

# 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

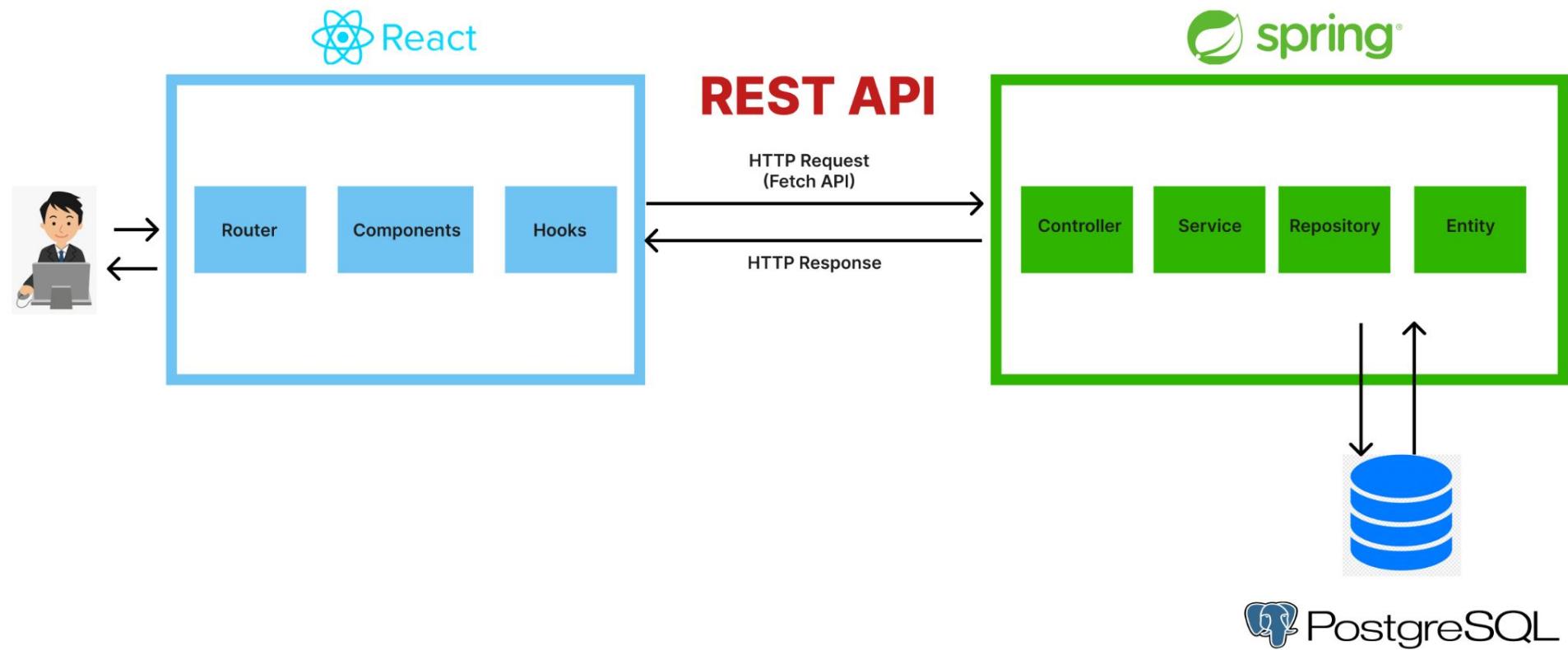
# Frontend



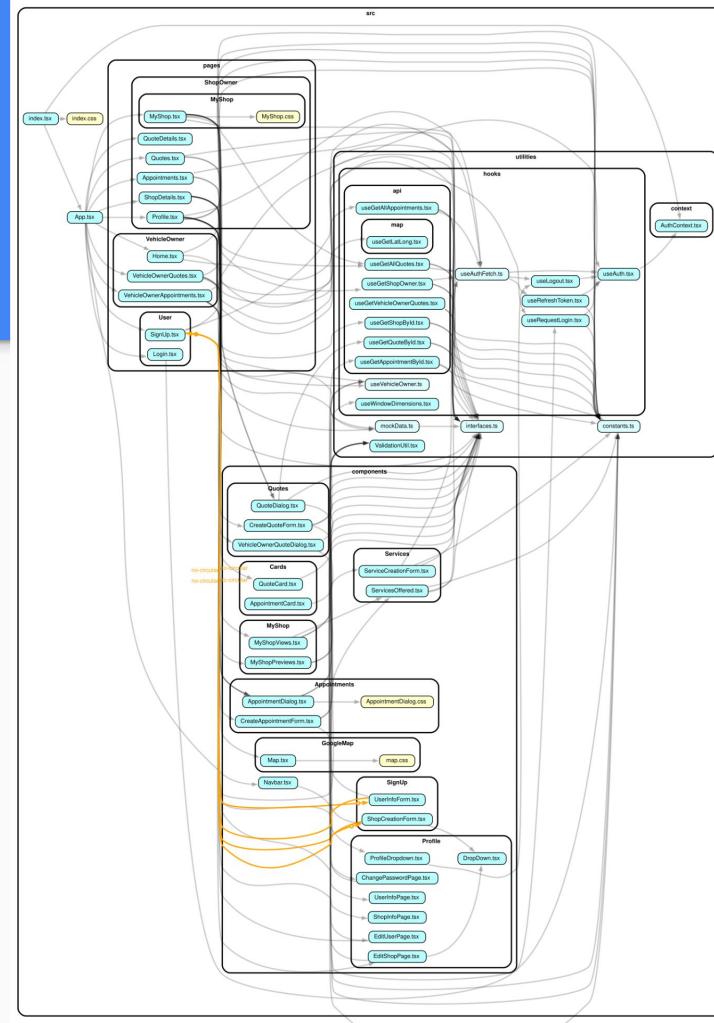
# Backend



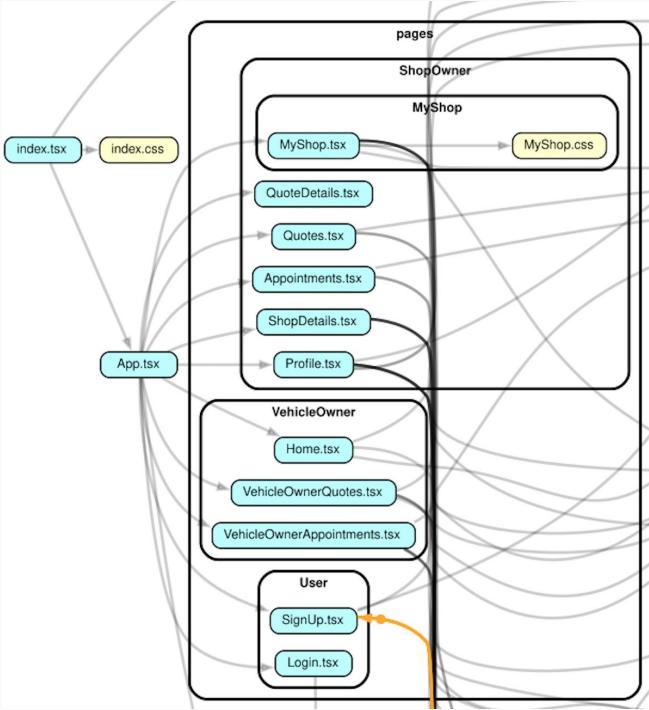
