DIGT2107: Practice of Software Development

Project Iteration 3.1: Software Design Documentation

OpenBid

Team 1: Tyler Mani Yanness Alaister

Due: February 1, 2026

Course: DIGT2107 – Winter Term 2026 | **Instructor:** Dr. May Haidar

1 Introduction

Project Name: OpenBid
Team Number: 1
Team Members: Tyler, Mani, Yanness, Alaister

Document Overview This document outlines the Software Design Document (SDD) for Iteration 3.1 of the OpenBid platform. It provides advanced software design representations through Class Diagrams, Sequence Diagrams, and Activity Diagrams for all major system components:

- **Stripe KYC & User Profile** – Identity verification and profile management
- **Jobs & Bids** – Job posting and bidding functionality
- **Authentication & 2FA** – User authentication with Duo MFA
- **Payments & Escrow** – Stripe payment processing and escrow management

Each section includes class diagrams showing component structure, sequence diagrams illustrating key flows, activity diagrams depicting workflows, and design stories for the backlog.

2 Class Diagrams

2.1 User & Profile Domain

The following class diagram illustrates the core classes related to user management and profile functionality.

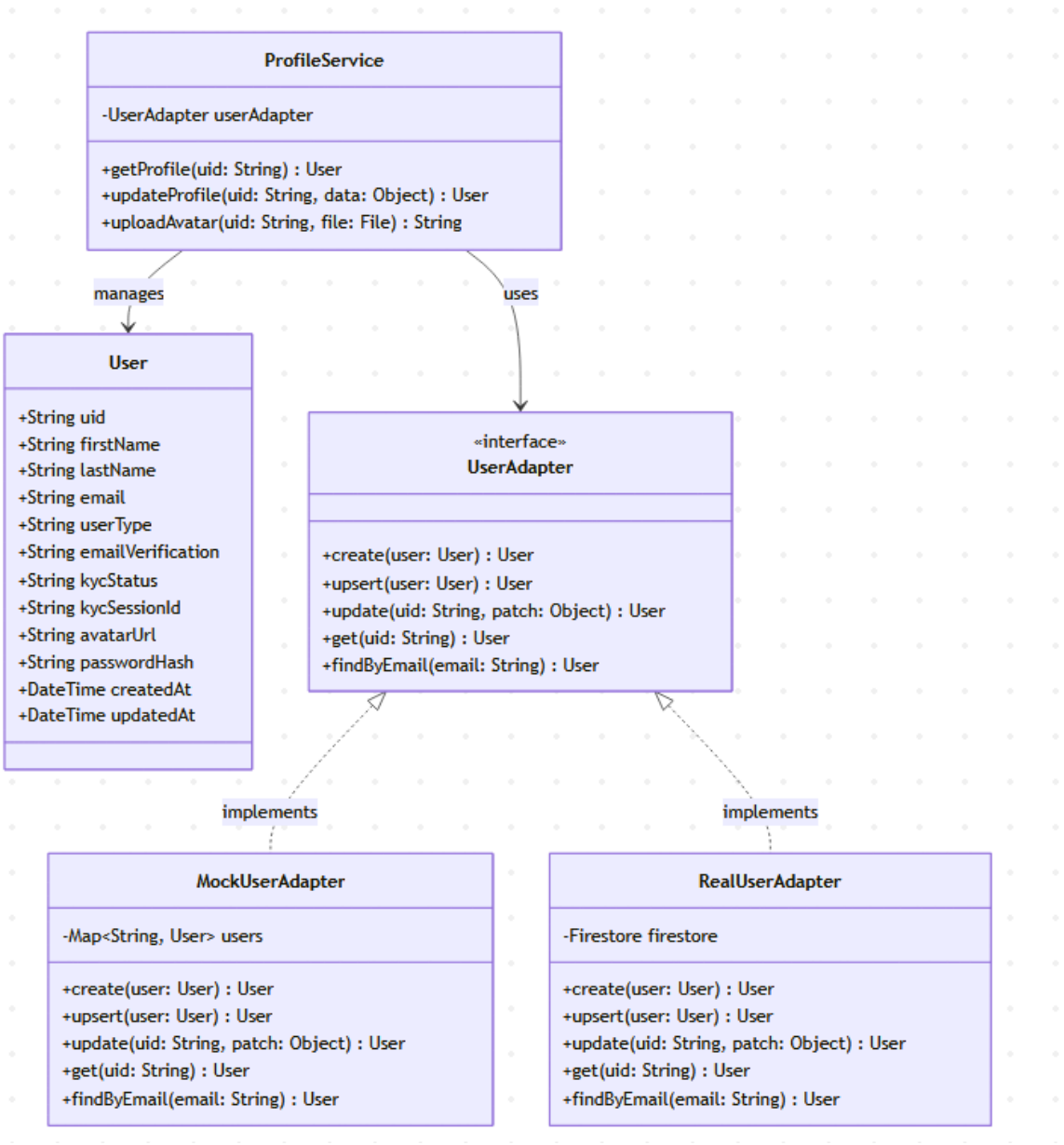

Figure 1: User & Profile Domain Class Diagram

2.2 Stripe KYC Domain

The following class diagram illustrates the classes related to Stripe Identity verification (KYC).

