Software Architecture & Development:
PetPal

Master of Science
Applied Computer Science

Vindhya Hegde
Sushant Huilgol
Kamal Kumar Sardiwal
Kalpesh Patil
Ambuj Solanki

srh

Introduction

01

PetPal

Our app provides everything a pet needs in one convenient place. From adoption and events to vet bookings and daycare, we've got it all.

Features:

- Adoption services
- Pet events
- Pet Services - (Veterinary and Grooming Centers)
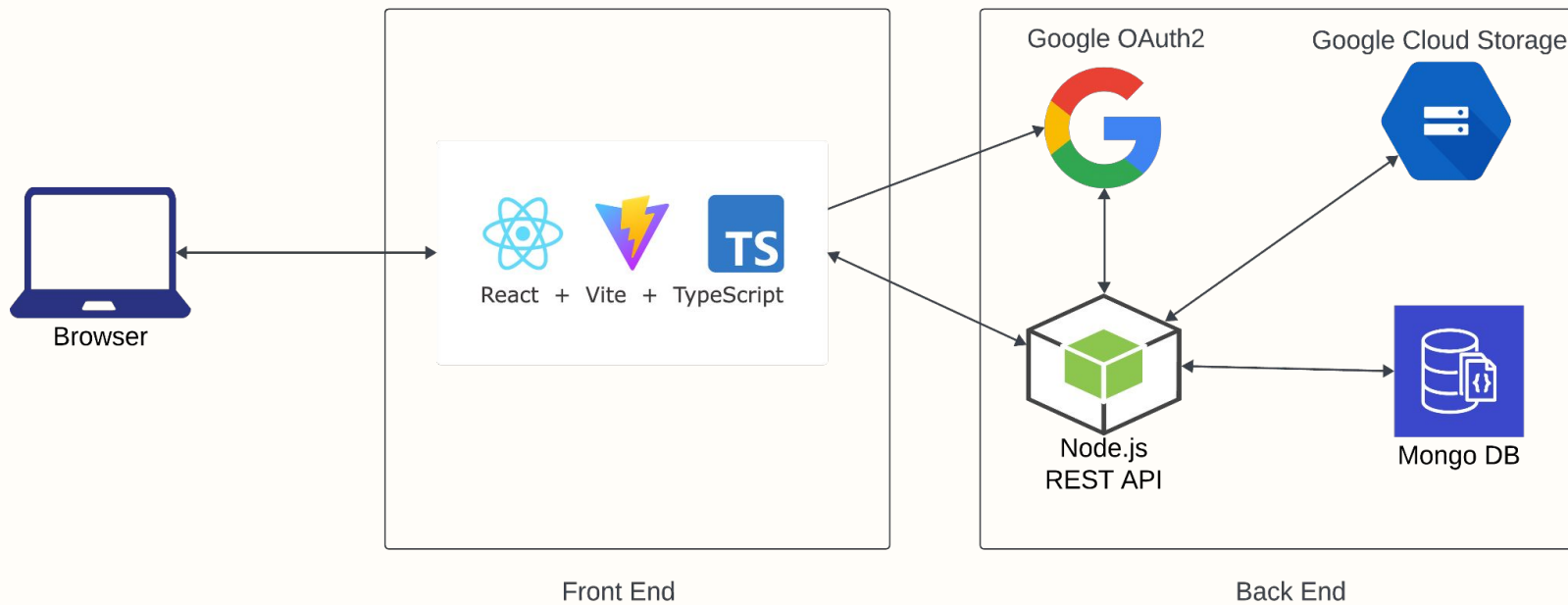- Lost and found
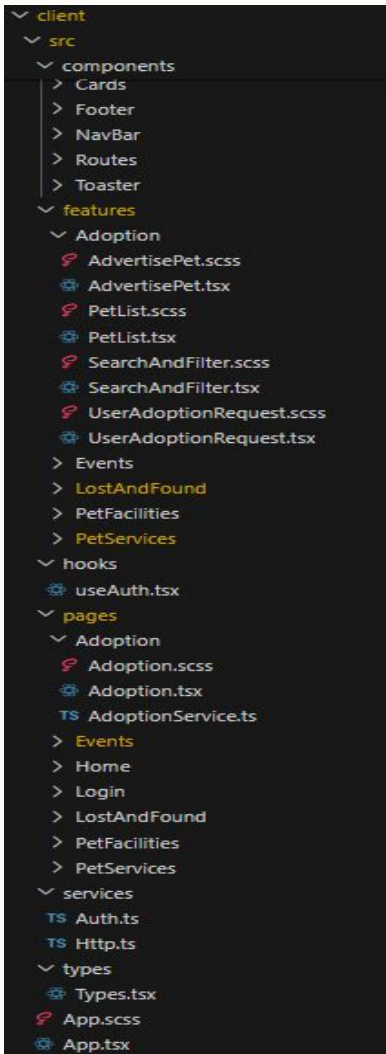- Pet Facilities - (Daycares and Caretakers)

