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.

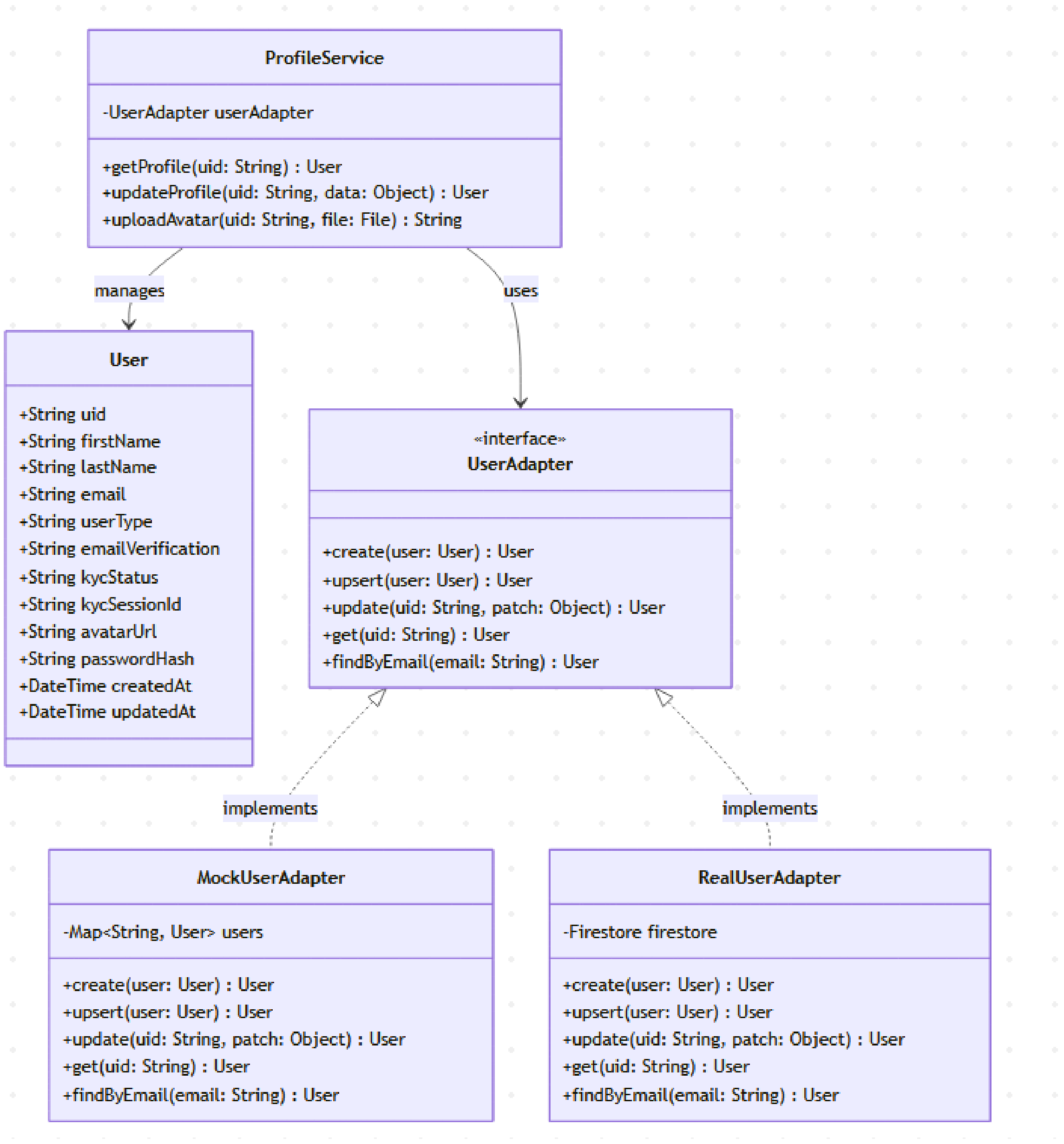

ProfileService
-UserAdapter userAdapter
+getProfile(uid: String) : User
+updateProfile(uid: String, data: Object) : User
+uploadAvatar(uid: String, file: File) : String
manages
uses
User
+String uid
+String firstName
+String lastName
+String email
+String userType
+String emailVerification
+String kycStatus
+String kycSessionId
+String avatarUrl
+String passwordHash
+DateTime createdAt
+DateTime updatedAt
«interface»
UserAdapter
+create(user: User) : User
+upsert(user: User) : User
+update(uid: String, patch: Object) : User
+get(uid: String) : User
+findByEmail(email: String) : User
implements
implements
MockUserAdapter
-Map<String, User> users
+create(user: User) : User
+upsert(user: User) : User
+update(uid: String, patch: Object) : User
+get(uid: String) : User
+findByEmail(email: String) : User
RealUserAdapter
-Firestore firestore
+create(user: User) : User
+upsert(user: User) : User
+update(uid: String, patch: Object) : User
+get(uid: String) : User
+findByEmail(email: String) : User

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).

