Local Helper

Kun Yu
Zachary Garces
Thomas Westfall
Alexander Yang

Hunter College
CSCI 49900
Fall 2020
Final Presentation

The Product

LocalHelper is a local community-centered user-friendly application to connect people who can offer their time, skills, and items to those in need.

- ➢ **Vision:** Connect people with community resources
- ➢ **Target Audience:** People seeking help
- ➢ **Problem:** People not knowing where to go for help or lacking the means to get paid help
- ➢ **Strategy:** An intuitive, user-friendly application that helps connect people in need with a network of local helpers.
- ➢ **Goals:** A new user successfully navigating the product’s UI and a growing number of satisfied users.

Product Goals

- ➢ User is able to register and login
- ➢ User is able to create/view/edit/delete posts that they have made
- ➢ User is able to easily distinguish between posts requesting help and posts offering help
- ➢ User is able to notify another user of their intention to connect
- ➢ User is able to search for posts that match given keywords
- ➢ User is able to filter posts by language other criteria
- ➢ User is able to communicate with another user

Demo

Let’s see the product!

Technologies Used

- ➢ Flutter
- ➢ Express/Node.js
- ➢ Heroku
- ➢ PostgreSQL
- ➢ Sequelize
- ➢ Postman
- ➢ Discord

Work Distribution

Front-end

- Zach
  - Wrapped backend endpoints
  - Bootstrapped project
- Alex
  - UI Design
  - Bug testing/catching
  - Messaging

Back-end

- Thomas
  - Messaging
  - Bug fixing
  - Error handling
- Kun
  - API Design
  - Database Design
  - Created core API functionality

Challenges

- Flutter cross-platform web compilation is in beta
- Flutter ios complication takes a lot of tuning
- The messaging isn’t live, meaning you’d have to refresh to see new messages

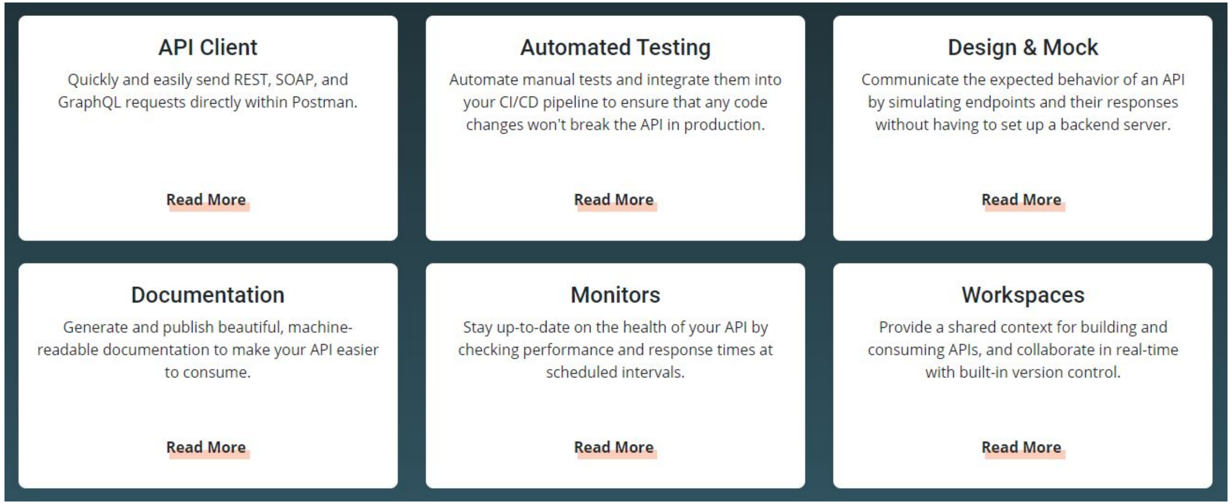
Questions about the project or demo

Technical Presentations

What is Postman?

