Sayyara

Development Process

Introduction

The Problem

Your vehicle has broken down - *You need a repair ASAP!*

What are the options for a *price-conscious* vehicle owner?

- Search for nearby automotive repair shops - start making phone calls
- Sit through countless automated menus hoping to talk to a human being
- Repeat your problem and vehicle information every time you get through
- Receive a quote, write it down, rinse and repeat…
- Finally, go with the best price

*Potentially waste **hours** of valuable time*

The Solution

Centralize and *streamline* the automotive repair process

Sayyara functions as your hub for vehicle repair

Without ever leaving the app, the user can:

- Browse repair shops and view their services
- Request quotes for their issue from any participating shop
- Monitor the status of their quotes - All price estimates are consolidated in one place
- Schedule an appointment directly after accepting an estimate

*No more waiting on hold - Shops **compete** for your business!*

Customers Benefit

- Save time, stress and hassle
- Every part of the process is in one place.
- See all available options at a glance
- Easily compare quote estimates for the best deal

Shop Owners Benefit

- See your upcoming appointments on the homepage
- Give price estimates with ease
- See relevant client info, including vehicle specifications and contact information
- Get right down to business, cut out unnecessary verbal communication

Demo

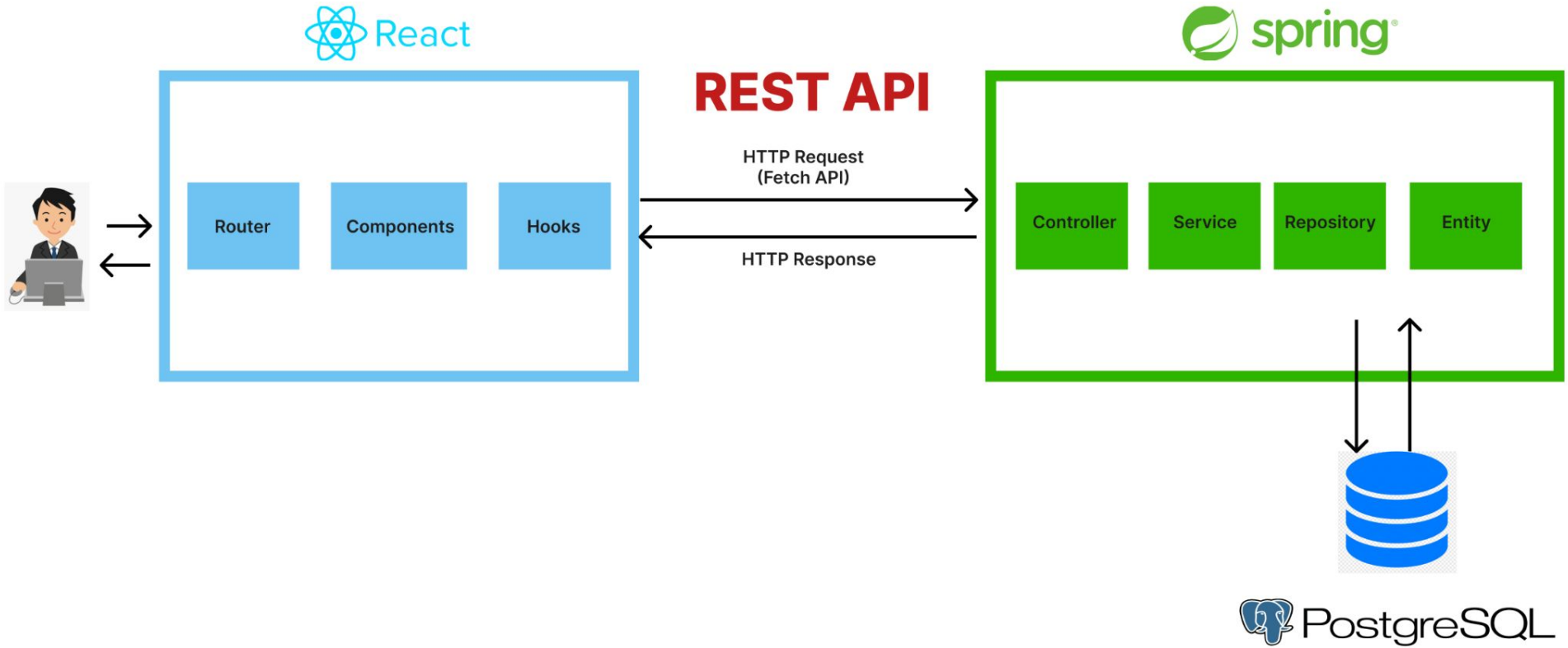
https://www.youtube.com/watch?v=UtZrf9WLq-E