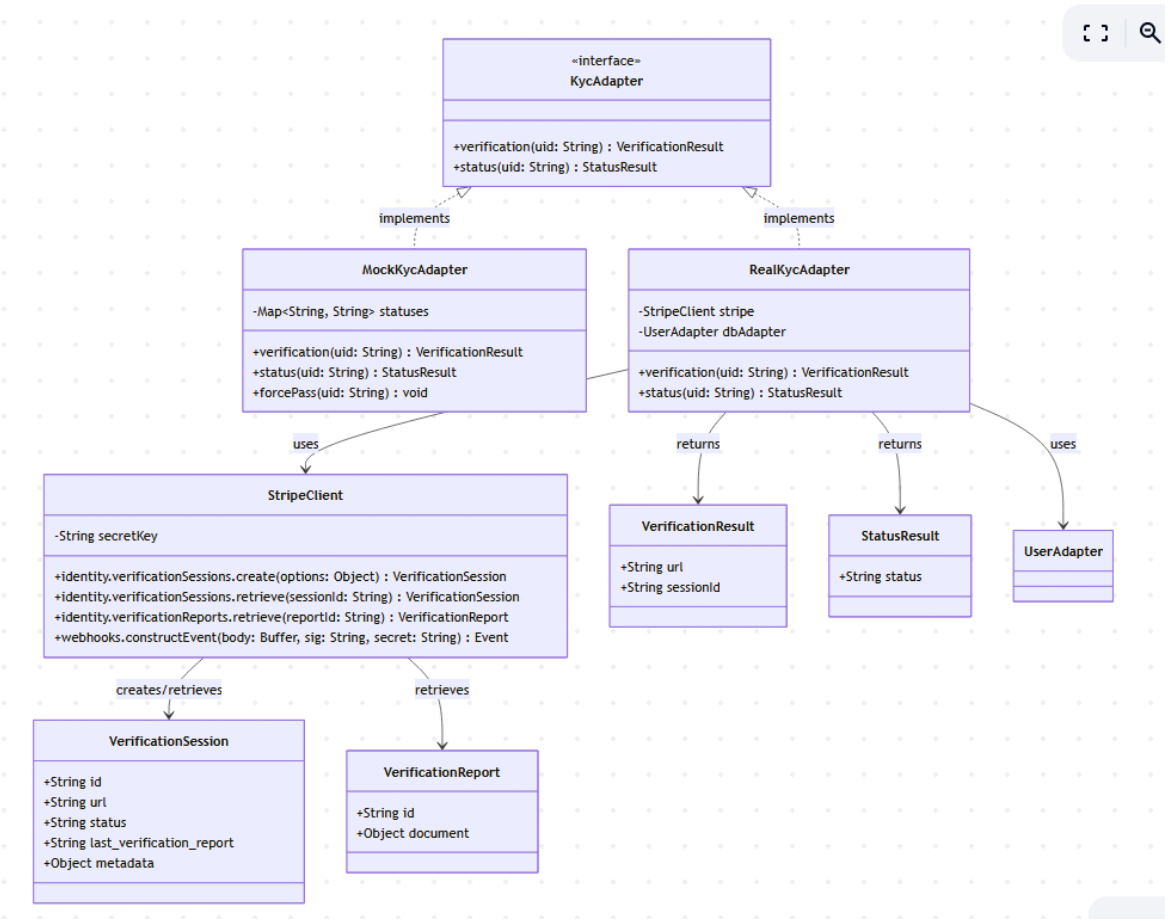

Figure 2: Stripe KYC Domain Class Diagram

2.3 Component Relationships

The following diagram shows how the KYC and Profile components interact with each other and external services.

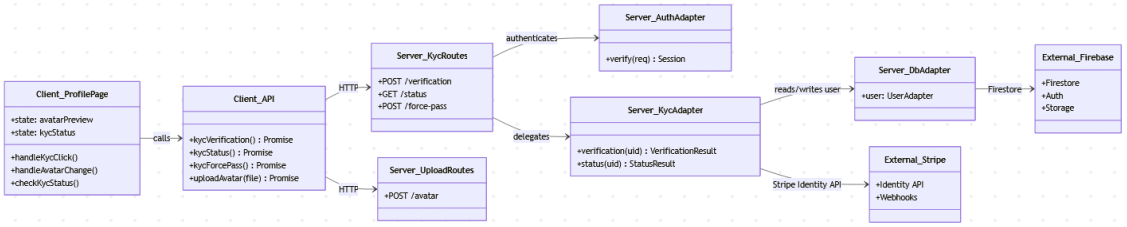

Figure 3: KYC and Profile Component Relationships

3 Sequence Diagrams

3.1 KYC Verification Flow - Part A: Initialization

This sequence diagram demonstrates the initial flow when a user starts KYC verification.

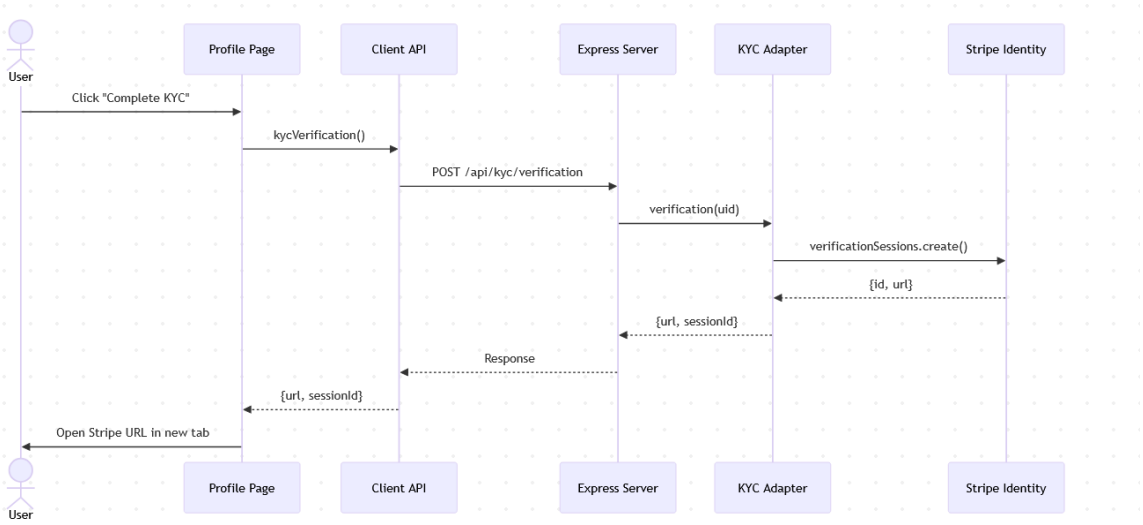

Figure 4: KYC Verification Initialization Flow

Flow Description:

1. User clicks "Complete KYC" button on Profile page.
2. Profile page calls `api.kycVerification()`.
3. Client API sends POST request to `/api/kyc/verification`.
4. Server delegates to KYC adapter's `verification()` method.
5. KYC adapter creates Stripe verification session.
6. Stripe returns session ID and verification URL.
7. Profile page opens Stripe verification in new tab.

3.2 KYC Verification Flow - Part B: Webhook & Status Check

This sequence diagram demonstrates the webhook handling and status check after user completes Stripe verification.