- Collaboration platform for API development
- Its features simplifies each step of building an API

POSTMAN



API Client
Quickly and easily send REST, SOAP, and GraphQL requests directly within Postman.
Read More

Automated Testing
Automate manual tests and integrate them into your CI/CD pipeline to ensure that any code changes won't break the API in production.
Read More

Design & Mock
Communicate the expected behavior of an API by simulating endpoints and their responses without having to set up a backend server.
Read More

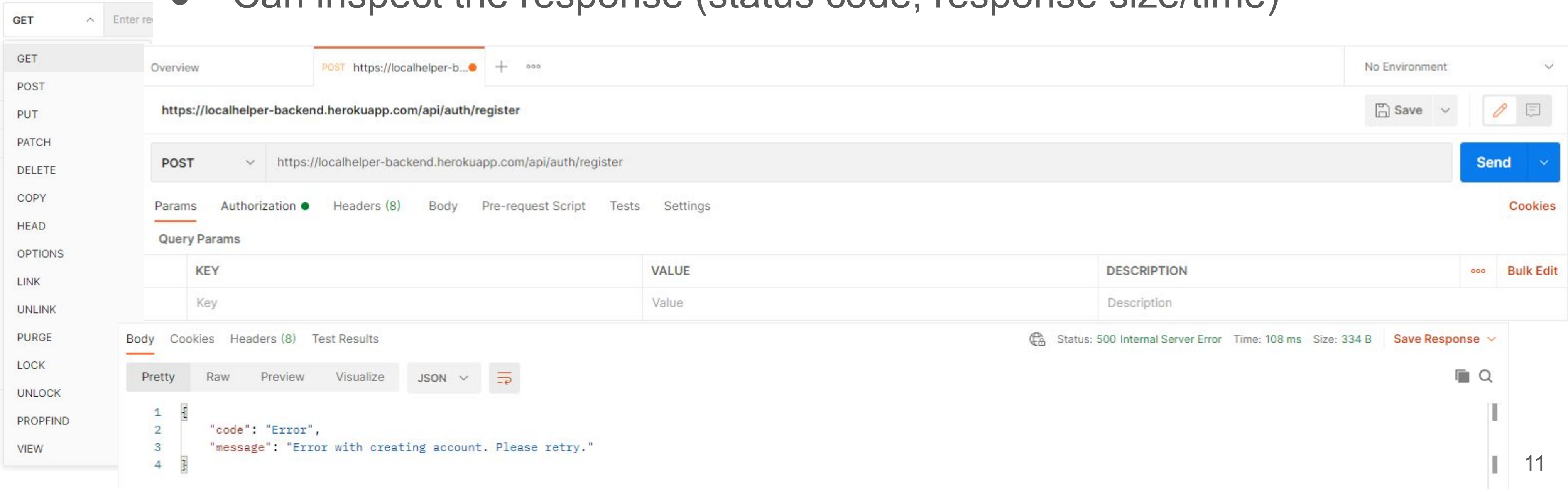
Documentation
Generate and publish beautiful, machine-readable documentation to make your API easier to consume.
Read More

Monitors
Stay up-to-date on the health of your API by checking performance and response times at scheduled intervals.
Read More

Workspaces
Provide a shared context for building and consuming APIs, and collaborate in real-time with built-in version control.
Read More

Why use Postman? (Testing APIs)

- Simple UI for testing APIs
- Can make many different requests
- Can inspect the response (status code, response size/time)



https://localhelper-backend.herokuapp.com/api/auth/register

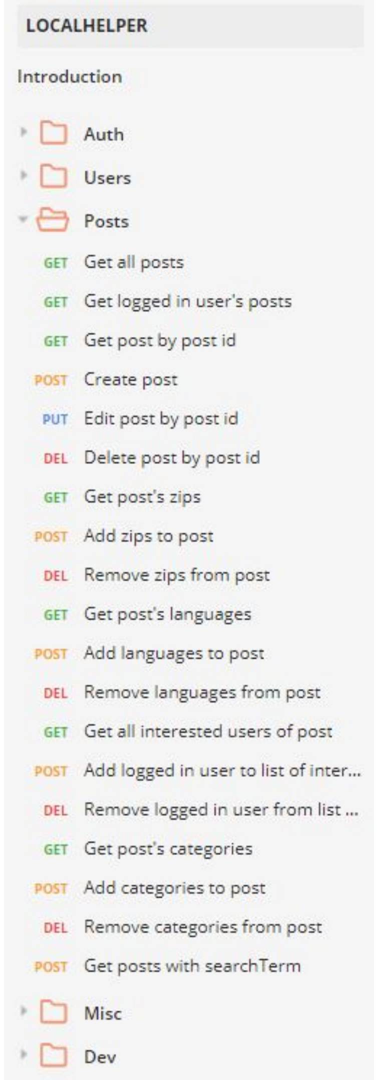
POST https://localhelper-backend.herokuapp.com/api/auth/register