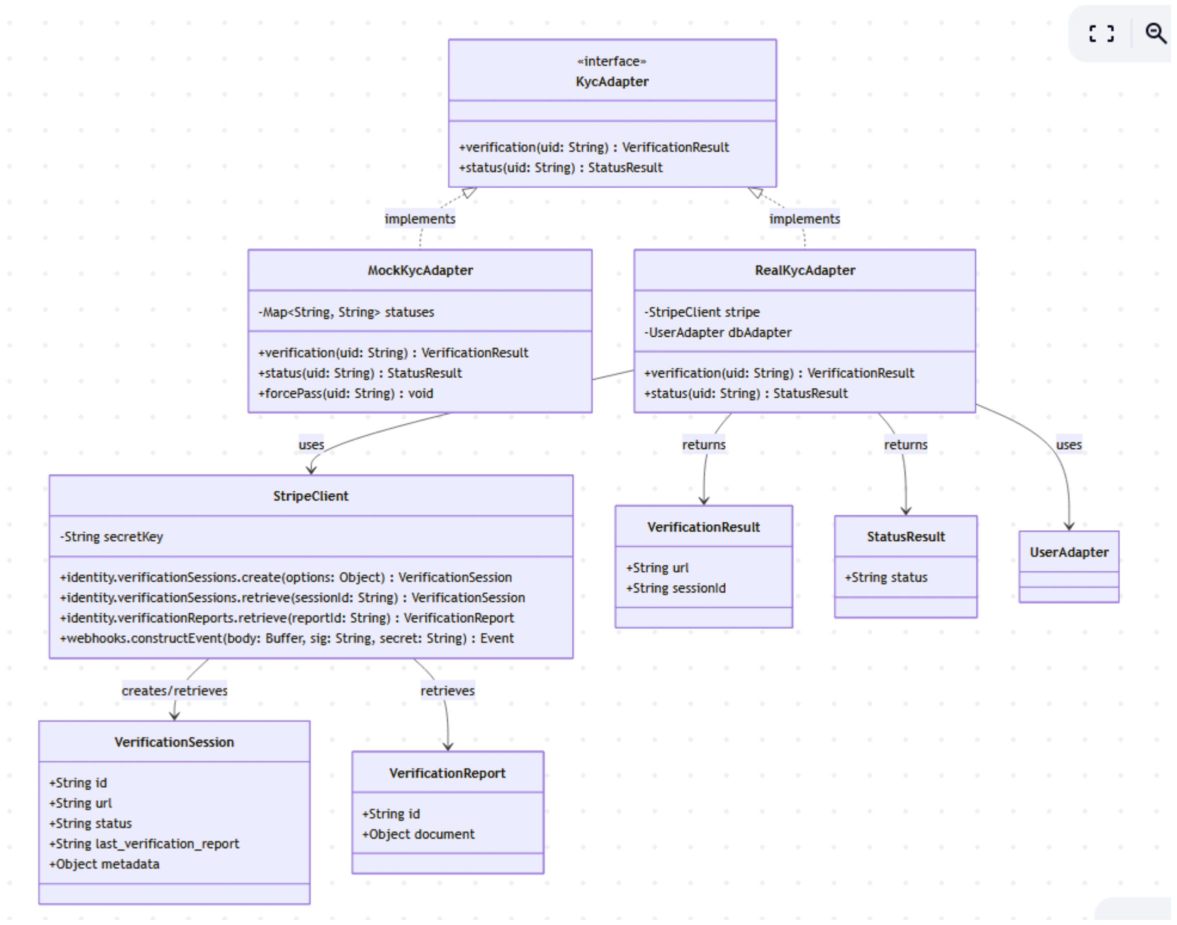

«interface»
KycAdapter
+verification(uid: String) : VerificationResult
+status(uid: String) : StatusResult
implements
implements
MockKycAdapter
-Map<String, String> statuses
+verification(uid: String) : VerificationResult
+status(uid: String) : StatusResult
+forcePass(uid: String) : void
RealKycAdapter
-StripeClient stripe
-UserAdapter dbAdapter
+verification(uid: String) : VerificationResult
+status(uid: String) : StatusResult
uses
returns
returns
uses
StripeClient
-String secretKey
+identity.verificationSessions.create(options: Object) : VerificationSession
+identity.verificationSessions.retrieve(sessionId: String) : VerificationSession
+identity.verificationReports.retrieve(reportId: String) : VerificationReport