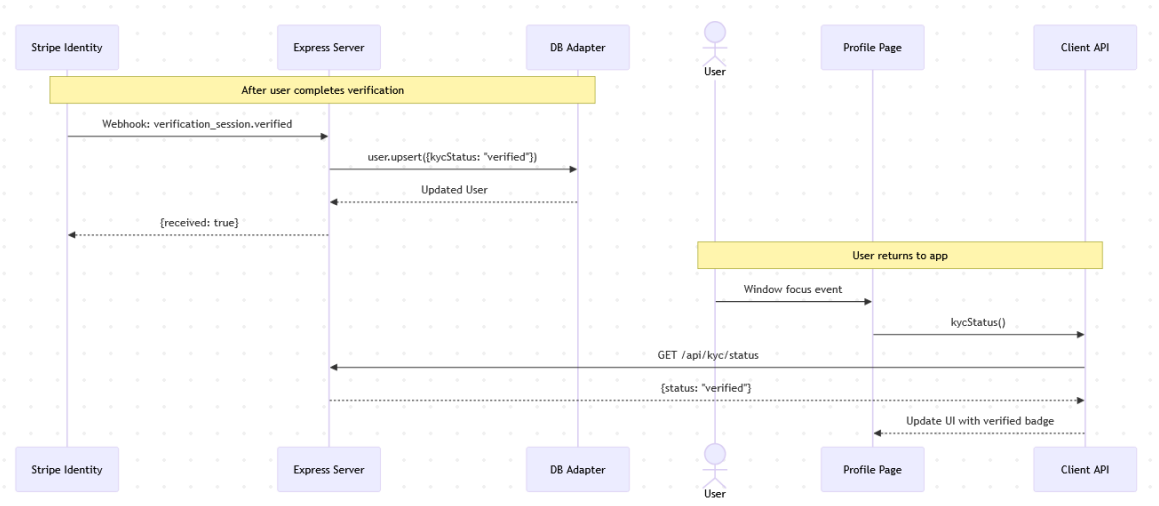

Figure 5: KYC Webhook and Status Check Flow

Flow Description:

1. After user completes verification, Stripe sends webhook event.
2. Server verifies webhook signature and updates user's KYC status.
3. When user returns to app, window focus event triggers status check.
4. Profile page fetches updated KYC status from server.
5. UI updates to show verified badge.

3.3 Profile Avatar Upload Flow

This sequence diagram demonstrates the profile avatar upload functionality.

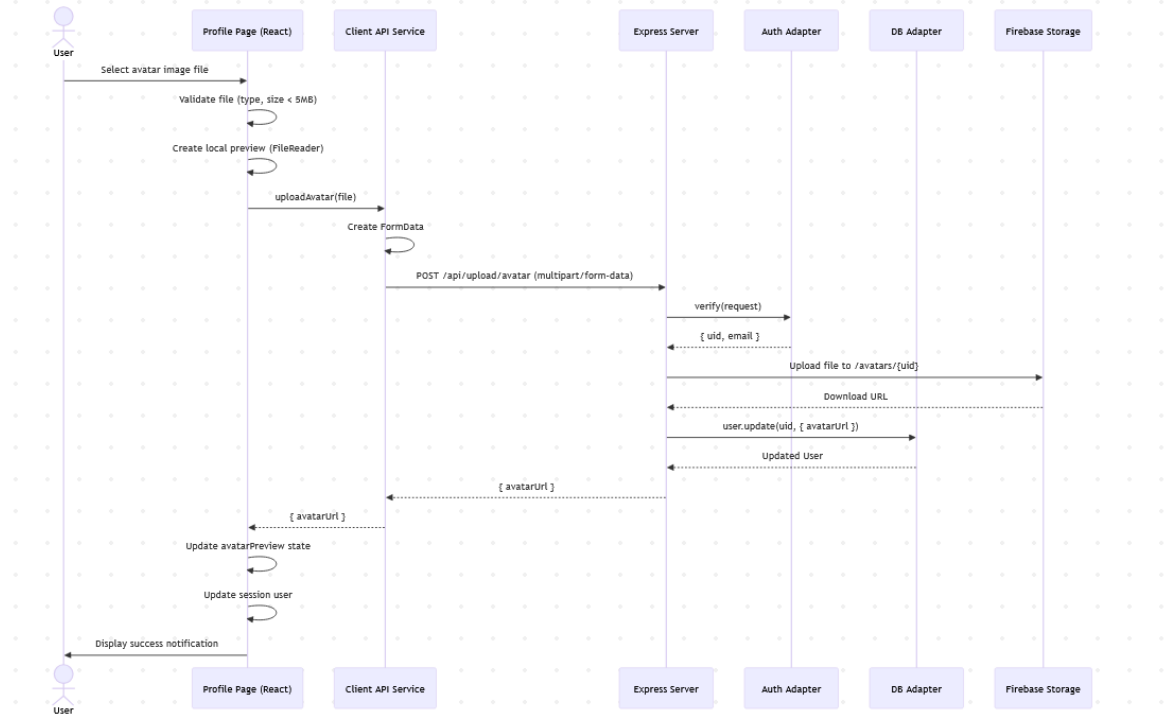

Figure 6: Avatar Upload Sequence Diagram

4 Activity Diagrams

4.1 KYC Verification Workflow

The following activity diagram illustrates the complete workflow for KYC verification with all decision points.

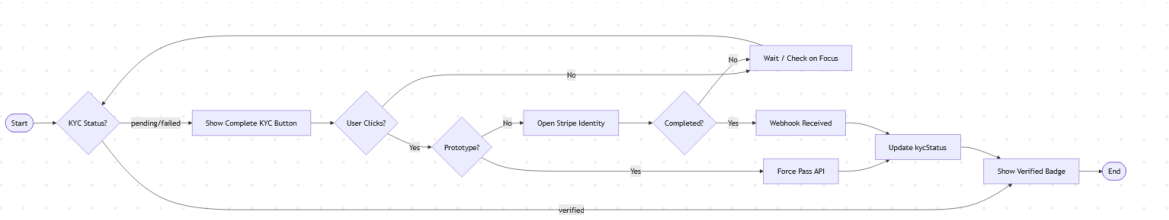

Figure 7: KYC Verification Activity Diagram

4.2 Profile Management Workflow

The following activity diagram illustrates the user profile management workflow.