Software Architecture Diagram

02

Browser

React + Vite + TypeScript

Front End

Google OAuth2

Google Cloud Storage

Node.js
REST API

Mongo DB

Back End

srh

React application folder structure

03

```
∨ client
  ∨ src
    ∨ components
      > Cards
      > Footer
      > NavBar
      > Routes
      > Toaster
    ∨ features
      ∨ Adoption
          AdvertisePet.scss
          AdvertisePet.tsx
          PetList.scss
          PetList.tsx
          SearchAndFilter.scss
          SearchAndFilter.tsx
          UserAdoptionRequest.scss
          UserAdoptionRequest.tsx
      > Events
      > LostAndFound
      > PetFacilities
      > PetServices
    ∨ hooks
        useAuth.tsx
    ∨ pages
      ∨ Adoption
          Adoption.scss
          Adoption.tsx
       TS AdoptionService.ts
      > Events
      > Home
      > Login
      > LostAndFound
      > PetFacilities
      > PetServices
    ∨ services
     TS Auth.ts
     TS Http.ts
    ∨ types
        Types.tsx
      App.scss
      App.tsx
```

"**components** " - Holds reusable components that are shared across multiple features.

"**features**" - Each feature has its own folder, which contains all its tightly coupled components, hooks, services, and styles. This encapsulation helps in managing feature-specific logic and makes the codebase more modular.

"**hooks**" - Contains reusable custom hooks that are not specific to any single feature.

"**pages**" - Contains page-level components. Each page typically corresponds to a route in your application and aggregates multiple components to form a complete view.

"**services**" - Includes services that are shared between many features.

"**types**" - Includes interfaces.

Nodejs Folder Structure

03

"config" - This folder holds configuration files and settings for the server.

"controller" - The files in this folder process requests, interact with models, and send responses back to the client. They act as an intermediary between the routes and models.

"models" - The files in this folder define the structure of the data, the relationships between different data entities, and include functions for interacting with the database (e.g., schemas for MongoDB using Mongoose).

"routes" - Contains handlers that map HTTP requests to specific controller functions, defining how different endpoints of API should respond.
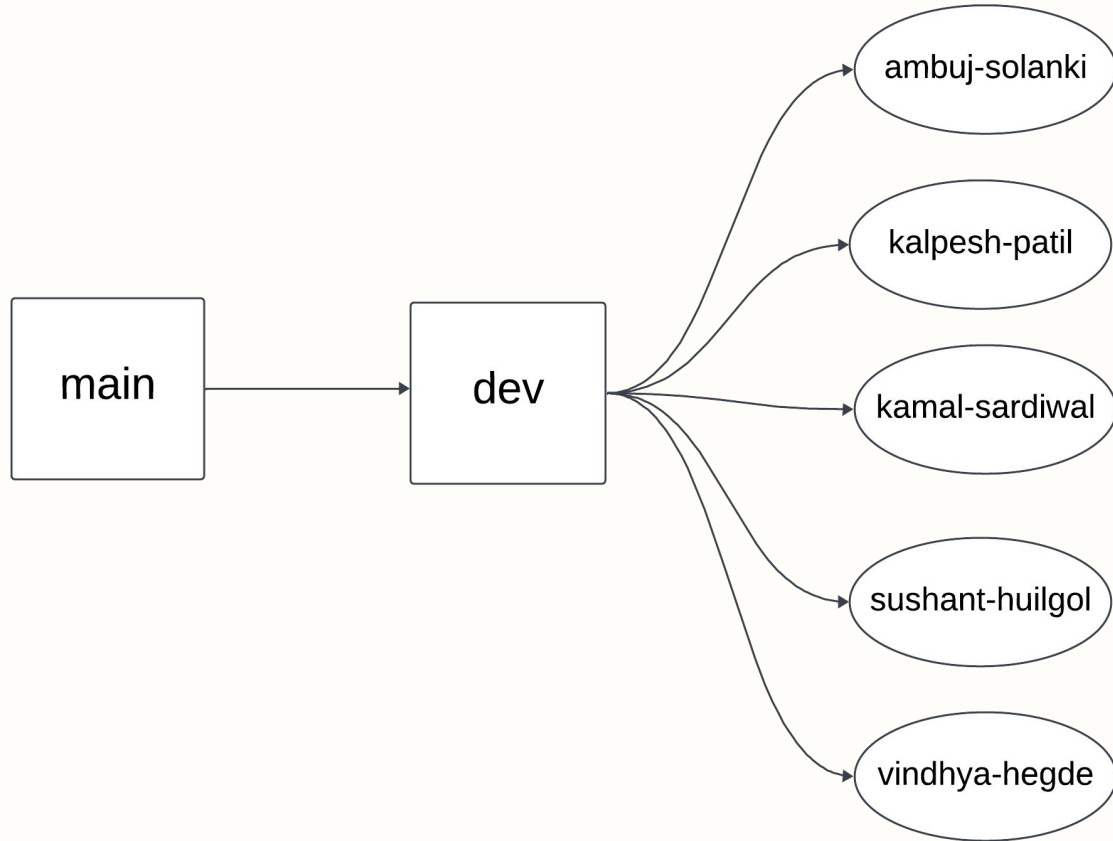
"utils" - This folder contains reusable pieces of code that can be used across various parts of application to perform common tasks.