# End to End Architecture Diagram



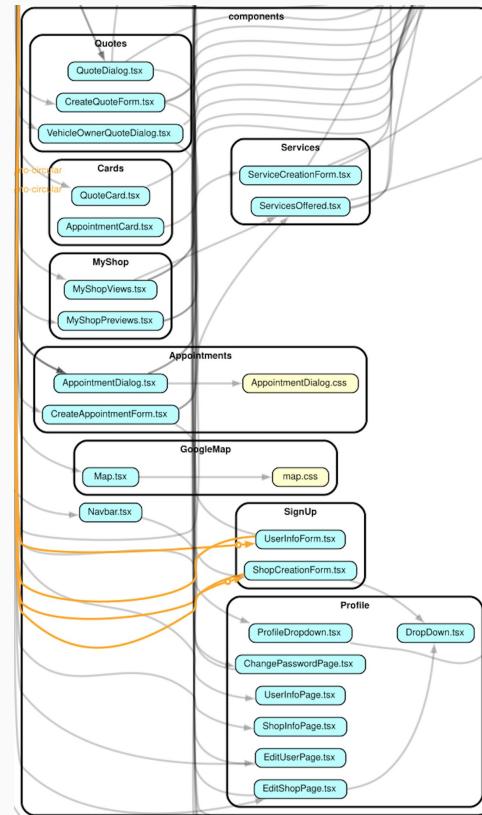
# Frontend Architecture



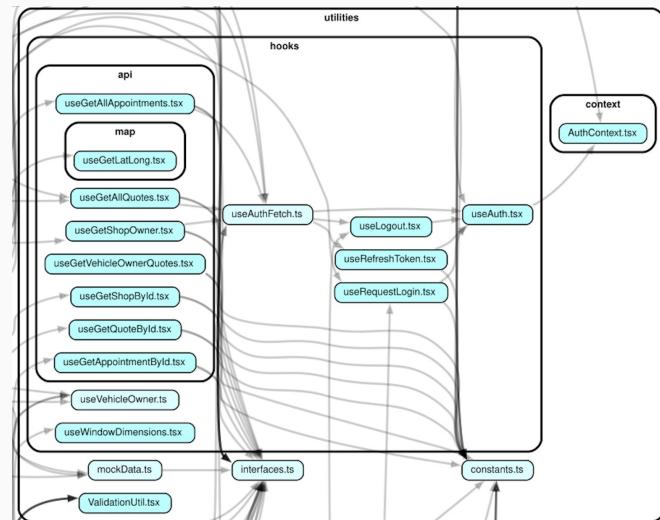
# Pages



# Components



# Utilities



# Backend Structure

```
✓ com.backend.spring
  > controllers
  > dto
  > entities
  > exceptions
  > repositories
  > security
  > services
  > validators
```