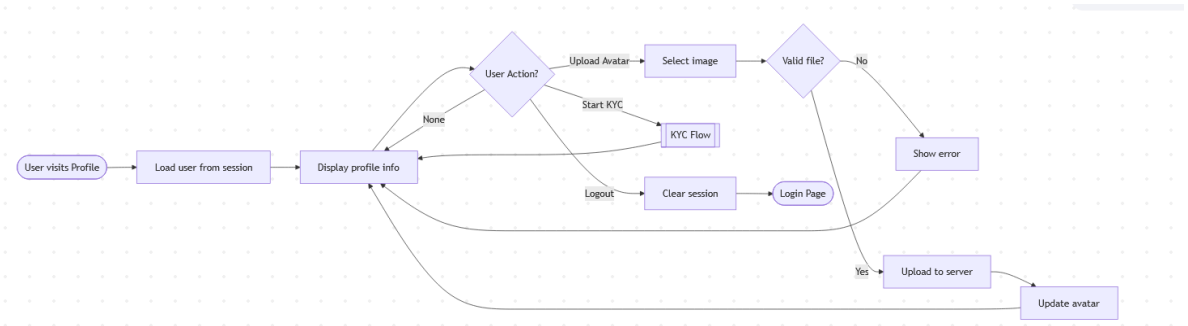

Figure 8: User Profile Management Activity Diagram

5 Component Summary

Component	Type	Technology	Responsibility
`Profile.jsx`	React Component	React, Carbon Design	User profile UI, avatar upload, KYC status
`kyc.routes.js`	Express Router	Node.js, Express	KYC API endpoints
`kyc.real.js`	Adapter	Stripe SDK	Real Stripe Identity integration
`kyc.mock.js`	Adapter	In-memory	Mock KYC for prototype mode
`index.js`	Server Entry	Express	Stripe webhook handler
`api.js`	Client Service	Fetch API	HTTP client for KYC/Profile calls

6 Updated Backlog with Design Stories

The KYC and Profile features documented above are complete. The following design stories outline the upcoming **In-App Messaging** feature for the next iteration:

Messaging Design Stories

1. **DS-MSG-001: Design Message Data Model** (High, 3 pts)
   Define message schema including sender, receiver, content, timestamp, read status, and job/bid thread linking. Design conversation thread structure for organizing messages between contractors and bidders.

2. **DS-MSG-002: Design Real-Time Architecture** (High, 5 pts)
   Evaluate and select real-time messaging approach: Firebase Realtime Database listeners vs Firestore onSnapshot vs WebSockets. Design message synchronization and offline support.

3. **DS-MSG-003: Design Chat UI Component** (Medium, 3 pts)
   Design chat interface with message bubbles, input field, send button, and typing indicators. Support for text messages and future attachment support.

4. **DS-MSG-004: Design Notification System** (Medium, 3 pts)
   Design push notification flow for new messages. Integrate with Firebase Cloud Messaging for real-time alerts when user is not in the app.

5. **DS-MSG-005: Design Job-Thread Linking** (High, 2 pts)
   Design how message threads are linked to specific jobs and bids. Ensure only awarded bidders can message contractors, and vice versa.

7 Preliminary Test Coverage Report

The following test coverage report was generated using Jest with the `-coverage` flag. This report covers the KYC and Profile-related components.

Test Results Summary

- **Test Suites:** 5 passed, 1 skipped, 6 total
- **Tests:** 33 passed, 2 skipped, 35 total
- **Snapshots:** 0 total

Coverage by Component (KYC & Profile)

File	% Stmts	% Branch	% Funcs	% Lines	
`kyc.routes.js`	87.09	92.85	100	89.28	
`kyc.real.js`	38.23	25.00	100	36.36	
`kyc.mock.js`	25.00	0.00	0	25.00	
`db.real.js` (User data)	64.23	43.10	71.42	72.41	
Overall	53.73	46.20	57.14	56.33	

Analysis

- **KYC Routes:** Excellent coverage at 89% line coverage. All API endpoints are tested.
- **KYC Real Adapter:** Lower coverage (36%) due to Stripe API integration being mocked during testing.
- **Database Adapter:** Good coverage at 72% for user data operations.
- **Gaps Identified:** The `kyc.mock.js` adapter has low coverage as it is primarily used for prototype mode testing only.

8 GUI Implementation Screenshots

The following screenshots demonstrate the Profile page implementation, showing the KYC verification status and user interface elements.