Status: 500 Internal Server Error Time: 108 ms Size: 334 B

```
{
    "code": "Error",
    "message": "Error with creating account. Please retry."
}
```

Why use Postman? (Documentation)

- Provides a platform to create clean looking documentations for APIs
- Allows you to provide example requests and example responses
- Allows for organizational groupings of APIs



LOCALHELPER

Introduction

Auth

Users

Posts

GET Get all posts

GET Get logged in user's posts

GET Get post by post id

POST Create post

PUT Edit post by post id

DEL Delete post by post id

GET Get post's zips

POST Add zips to post

DEL Remove zips from post

GET Get post's languages

POST Add languages to post

DEL Remove languages from post

GET Get all interested users of post

POST Add logged in user to list of inter...

DEL Remove logged in user from list ...

GET Get post's categories

POST Add categories to post

DEL Remove categories from post

POST Get posts with searchTerm

Misc

Dev

Auth

This folder contains endpoints related to authentication, namely registration and login.

POST Register

http://localhelper-backend.herokuapp.com/api/auth/register

Creates a user object and registers the user

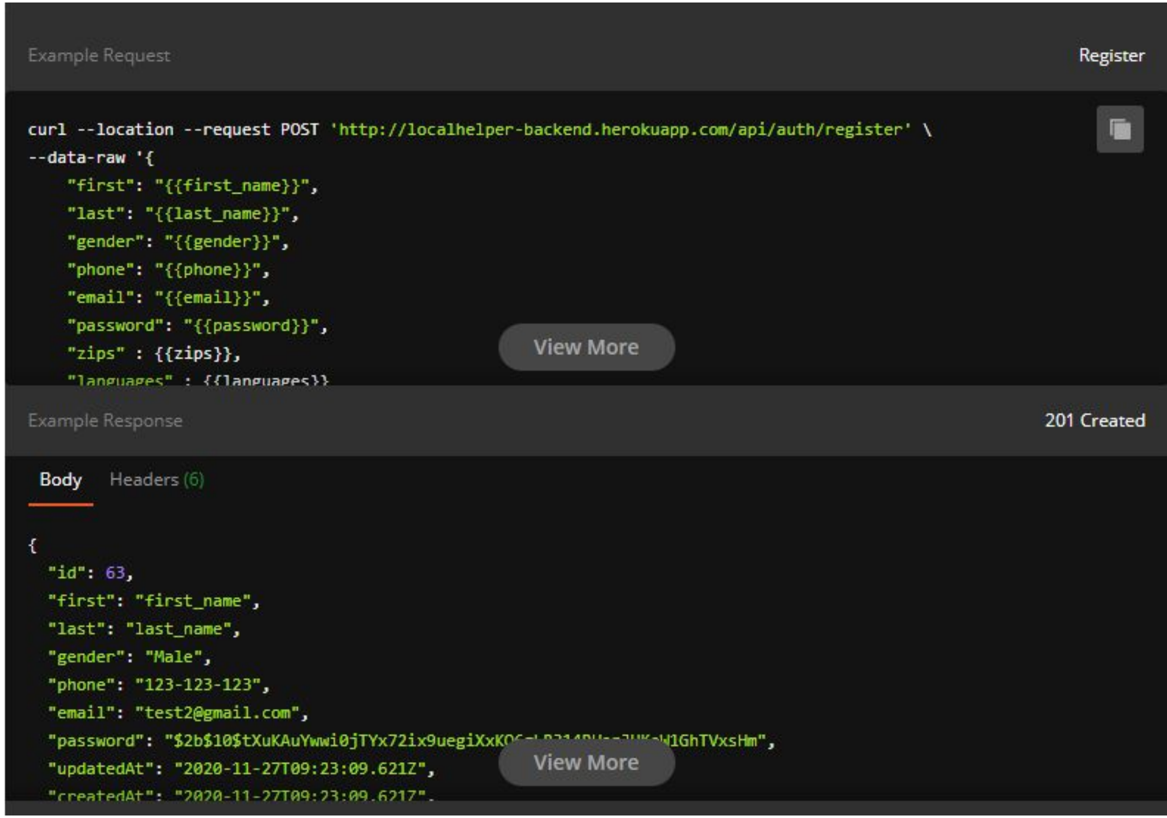
Returns created user object.