+webhooks.constructEvent(body: Buffer, sig: String, secret: String) : Event
VerificationResult
+String url
+String sessionId
StatusResult
+String status
UserAdapter
creates/retrieves
retrieves
VerificationSession
+String id
+String url
+String status
+String last_verification_report
+Object metadata
VerificationReport
+String id
+Object document

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.

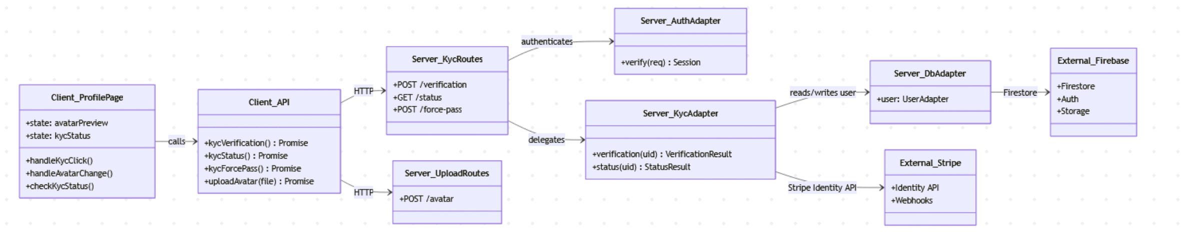


Figure 3: KYC and Profile Component Relationships

3 Sequence Diagrams

3.1 KYC Verification Flow - Part A: Initialization

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

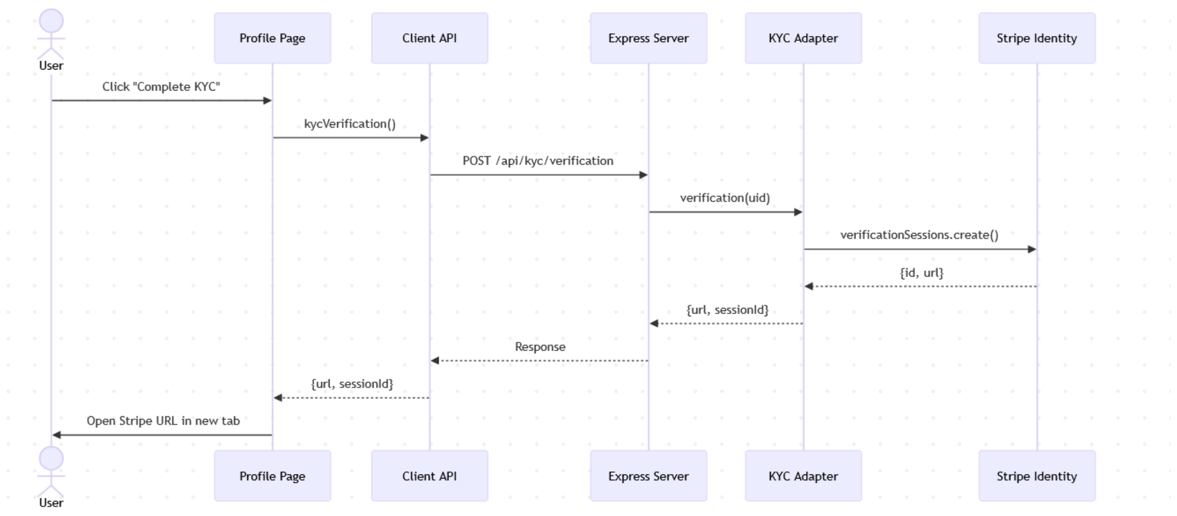


Figure 4: KYC Verification Initialization Flow

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.