Profile Page - Before KYC Verification

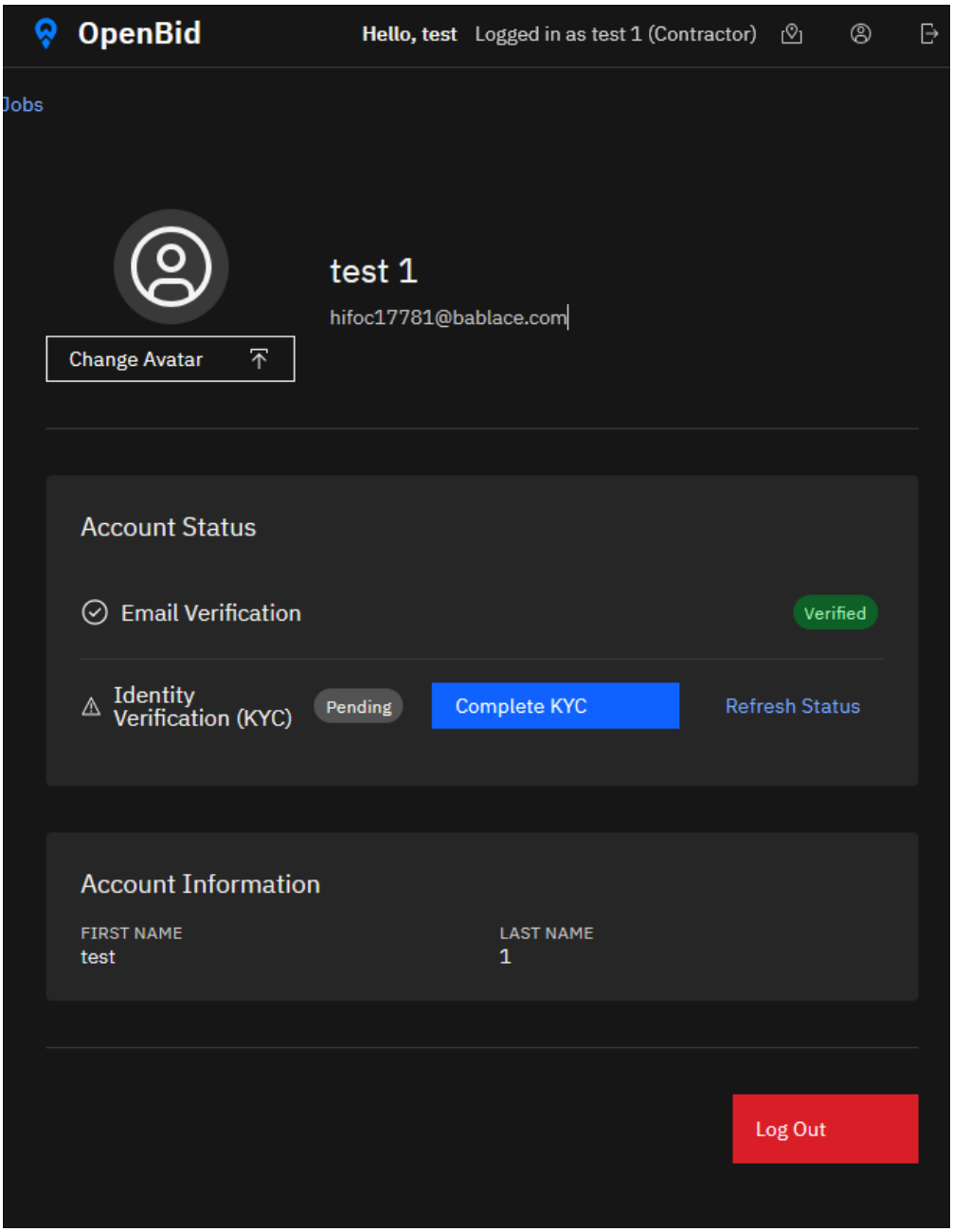

Figure 9: Profile page showing pending KYC status with "Complete KYC" button

Profile Page - After KYC Verification

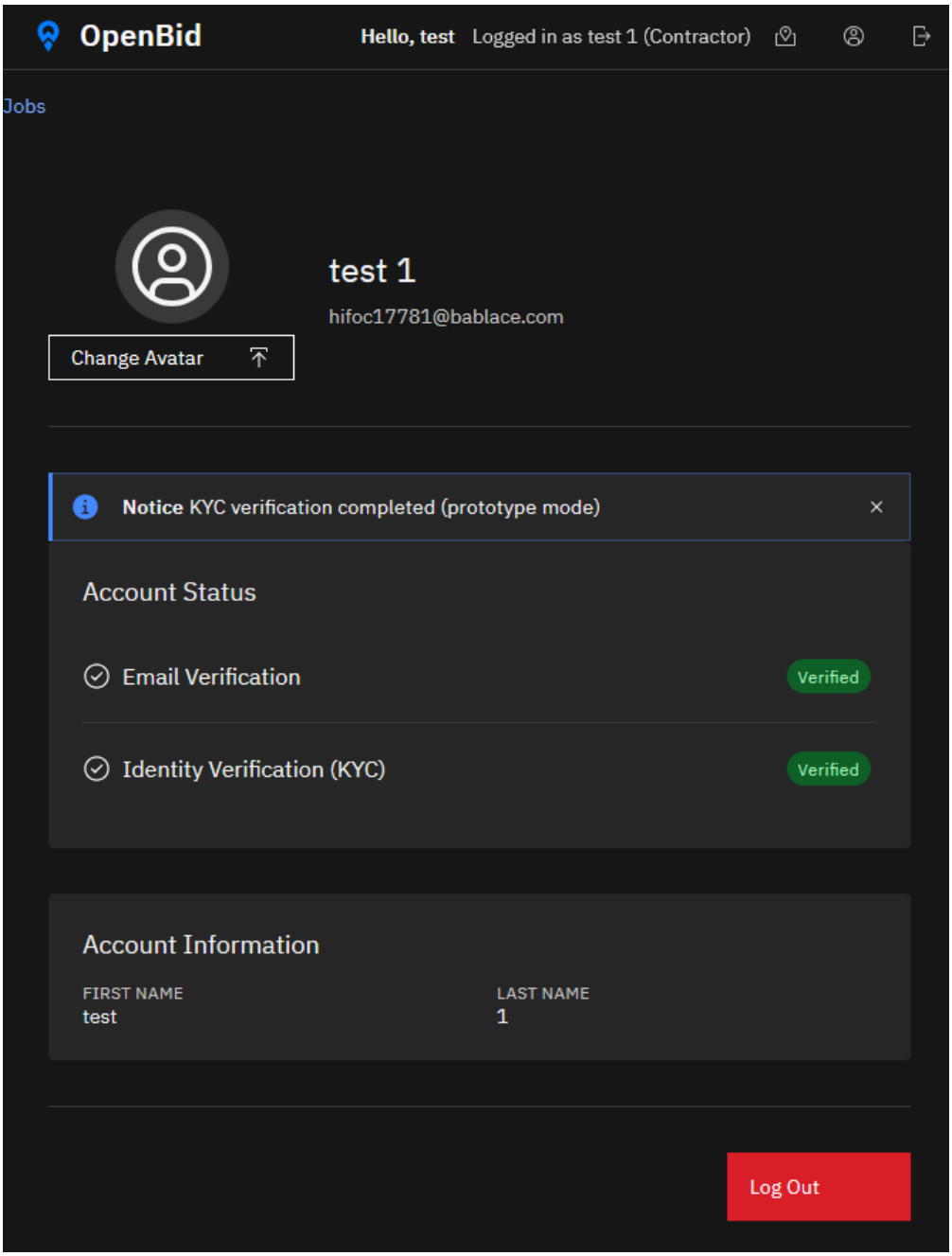

Figure 10: Profile page showing verified KYC status with green badge

Key UI Elements

- **Avatar Upload:** Users can upload and change their profile picture
- **KYC Status Display:** Visual indicator showing pending, verified, or failed status
- **Action Buttons:** "Complete KYC" and "Refresh Status" buttons for unverified users

- **Account Information:** Display of user's name, email, and verification status

[TEAMMATE 1] Jobs & Bids Domain

Section owner: [Name] – Please add your diagrams below

Class Diagrams

Sequence Diagrams

Activity Diagrams

Design Stories

[TEAMMATE 2] Authentication & 2FA Domain

Section owner: [Name] – Please add your diagrams below

Class Diagrams

Sequence Diagrams

Activity Diagrams

Design Stories

[TEAMMATE 3] Payments & Escrow Domain

Section owner: [Name] – Please add your diagrams below

Class Diagrams

Sequence Diagrams

Activity Diagrams

Design Stories

Class Diagrams

8.1 Mapview Class

This diagram represents the model of the mapview component.