Technologies Used

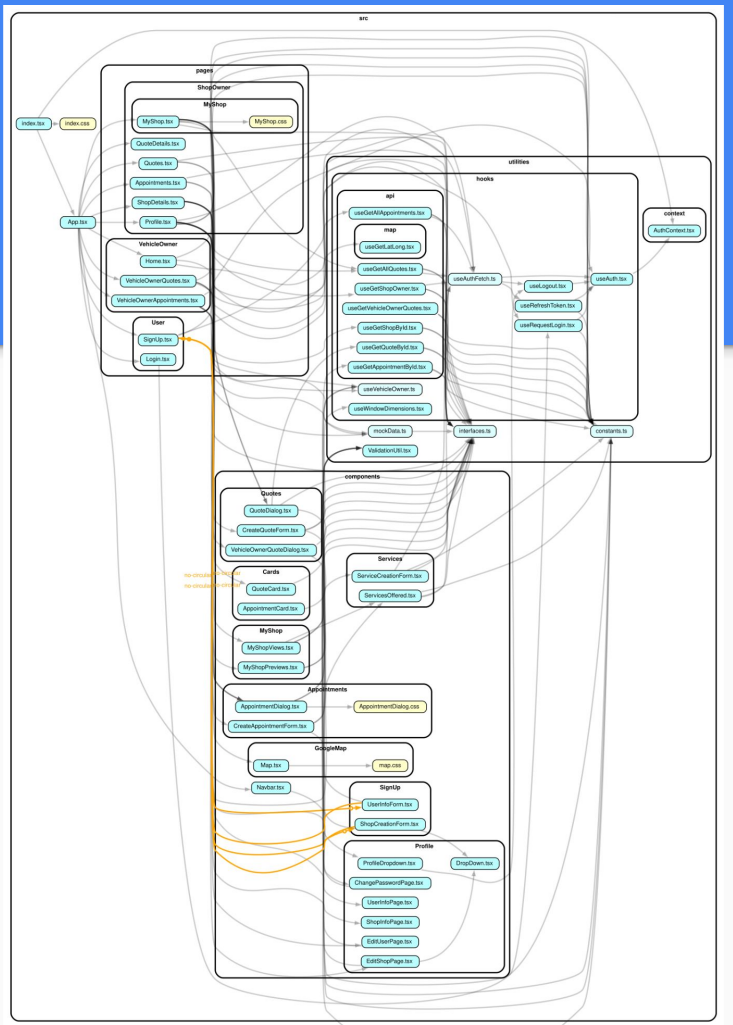
Frontend

Backend

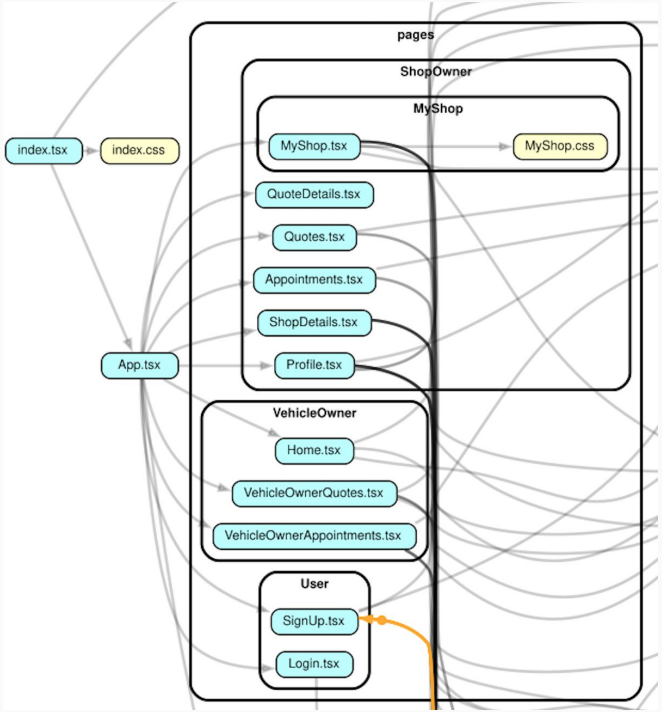
End to End Architecture Diagram



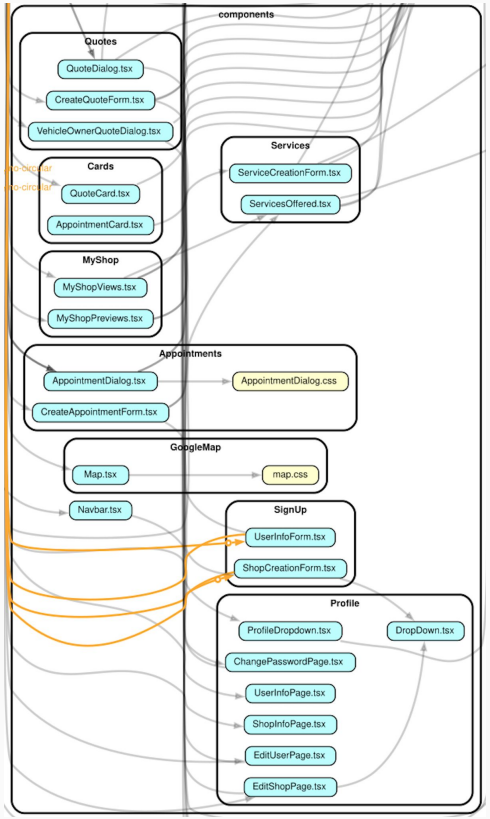
Frontend Architecture



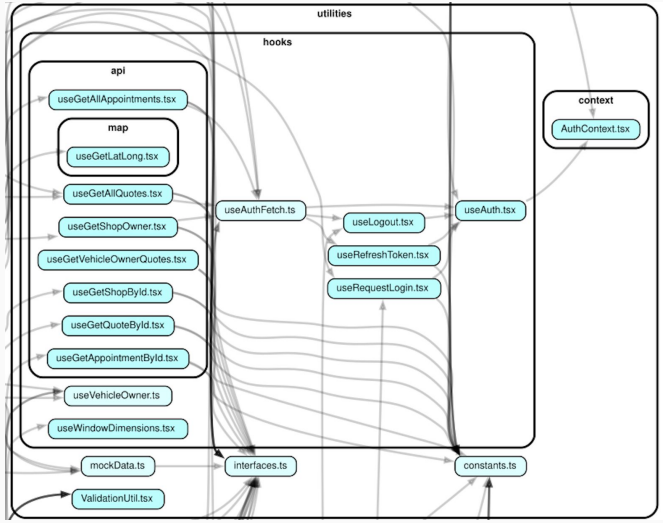
Pages



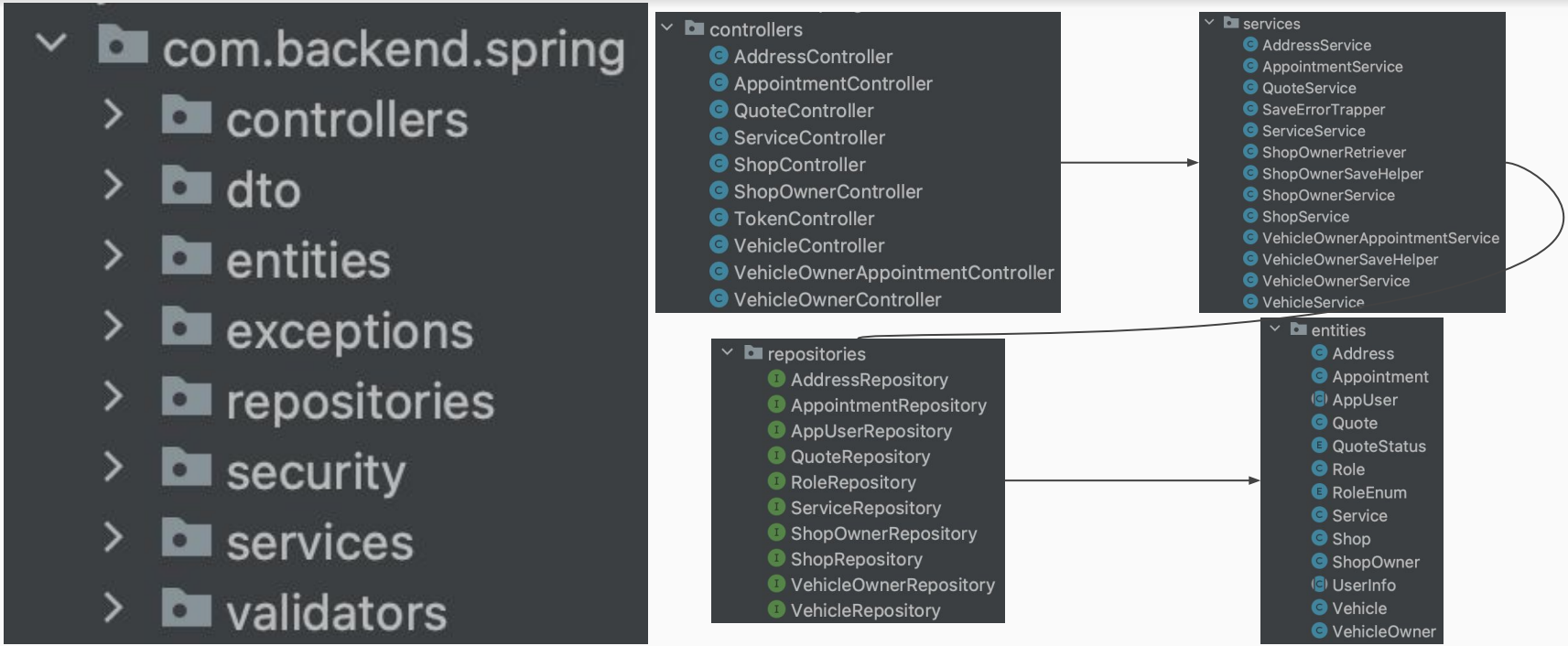
Components



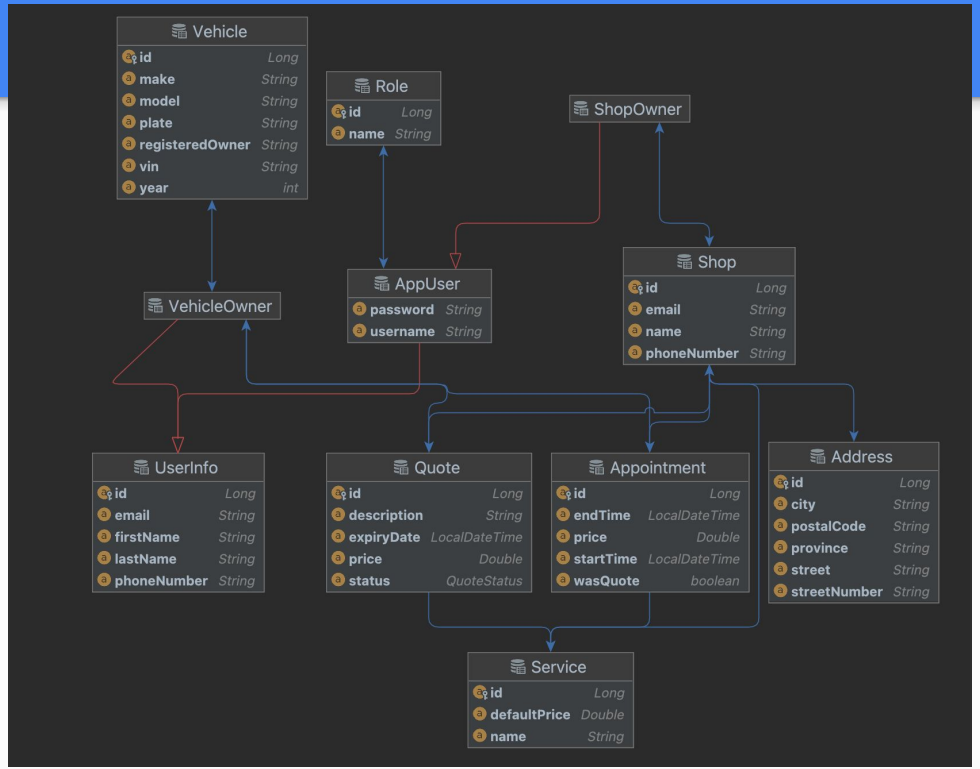
Utilities



Backend Structure



Database Tables



Clean Code Examples

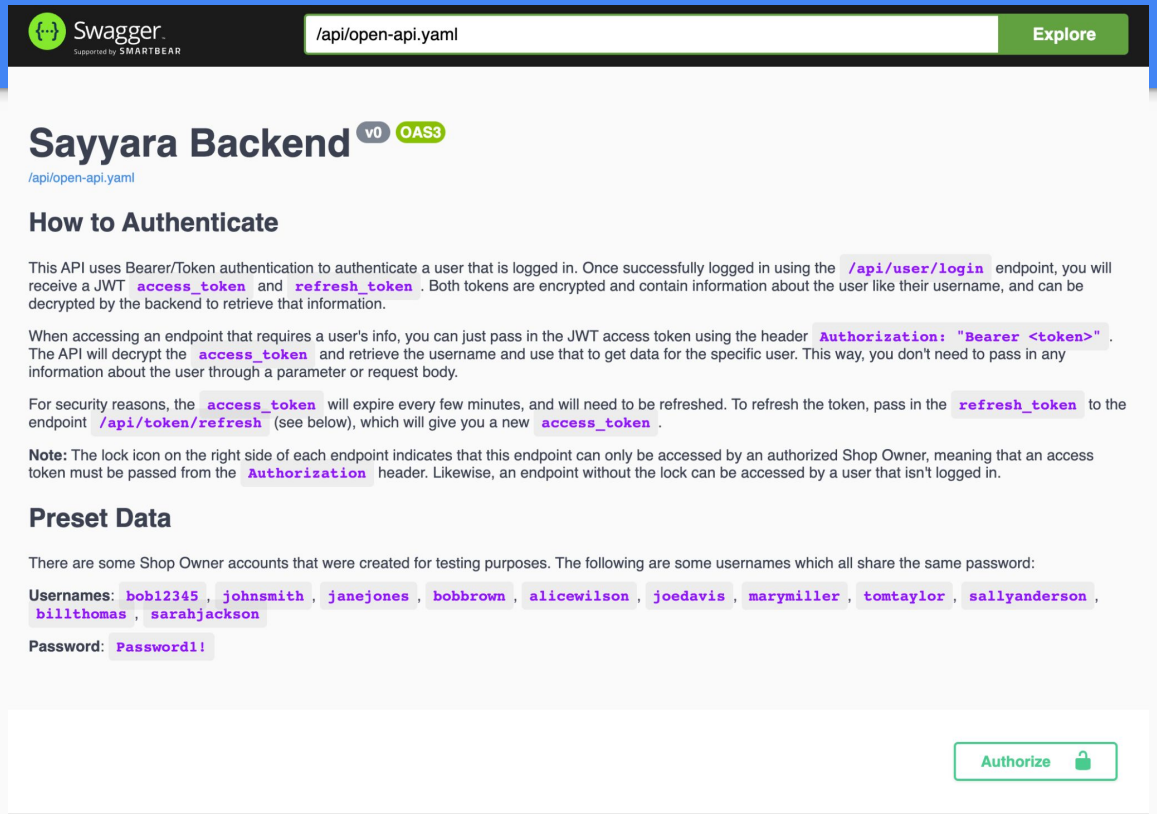
Frontend

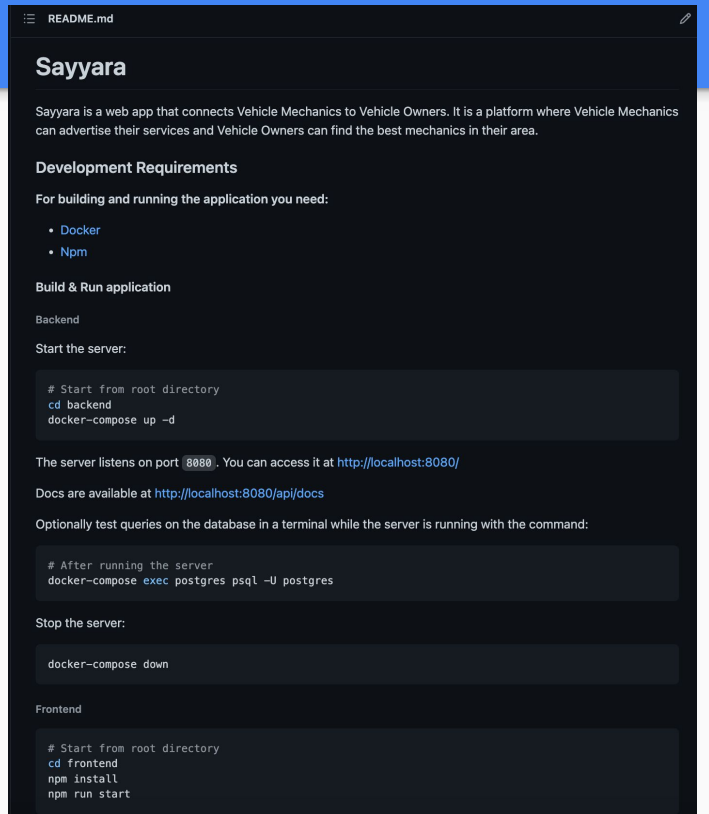
```tsx
/*
    This hook returns a function that is used EXACTLY as a `fetch` call.

    `authFetch` is the function returned, and it does 2 things:
        1. It adds the Authorization header to the request with the access_token
        2. It refreshes the token and calls fetch again if the request fails with a 401 error