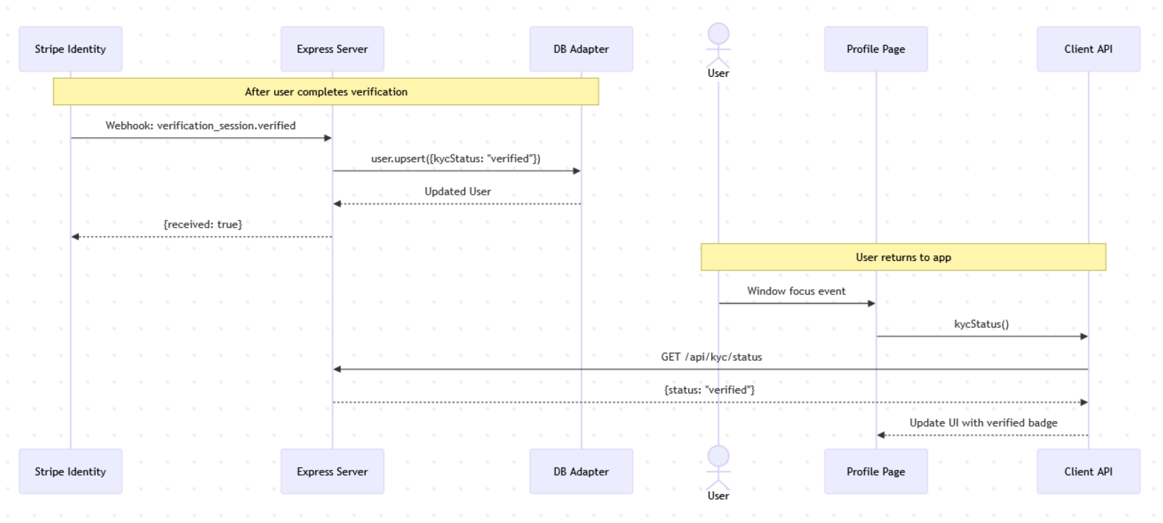


Figure 5: KYC Webhook and Status Check Flow

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.