srh

Git Structure
(sad-01-24-petpal)

04

Branch Structure

Branch Rules

main & dev branch
- Require a pull request before merging
- Require approvals from 2 approvers
- Do not allow bypassing the above settings
- Restrict who can push to matching branches

feature branches
- Restrict who can push to matching branches
- Restrict pushes that create matching branches

Commit Message Prefixes Followed

- **feat**: A new feature
- **fix**: A bug fix
- **docs**: Documentation only changes
- **style**: Changes that do not affect the meaning of the code (white-space, formatting, missing semi-colons, etc)
- **refactor**: A code change that neither fixes a bug nor adds a feature
- **perf**: A code change that improves performance
- **test**: Adding missing tests or correcting existing tests
- **chore**: Changes to the build process or auxiliary tools and libraries such as documentation generation
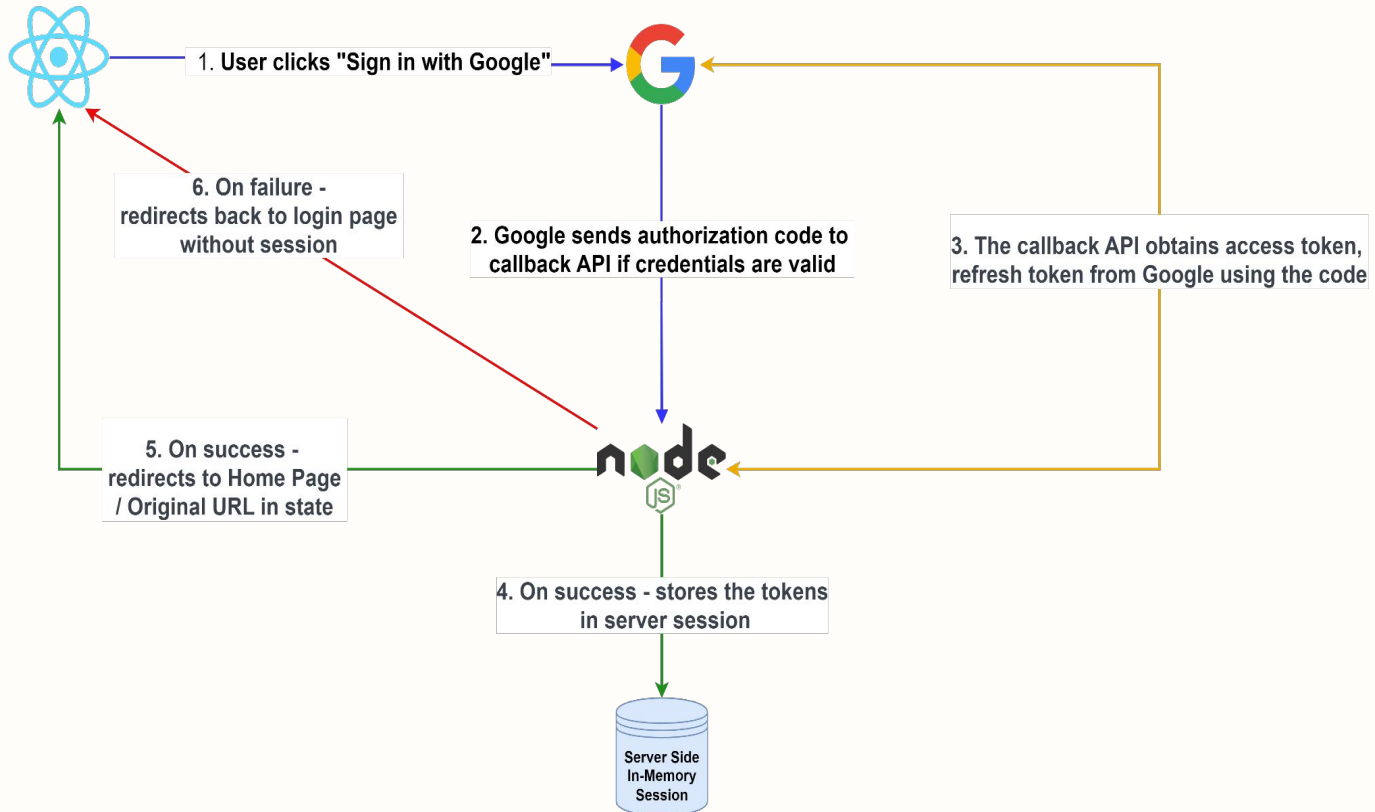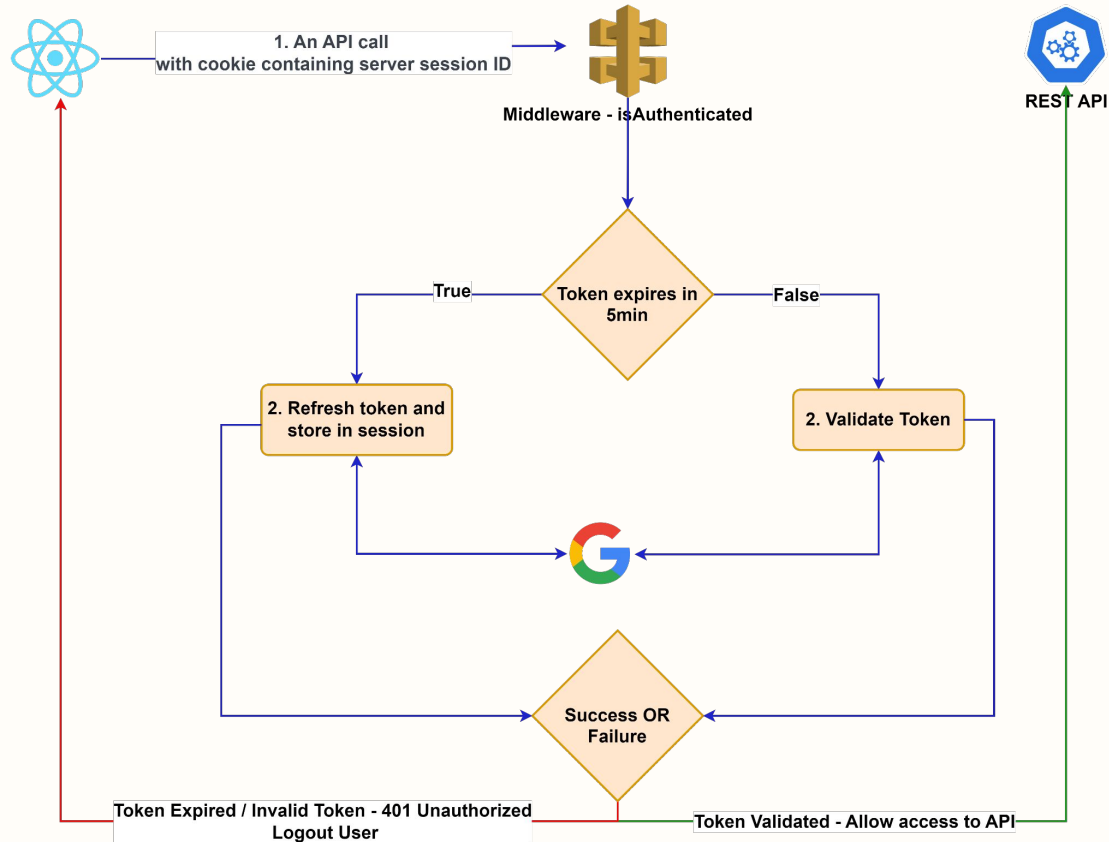
Google OAuth Login