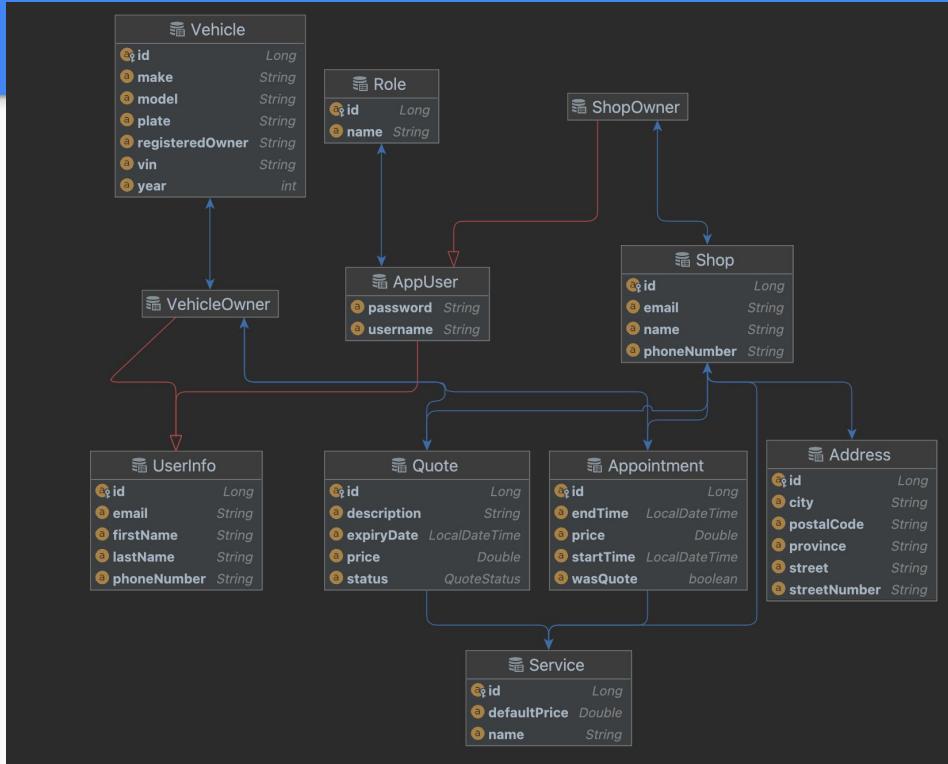
```
✓ controllers
  C AddressController
  C AppointmentController
  C QuoteController
  C ServiceController
  C ShopController
  C ShopOwnerController
  C TokenController
  C VehicleController
  C VehicleOwnerAppointmentController
  C VehicleOwnerController
```

```
✓ repositories
  I AddressRepository
  I AppointmentRepository
  I AppUserRepository
  I QuoteRepository
  I RoleRepository
  I ServiceRepository
  I ShopOwnerRepository
  I ShopRepository
  I VehicleOwnerRepository
  I VehicleRepository
```

```
✓ services
  C AddressService
  C AppointmentService
  C QuoteService
  C SaveErrorTrapper
  C ServiceService
  C ShopOwnerRetriever
  C ShopOwnerSaveHelper
  C ShopOwnerService
  C ShopService
  C VehicleOwnerAppointmentService
  C VehicleOwnerSaveHelper
  C VehicleOwnerService
  C VehicleService
```

```
✓ entities
  C Address
  C Appointment
  C AppUser
  C Quote
  E QuoteStatus
  C Role
  E RoleEnum
  C Service
  C Shop
  C ShopOwner
  C UserInfo
  C Vehicle
  C VehicleOwner
```

# Database Tables



# Clean Code Examples

## Frontend

```
/*
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

```
/**
 * 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



The Swagger UI interface shows the path `/api/open-api.yaml` and a prominent green **Explore** button.

## Sayyara Backend v0 OAS3

</api/open-api.yaml>

### How to Authenticate

This API uses Bearer/Token authentication to authenticate a user that is logged in. Once successfully logged in using the `/api/user/login` endpoint, you will receive a JWT `access_token` and `refresh_token`. Both tokens are encrypted and contain information about the user like their username, and can be decrypted by the backend to retrieve that information.

When accessing an endpoint that requires a user's info, you can just pass in the JWT access token using the header `Authorization: "Bearer <token>"`. The API will decrypt the `access_token` and retrieve the username and use that to get data for the specific user. This way, you don't need to pass in any information about the user through a parameter or request body.

For security reasons, the `access_token` will expire every few minutes, and will need to be refreshed. To refresh the token, pass in the `refresh_token` to the endpoint `/api/token/refresh` (see below), which will give you a new `access_token`.

**Note:** The lock icon on the right side of each endpoint indicates that this endpoint can only be accessed by an authorized Shop Owner, meaning that an access token must be passed from the `Authorization` header. Likewise, an endpoint without the lock can be accessed by a user that isn't logged in.