    ======= Usage =======
    const { authFetch } = useAuthFetch();

    authFetch("https://example.com/api/v1/endpoint", {
        method: "GET",
        headers: {
            "Content-Type": "application/json"
        }
    });

    Return type is the same as the fetch call, so can be used with either `await` or `.then()`
 */
const useAuthFetch = () => {
    let { auth: token } = useAuth();
    const refresh = useRefreshToken();
    const { logout } = useLogout();

    const authFetch = async (url: string, options: RequestInit = {}) => {...}

    return { authFetch }
}
```

Backend

```java
/**
 * This class parses an authentication header and retrieves the shop owner object from it.
 */
@Service
@RequiredArgsConstructor
class ShopOwnerRetriever {

    private final ShopOwnerRepository shopOwnerRepository;

    /**
     * Parses the authentication header and retrieves the shop owner object from it.
     *
     * @param authorization Authorization header received from an HTTP request
     * @return Shop owner object associated with the username in the authorization header
     * @throws InvalidDataException If the authorization header is invalid
     */
    ShopOwner getShopOwner(String authorization) throws InvalidDataException {
        String username = new AuthHeaderParser(authorization).getUsername();

        return shopOwnerRepository.findByUsername(username).orElseThrow(() ->
                new DataNotFoundException("Shop Owner with username " + username + " not found"));
    }

    /**
     * Parses the authentication header and retrieves the shop object from it.
     *