Body

- **first** `required` - A string to denote user's first name
- **last** `required` - A string to denote user's last name
- **gender** `required` - A string to denote user's gender
- **phone** `required` - A string to denote user's phone number
- **email** `required` - A string to denote user's email. **Must be unique**
- **password** `required` - A string to denote user's password.
- **zips** `optional` - An array of strings denoting zips.
- **languages** `optional` - An array of strings denoting languages.

BODY raw

```
{
    "first": "{{first_name}}",
    "last": "{{last_name}}",
    "gender": "{{gender}}",
    "phone": "{{phone}}",
    "email": "{{email}}",
    "password": "{{password}}",
    "zips" : {{zips}},
    "languages" : {{languages}}
}
```



Example Request — Register

```
curl --location --request POST 'http://localhelper-backend.herokuapp.com/api/auth/register' \
--data-raw '{
    "first": "{{first_name}}",
    "last": "{{last_name}}",
    "gender": "{{gender}}",
    "phone": "{{phone}}",
    "email": "{{email}}",
    "password": "{{password}}",
    "zips" : {{zips}},
    "languages" : {{languages}}
```

Example Response — 201 Created

Body Headers (6)

```
{
  "id": 63,
  "first": "first_name",
  "last": "last_name",
  "gender": "Male",
  "phone": "123-123-123",
  "email": "test2@gmail.com",
  "password": "$2b$10$tXuKAuYwwi0jTYx72ix9uegiXxK[illegible]W1GhTVxsHm",
  "updatedAt": "2020-11-27T09:23:09.621Z",
  "createdAt": "2020-11-27T09:23:09.621Z",
```

How we used Postman

- Used postman to test endpoints by mimicking network calls from front-end
- Used postman to test latency of endpoints to make sure the APIs are responsive
- Used postman to generate documentation for back-end API

Sequelize/ORMs

What is an ORM?

- “Object-Relational Mapping (ORM) is a technique that lets you query and manipulate data from a database using an object-oriented paradigm.”
- Don’t need to use SQL anymore, you interact directly with an object in the same language you're using.”

How it works?

- **M**apping between **O**bjects(as in object-orientated programing) and **R**elational databases
- Virtual layer between source code and database
- Translates objects and functions into tables and queries that directly manipulate the database

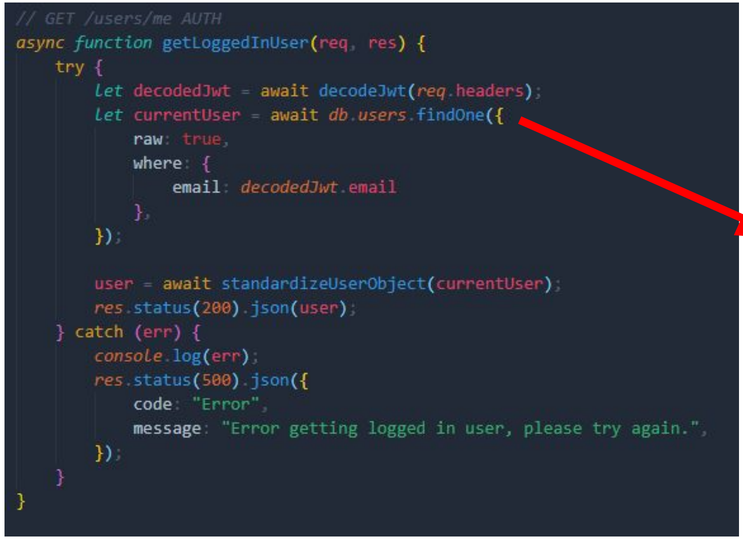
Pros

- Don’t need to learn/write SQL!
- Someone’s probably already done it better and faster.
- Sanitized, prepared statements are less of a security risk
- Abstracts the DB system, so can change it whenever