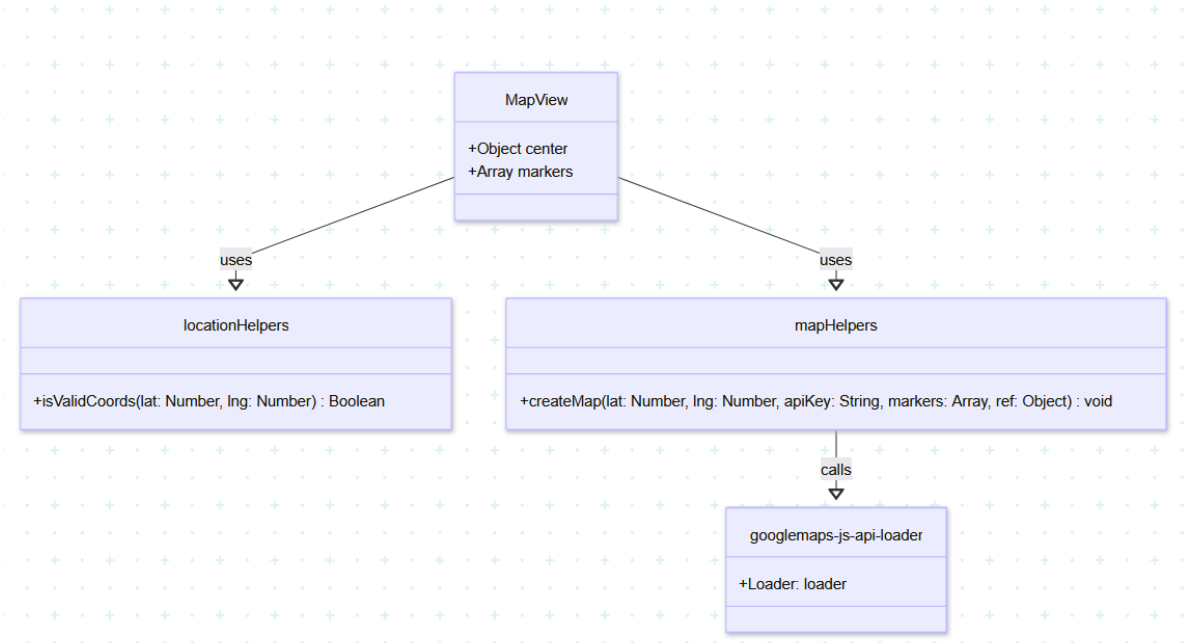

Figure 11: Mapview component as a class

8.2 SearchAutocomplete Class

This diagram represents the model of the searchAutocomplete component.

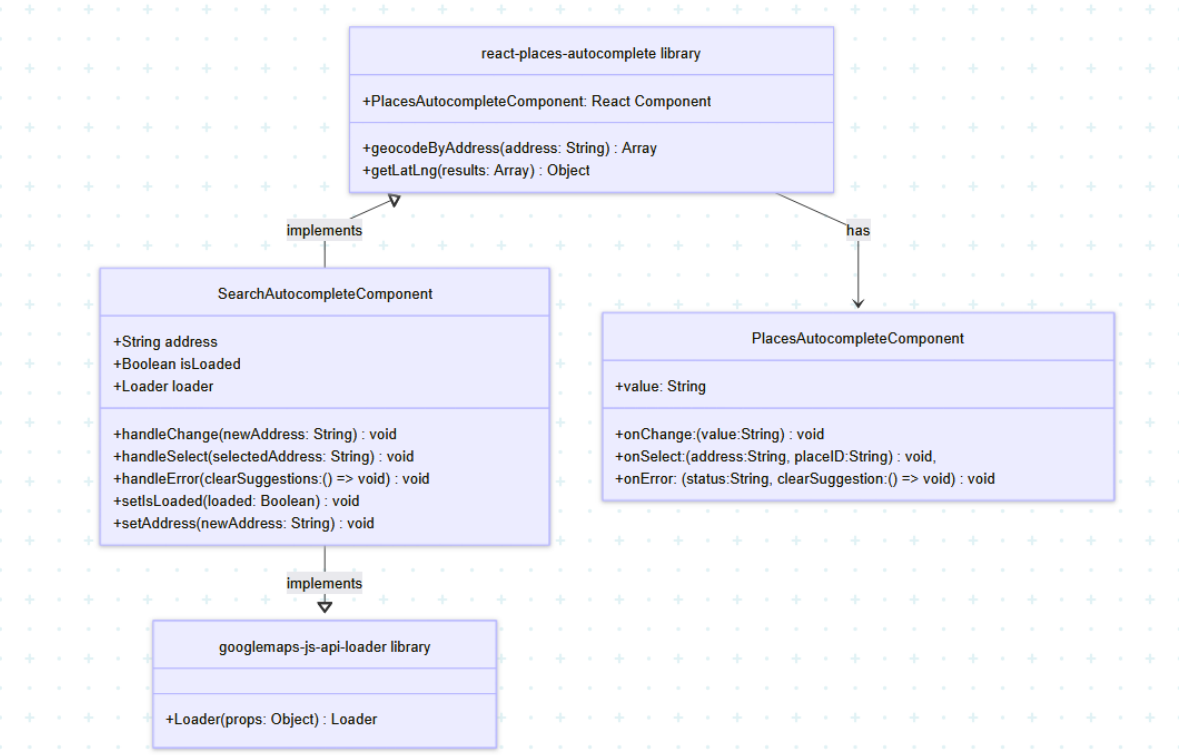

Figure 12: SearchAutocomplete component as a class

8.3 MapView and Autocomplete Classes

This diagram represents a page interacting with both the MapView and Autocomplete components.

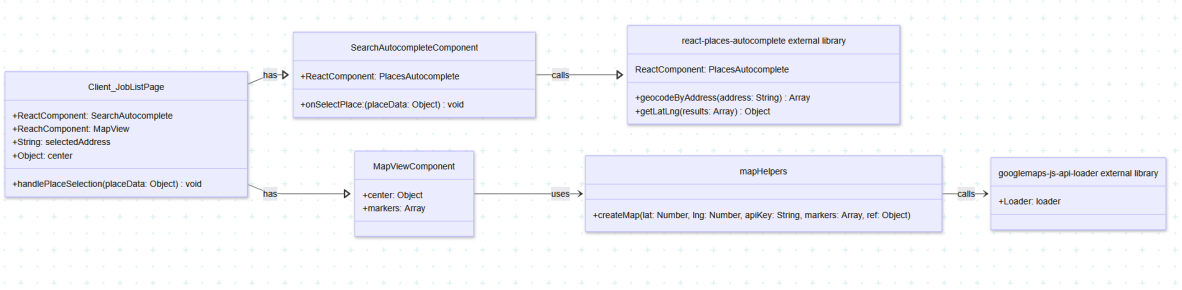

Figure 13: Autocomplete and Mapview components classes interaction

Sequence Diagrams

8.4 Mapview Component diagram flow on a page

This diagram represents a sequence diagram when visting a page with the mapview component (without the searchAutocomplete component).