     * @param authorization Authorization header received from an HTTP request
     * @return Shop object associated with the username in the authorization header
     * @throws InvalidDataException If the authorization header is invalid
     */
    Shop getShop(String authorization) throws InvalidDataException {
        return getShopOwner(authorization).getShop();
    }
}
```

Documentation





Application Access

Deployed Links

Website

Vercel

https://sayyara-web.vercel.app/

API Docs

Railway

https://sayyara-backend.up.railway.app/api/docs

CI/CD

Railway

PostgreSQL

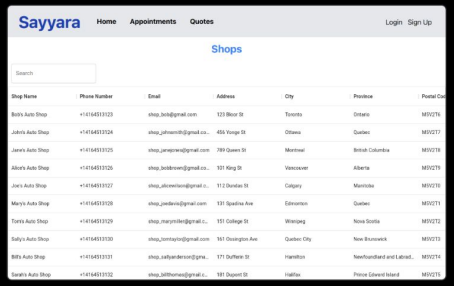
Created 2022-12-06

SpringBoot

sayyara-backend.up.railway....

2 hours ago via Up

Vercel



DEPLOYMENT

sayyara-mbxuyb533-sayyaradev-gmailcom.vercel.app

DOMAINS

sayyara-web.vercel.app +1

STATUS

Ready

CREATED

2h ago by sayyaradev-gmailcom

BRANCH

main

Update Sayyara website link in README

GitHub Actions

deploy-frontend.yaml

```yaml
name: Deploy Frontend to Vercel

env:
  VERCEL_ORG_ID: ${{ secrets.VERCEL_ORG_ID }}
  VERCEL_PROJECT_ID: ${{ secrets.VERCEL_PROJECT_ID }}

on:
  push:
    paths:
      - 'frontend/**'
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3

      - name: Install Vercel CLI
        run: npm install --global vercel@latest

      - name: Pull Vercel Environment Information
        run: vercel pull --yes --environment=production --token=${{ secrets.VERCEL_TOKEN }}

      - name: Build Project Artifacts
        run: vercel build --prod --token=${{ secrets.VERCEL_TOKEN }}

      - name: Deploy Project Artifacts to Vercel
        run: vercel deploy --prebuilt --prod --token=${{ secrets.VERCEL_TOKEN }}
```