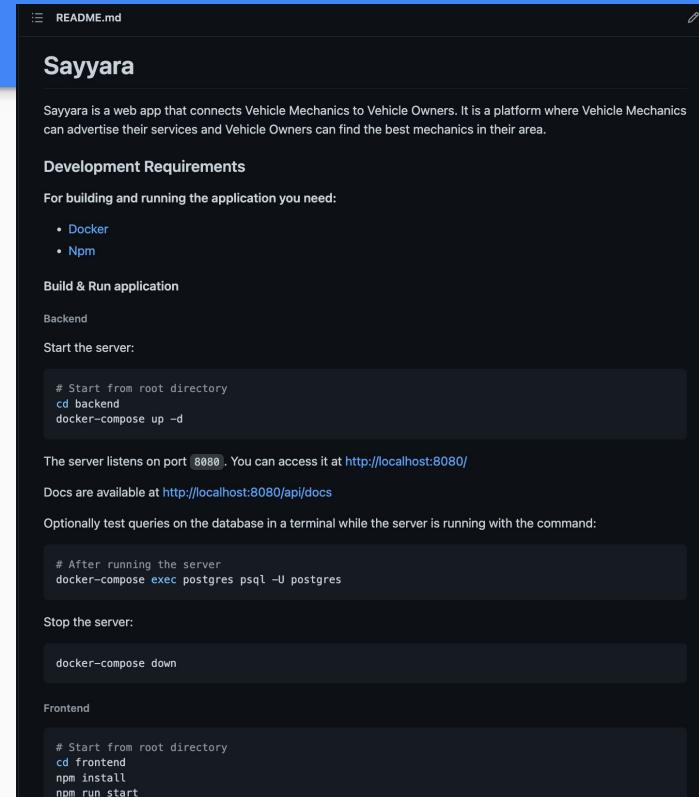
### Preset Data

There are some Shop Owner accounts that were created for testing purposes. The following are some usernames which all share the same password:

**Usernames:** `bob12345`, `johnsmith`, `janejones`, `bobbrown`, `alicewilson`, `joedavis`, `marymiller`, `tomtaylor`, `sallyanderson`, `billthomas`, `sarahjackson`

**Password:** `Password1!`

**Authorize** 



The README.md page for Sayyara contains the following sections:

- 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

# Application Access

# Deployed Links

**Website**



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

**API Docs**



**Railway**

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

# CI/CD



The screenshot shows the Sayyara app deployment on Vercel. The deployment URL is sayyara-mbxuyb533-sayyaradev-gmailcom.vercel.app. The deployment status is 'Ready' and it was created 2h ago by sayyaradev-gmailcom. The branch is 'main'. A note at the bottom says 'Update Sayyara website link in README'.

```
deploy-frontend.yaml
1 name: Deploy Frontend to Vercel
2
3 env:
4   VERCEL_ORG_ID: ${{ secrets.VERCEL_ORG_ID }}
5   VERCER_PROJECT_ID: ${{ secrets.VERCEL_PROJECT_ID }}
6
7 on:
8   push:
9     paths:
10       - 'frontend/**'
11     branches:
12       - main
13
14 jobs:
15   deploy:
16     runs-on: ubuntu-latest
17
18   steps:
19     - uses: actions/checkout@v3
20
21     - name: Install Vercel CLI
22       run: npm install --global vercel@latest
23
24     - name: Pull Vercel Environment Information
25       run: vercel pull --yes --environment=production --token=${{ secrets.VERCEL_TOKEN }}
26
27     - name: Build Project Artifacts
28       run: vercel build --prod --token=${{ secrets.VERCEL_TOKEN }}
29
30     - name: Deploy Project Artifacts to Vercel
31       run: vercel deploy --prebuilt --prod --token=${{ secrets.VERCEL_TOKEN }}
```

## GitHub Actions

# 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

## Handoff Details

1. Deployment Links
2. Link to demo
3. Architecture and Tech Stack
  - Frontend
  - Backend
4. Features we have complete...
5. Features we were planning o...
6. How application is setup
  - Frontend
  - Backend
  - CI/CD
7. Application Access

