provider Credential

```
cd <path_to_PotatoServer>/model
touch mailConfig.json
```

An example `mailConfig.json` should look like

```
{
    "amqp": "amqp://localhost",
    "queue": "email-queue",
    "exchange-type": "fanout",
    "exchange-name": "email-exchange",

    "server": {
      "port": 587,
      "host": "smtp.gmail.com",
      "user": "potatowebservice@gmail.com",
      "password": <credential>
    }
  }
```

3. Install RabbitMQ

RabbitMQ is used as the queue management package for sending emails. Follow the instructions here to install it: `https://www.rabbitmq.com/install-debian.html#apt-quick-start-cloudsmith`

After installing, start the server by

```
sudo systemctl start rabbitmq-server
```

4. Build the entire app

Build the front end

```
cd <path_to_PotatoServer>/view
npm install
npm run build
```

You will need to connect to a database to start the back end. After doing that, change `<path_to_PotatoServer>/model/.env.development` to reflect your configuration.

Make sure your `NODE_ENV` is set to `development`. If not

```
export NODE_ENV=development
```

Start the back end

```
cd <path_to_PotatoServer>/model
npm run install
npm run type-start
```

4. (Optional) To Restart the Front End

Enter the front end folder:

```
cd view
```

Update any dependencies:

```
npm run install
```

Build the front-end:

```
npm start
```