Local Development

Backend: 

Frontend: 

README.md

Sayyara

Sayyara is a web app that connects Vehicle Mechanics to Vehicle Owners. It is a platform where Vehicle Mechanics can advertise their services and Vehicle Owners can find the best mechanics in their area.

Development Requirements

For building and running the application you need:

- Docker
- Npm

Build & Run application

Backend

Start the server:

```
# Start from root directory
cd backend
docker-compose up -d
```

The server listens on port `8080`. You can access it at http://localhost:8080/

Docs are available at http://localhost:8080/api/docs

Optionally test queries on the database in a terminal while the server is running with the command:

```
# After running the server
docker-compose exec postgres psql -U postgres
```

Stop the server:

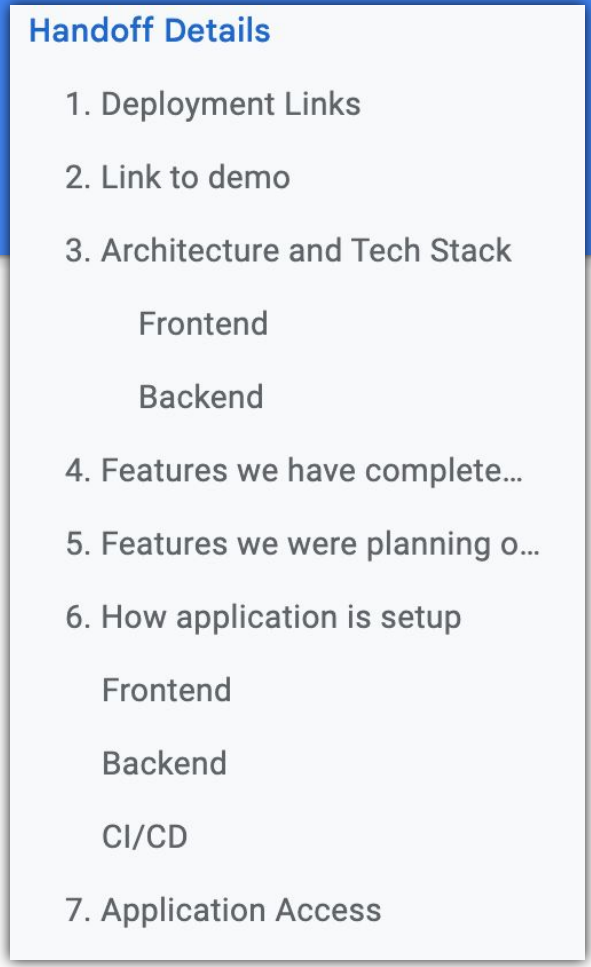
```
docker-compose down
```

Frontend

```
# Start from root directory
cd frontend
npm install
npm run start
```

Handoff to Partner

- Prepared an “*everything you need to know to continue development*” document
- Custom **project email** with access to deployment
- Custom GitHub account