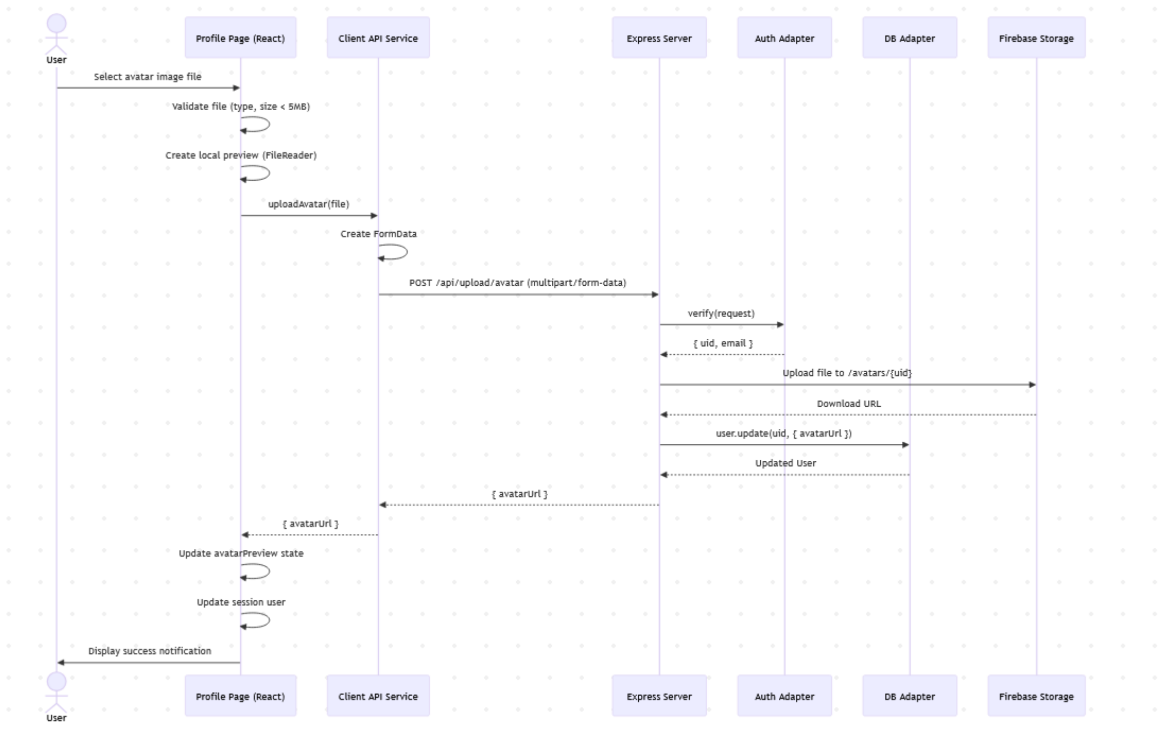


Figure 6: Avatar Upload Sequence Diagram

4 Activity Diagrams

4.1 KYC Verification Workflow

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

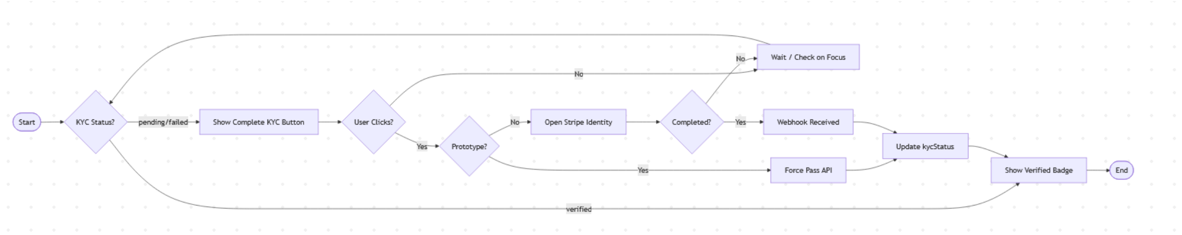


Figure 7: KYC Verification Activity Diagram

4.2 Profile Management Workflow

The following activity diagram illustrates the user profile management workflow.