# Key Learnings

# Key Learnings



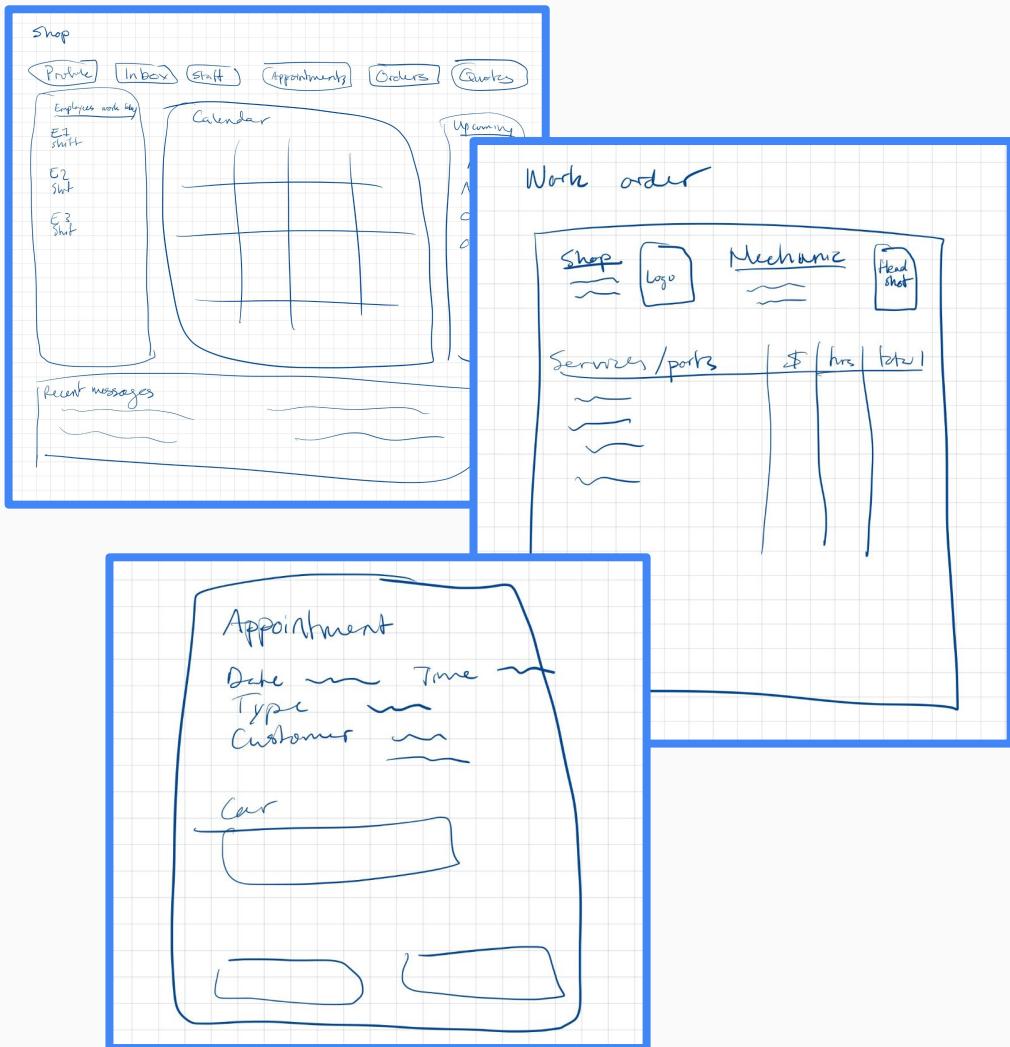
Mockups



Development Roadmap



Standup Agenda