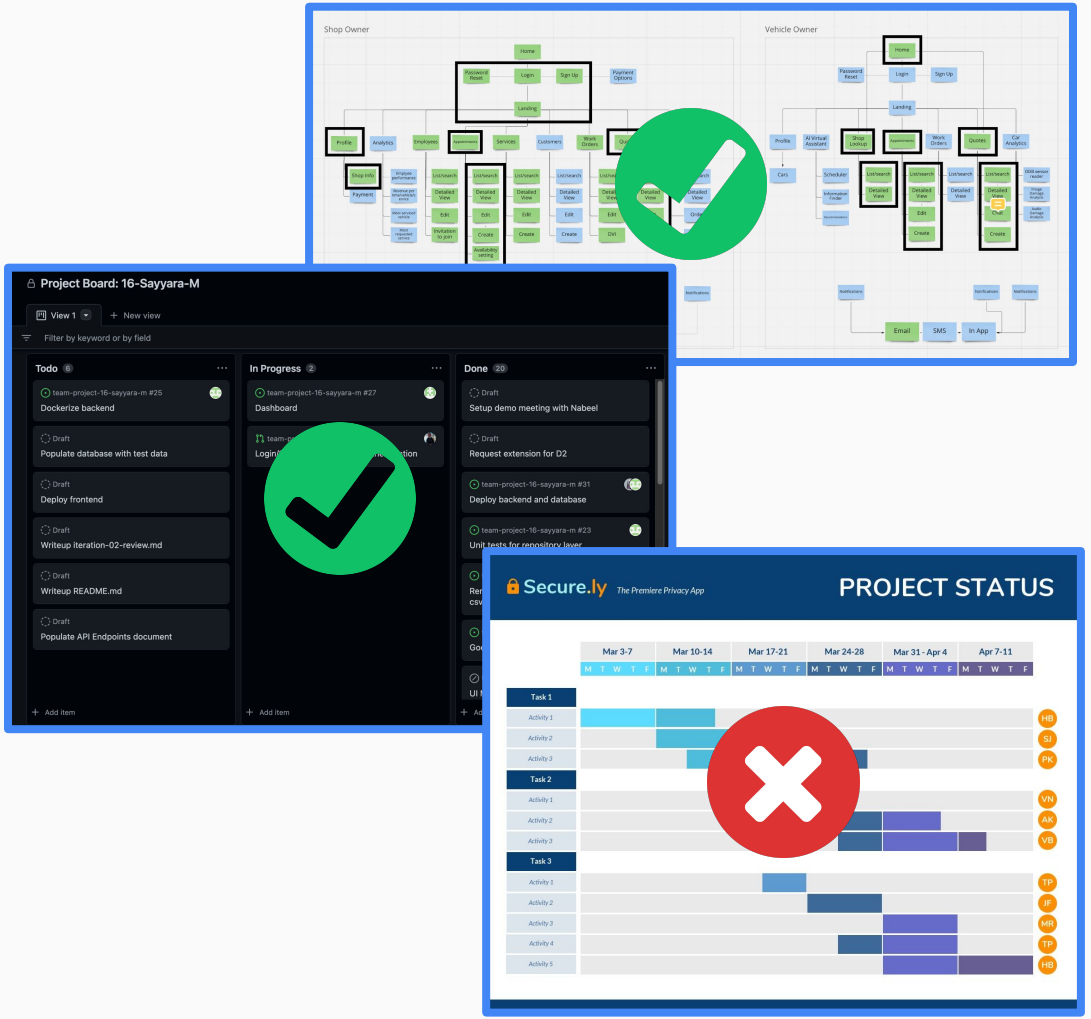
Key Learnings

Key Learnings

- Mockups
- Development Roadmap
- Standup Agenda

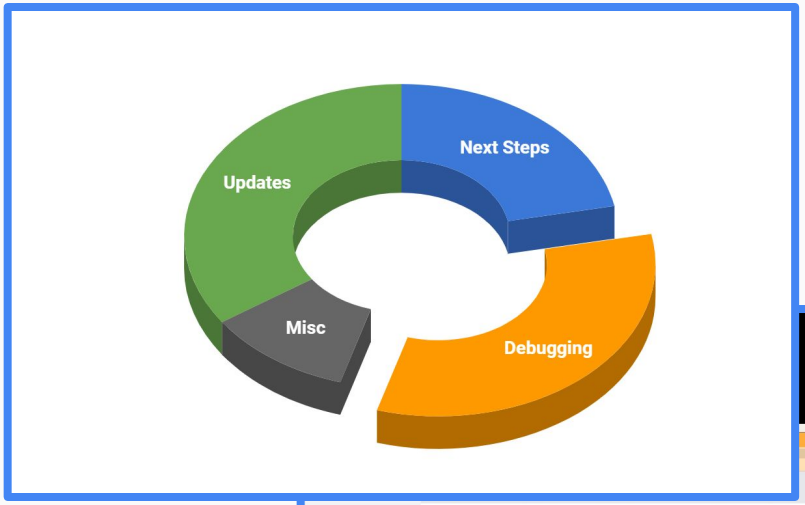
Key Learnings

- Mockups
- Development Roadmap
- Standup Agenda



Key Learnings

- Mockups
- Development Roadmap
- Standup Agenda



Key Learnings

- Mockups
- Development Roadmap
- Standup Agenda

Thanks!

Sayyara