05

OAuth2 Login Flow

1. **User clicks "Sign in with Google"**

2. **Google sends authorization code to callback API if credentials are valid**

3. **The callback API obtains access token, refresh token from Google using the code**

6. On failure - redirects back to login page without session

5. **On success - redirects to Home Page / Original URL in state**

4. On success - stores the tokens in server session

Server Side In-Memory Session

Authenticated User Flow - Token Validation

1. An API call
with cookie containing server session ID

Middleware - isAuthenticated

REST API

Token expires in 5min

True

False

2. Refresh token and store in session

2. Validate Token

Success OR Failure

Token Expired / Invalid Token - 401 Unauthorized Logout User

Token Validated - Allow access to API

Individual Contribution

06

Individual Contribution

Vindhya
01
- Events page: Frontend and Backend
- Google OAuth2 - User and API authentication
- Image Upload Service
- Frontend setup

Sushant
02
- Master Detail View
- Services page
- Frontend and Backend
- Git repository setup
- Reusable React Components
- Devops

Kalpesh
03
- Lost & Found page
- Frontend and Backend Development
- Backend setup
- Database Setup

Kamal
04
- Adoption page
- Frontend and Backend

Ambuj
05
- Facilities page
- Frontend and Backend

Demo

srh

Thanks for your attention.

Danke für Ihre Aufmerksamkeit.