# Key Learnings



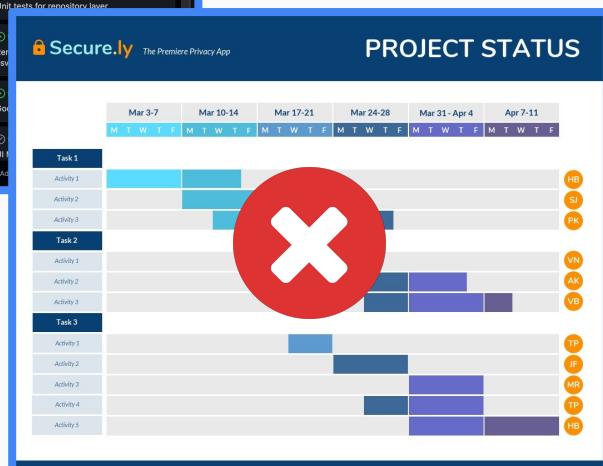
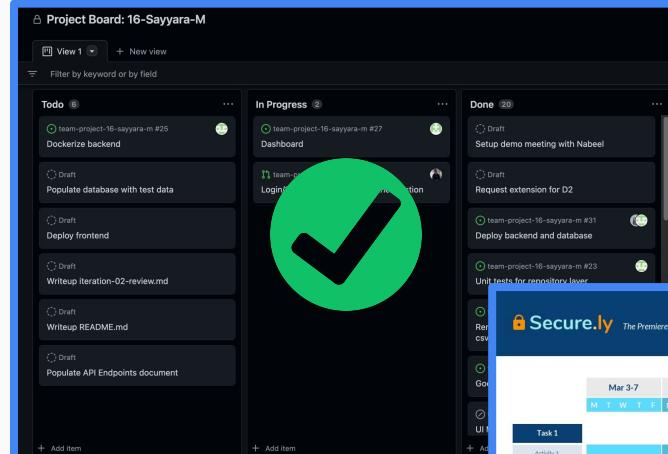
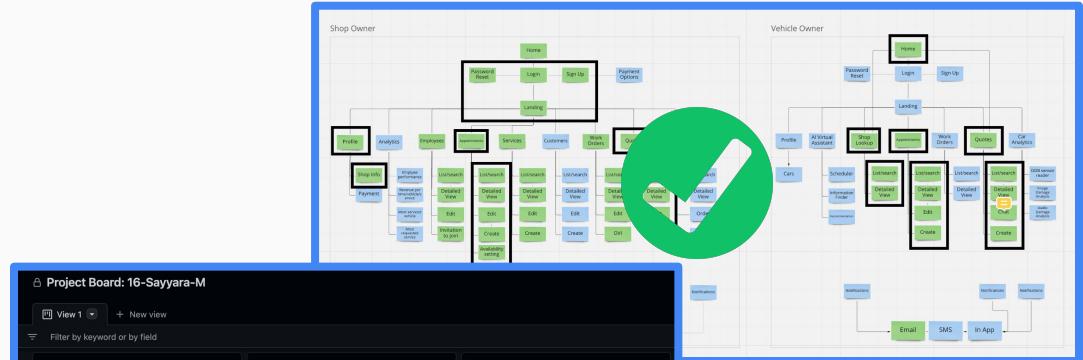
Mockups