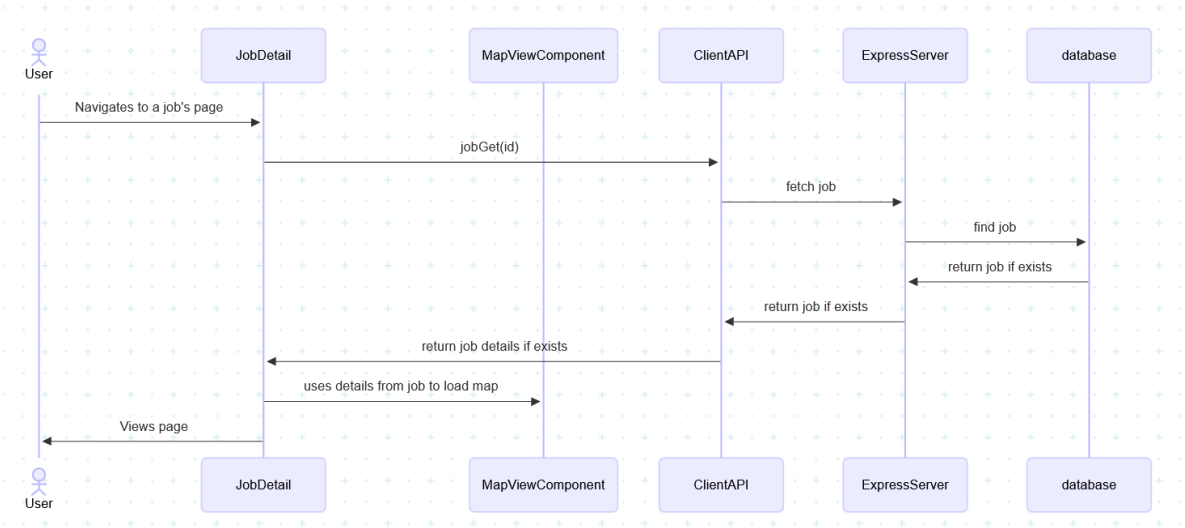

Figure 14: Mapview component sequence diagram

Flow Description: There are multiple pages that use the Mapview Component without the SearchAutocomplete component, but the JobDetail page was chosen in this example.

1. User navigates to a page with a Mapview component (and no SearchAutocomplete component).
2. Server fetches details of a job.
3. Page is hydrated with information regarding the job and the Mapview is updated with the job address.

8.5 SearchAutocomplete Component diagram flow on a page

*Intentionally omitted as this component will exclusively work with the Mapview component, never on its own.

8.6 Mapview and SearchAutocomplete Components diagram flow on a page

This diagram represents a sequence diagram when visting a page with both the mapview and searchautocomplete components.

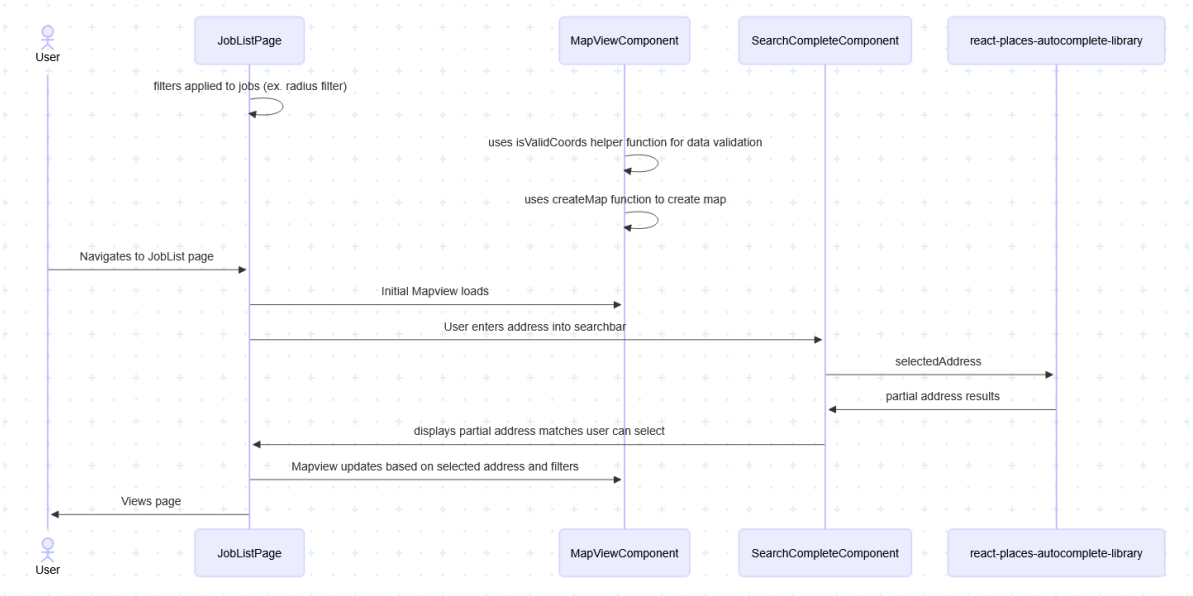

Figure 15: Mapview and search components sequence diagram

Flow Description: There are multiple pages that use both the Mapview Component and the SearchAutocomplete component, but the JobList page was chosen in this example.

1. User navigates to a page with both the Mapview component and SearchAutocomplete component).
2. An initial mapview loads in the page.
3. User updates the address in the search autocomplete searchbar.
4. react-places-autocomplete-library is called and returns partial address results.
5. Client UI displays partial address results.
6. User selects an address result and the page is updated.

Activity Diagrams

8.7 SearchAutocomplete and Mapview Component flow on a page

This diagram represents the actions occuring by the user and the system when interacting with the searchAutocomplete and Mapview components.

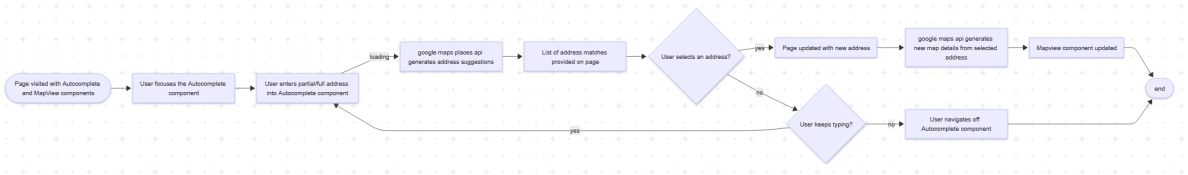

Figure 16: Mapview and searchAutocomplete components activity diagram

Design Stories

Admin Design Stories

1. **DS-Admin-001: Design Admin UI** (High, 5 pts)
   Design key admin pages and components- realtime updates, navigation, etc. should be provided within components.
2. **DS-Admin-002: Design Admin Data Model** (High, 5 pts)
   Design new user Admin in database and provide appropriate permissions.
3. **DS-Admin-003: Design with accessibility in mind** (Medium, 2 pts)
   Design an accessible dashboard to ensure any person can utilize the platform as an admin.