Cons

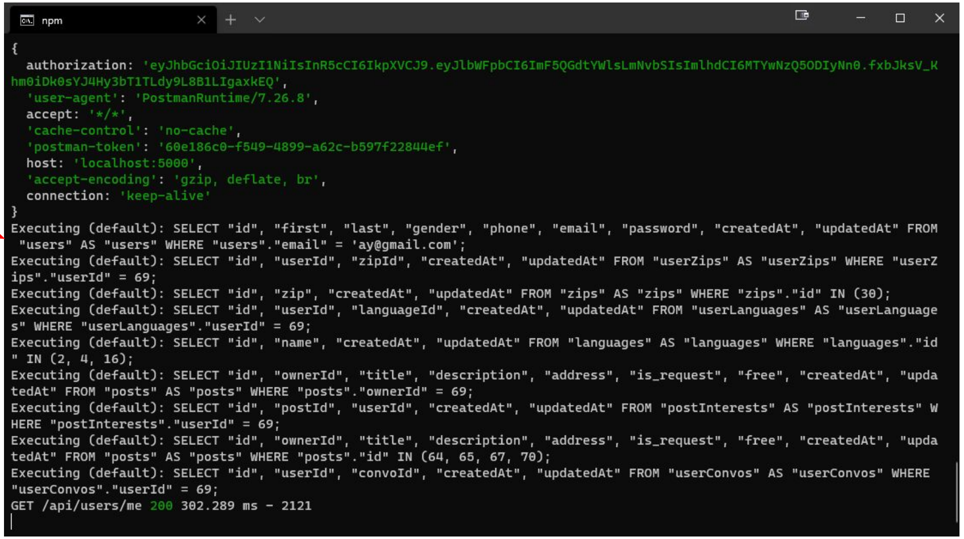
- Have to learn a new technology, ORM libraries are not lightweight tools
- Documentation can be lackluster
- Performance is usually ok but SQL master can write better queries
- Abstracts the DB which hides runtimes and logic

Sequelize



```
// GET /users/me AUTH
async function getLoggedInUser(req, res) {
    try {
        let decodedJwt = await decodeJwt(req.headers);
        let currentUser = await db.users.findOne({
            raw: true,
            where: {
                email: decodedJwt.email
            },
        });

        user = await standardizeUserObject(currentUser);
        res.status(200).json(user);
    } catch (err) {
        console.log(err);
        res.status(500).json({
            code: "Error",
            message: "Error getting logged in user, please try again.",
        });
    }
}
```



```
{
  authorization: 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6ImF5QGdtYWlsLmNvbSIsImlhdCI6MTYwNzQ5ODIyNn0.fxbJksV_K
hm0iDk0sYJ4Hy3bT1TLdy9L8B1LIgaxkEQ',
  'user-agent': 'PostmanRuntime/7.26.8',
  accept: '*/*',
  'cache-control': 'no-cache',
  'postman-token': '60e186c0-f549-4899-a62c-b597f22844ef',
  host: 'localhost:5000',
  'accept-encoding': 'gzip, deflate, br',
  connection: 'keep-alive'
}
Executing (default): SELECT "id", "first", "last", "gender", "phone", "email", "password", "createdAt", "updatedAt" FROM
 "users" AS "users" WHERE "users"."email" = 'ay@gmail.com';
Executing (default): SELECT "id", "userId", "zipId", "createdAt", "updatedAt" FROM "userZips" AS "userZips" WHERE "userZ
ips"."userId" = 69;
Executing (default): SELECT "id", "zip", "createdAt", "updatedAt" FROM "zips" AS "zips" WHERE "zips"."id" IN (30);
Executing (default): SELECT "id", "userId", "languageId", "createdAt", "updatedAt" FROM "userLanguages" AS "userLanguage
s" WHERE "userLanguages"."userId" = 69;
Executing (default): SELECT "id", "name", "createdAt", "updatedAt" FROM "languages" AS "languages" WHERE "languages"."id
" IN (2, 4, 16);
Executing (default): SELECT "id", "ownerId", "title", "description", "address", "is_request", "free", "createdAt", "upda
tedAt" FROM "posts" AS "posts" WHERE "posts"."ownerId" = 69;
Executing (default): SELECT "id", "postId", "userId", "createdAt", "updatedAt" FROM "postInterests" AS "postInterests" W
HERE "postInterests"."userId" = 69;
Executing (default): SELECT "id", "ownerId", "title", "description", "address", "is_request", "free", "createdAt", "upda
tedAt" FROM "posts" AS "posts" WHERE "posts"."id" IN (64, 65, 67, 70);
Executing (default): SELECT "id", "userId", "convoId", "createdAt", "updatedAt" FROM "userConvos" AS "userConvos" WHERE
"userConvos"."userId" = 69;
GET /api/users/me 200 302.289 ms - 2121
```

Thanks!

Any questions?