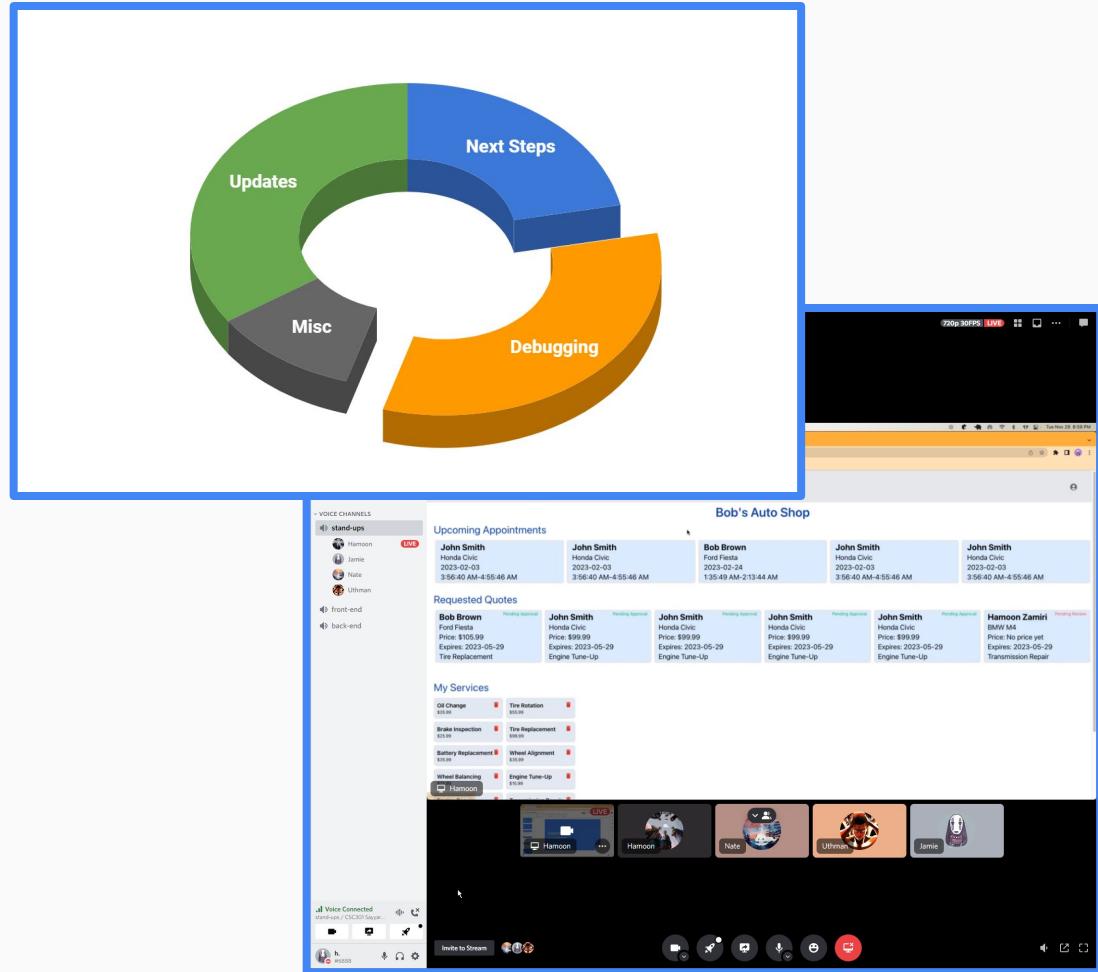
Development Roadmap



Standup Agenda



## Key Learnings



# Key Learnings



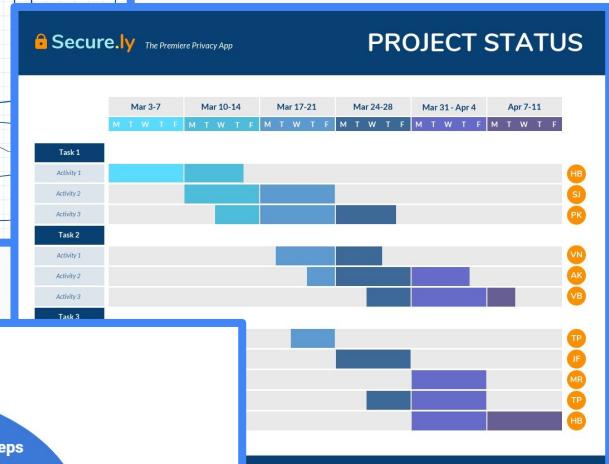
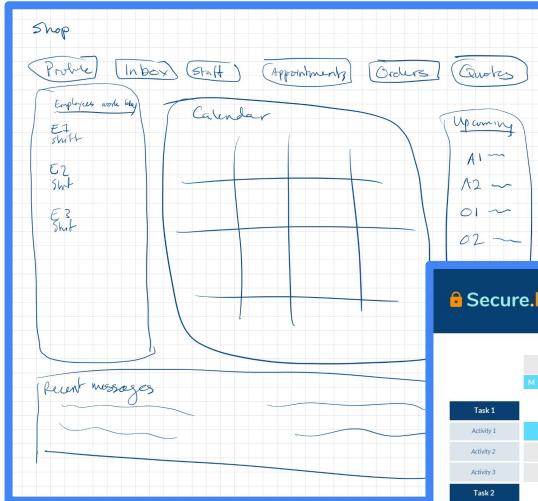
Mockups



Development Roadmap



Standup Agenda



# Thanks!

Sayyara