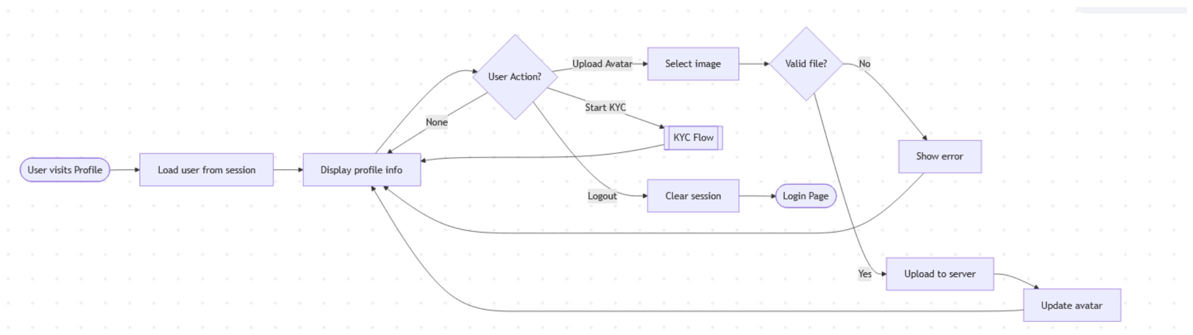


Figure 8: User Profile Management Activity Diagram

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	

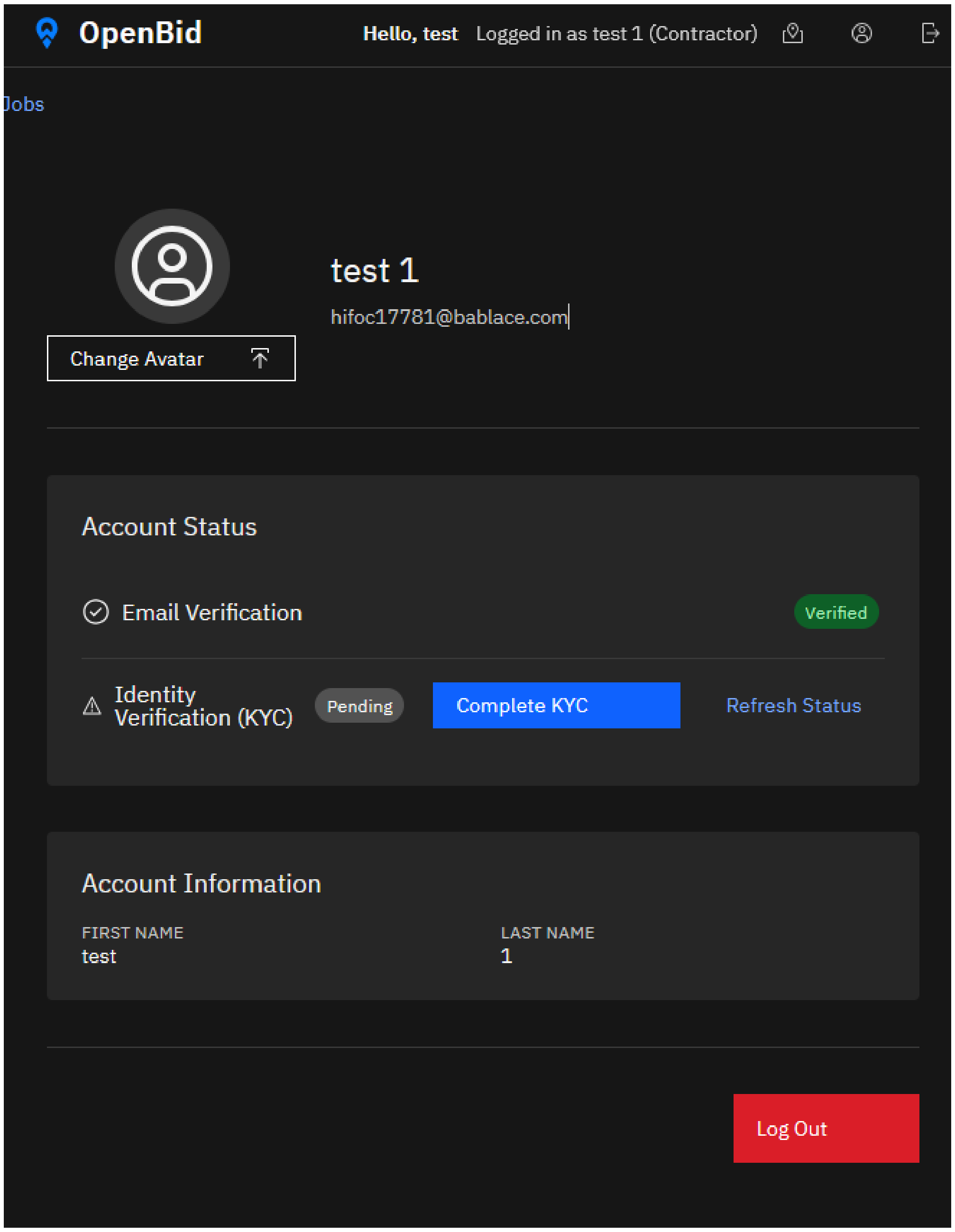
Analysis

- **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



test 1

hifoc17781@bablace.com

Account Status

Email Verification — Verified

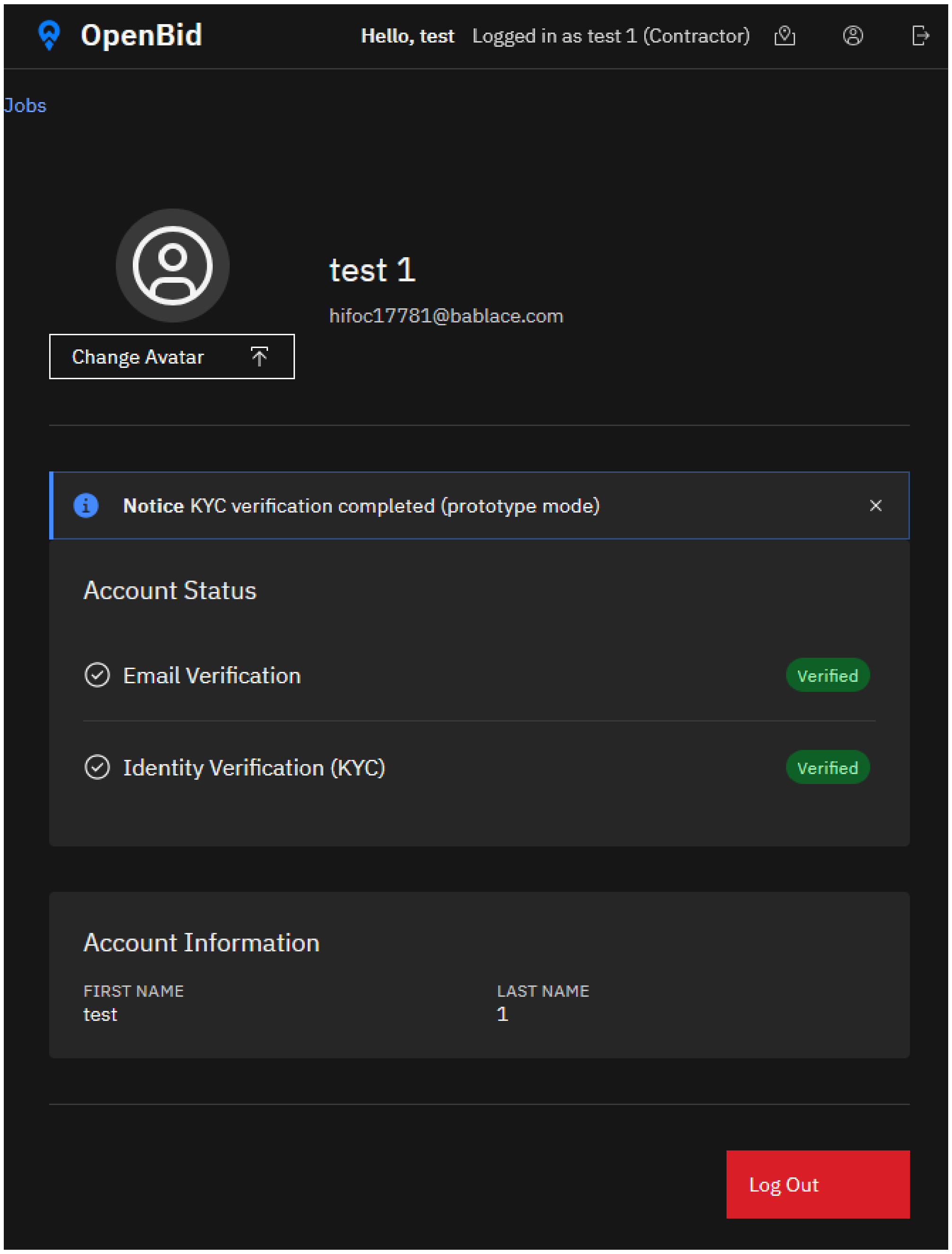
Identity Verification (KYC) — Pending

Account Information

FIRST NAME	LAST NAME
test	1

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

Profile Page - After KYC Verification


OpenBid

Hello, test Logged in as test 1 (Contractor)

Jobs

test 1

hifoc17781@bablace.com

Change Avatar

Notice KYC verification completed (prototype mode)

Account Status

Email Verification — Verified

Identity Verification (KYC) — Verified

Account Information

FIRST NAME	LAST NAME
test	1

Log Out

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

Firestore Persistence: Users, Jobs & Bids

Overview

This section documents the Firestore-backed persistence layer that stores **user**, **job**, and **bid** information. The primary implementation is the server adapter `server/src/adapters/db.real.js`, which uses the Firebase Admin SDK to interact with the `users`, `jobs`, and `bids` collections.