GUI

Page with Mapview Component

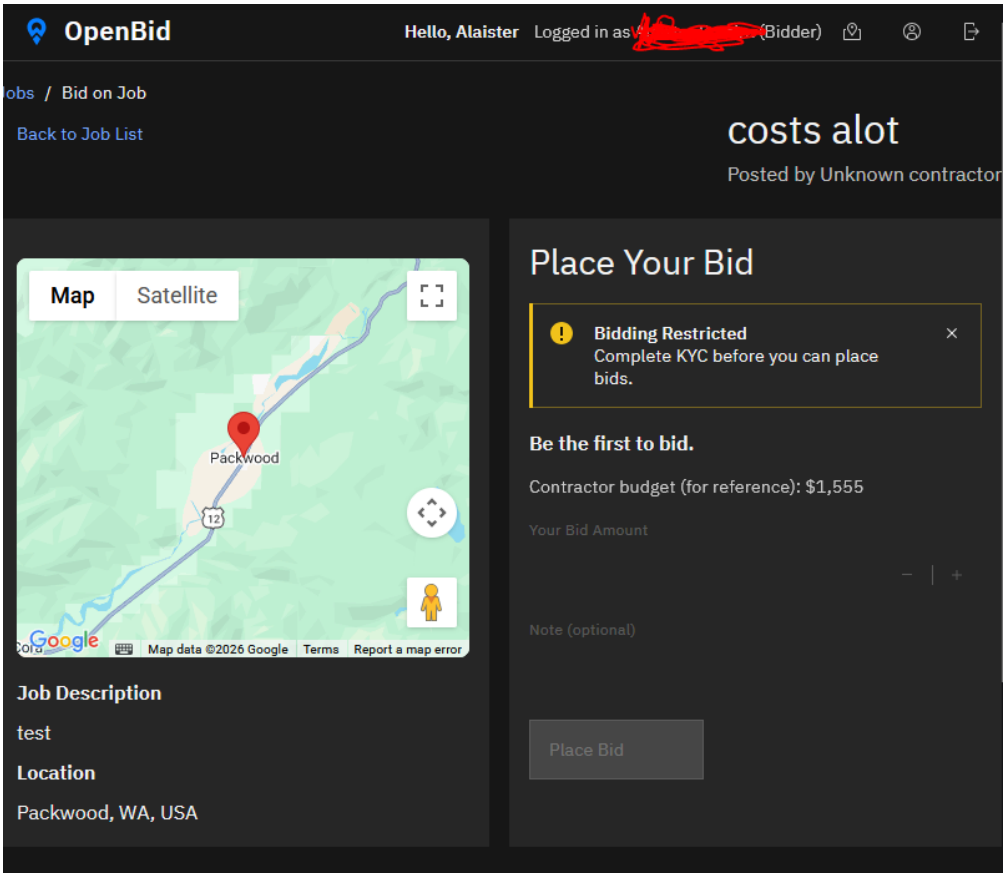

Figure 17: An example of a page that displays the MapView component without the SearchAutocomplete component.

Page with Mapview and SearchAutocomplete Components

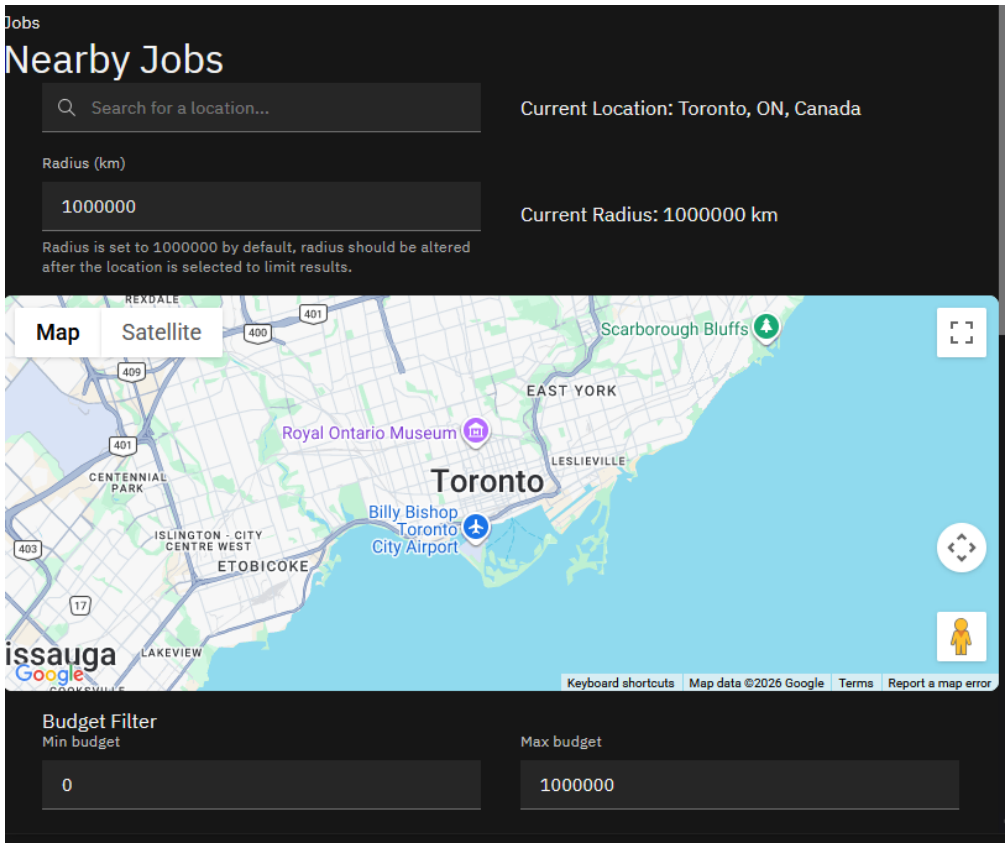

Figure 18: An example of a page that displays both the SearchAutoComplete component (as the bar to type an address) and the MapView component.

Coverage by Component (Maps)

File	% Stmts	% Branch	% Funcs	% Lines	
`locationHelpers.js`	100	100	100	100	
`mapHelpers.js`	100	100	100	100	
`MapView.jsx`	0	0	0	0	
`SearchAutocomplete.jsx`	0	0	0	0	

*There is full coverage for the map and search autocomplete in the js files helper functions. However, the MapView and SearchAutocomplete components themselves require component tests.