Class Diagrams

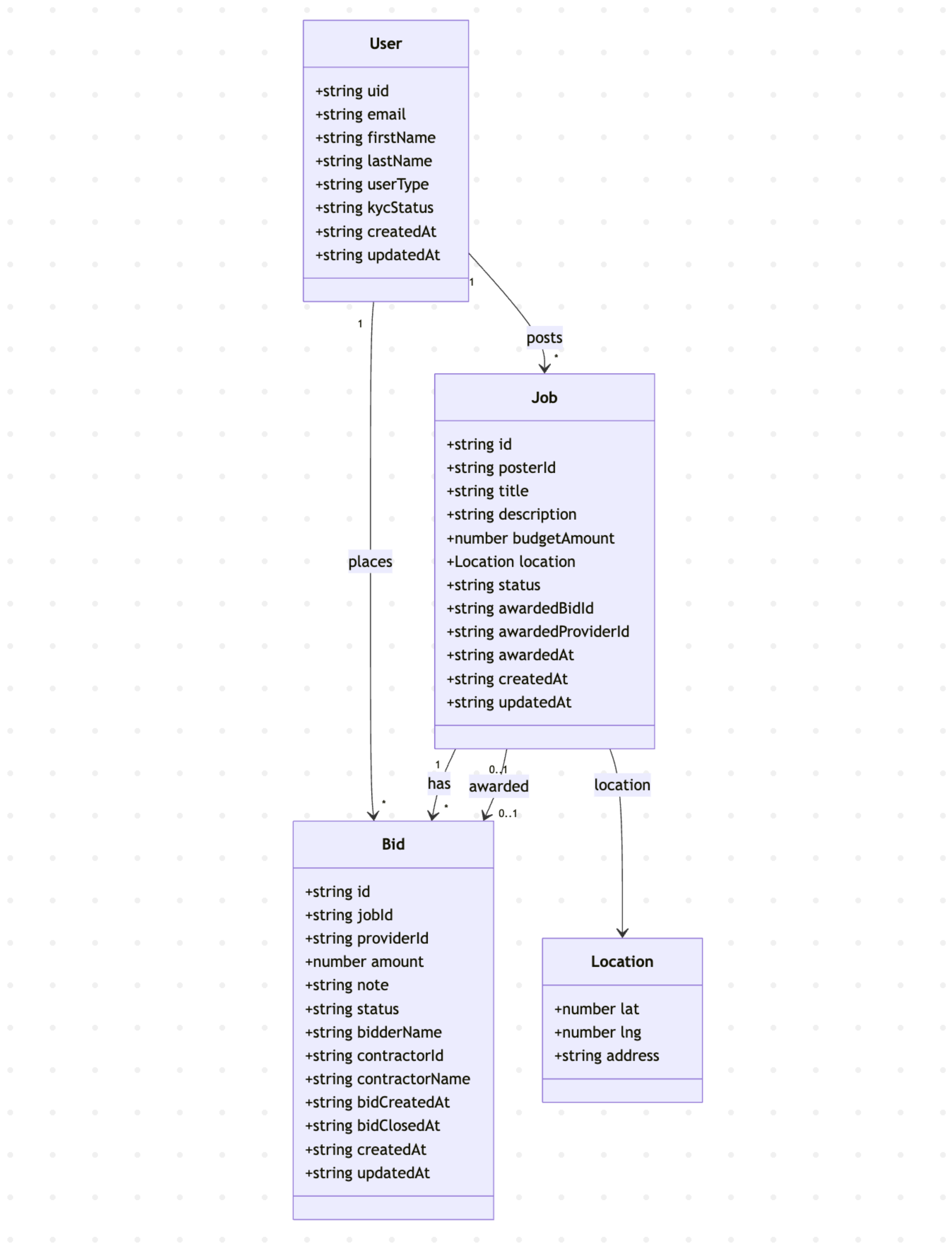


Figure 11: Users, Jobs, and Bids: Firestore Schema / Domain Entity Class Diagram

Firestore Collections & Domain Entities

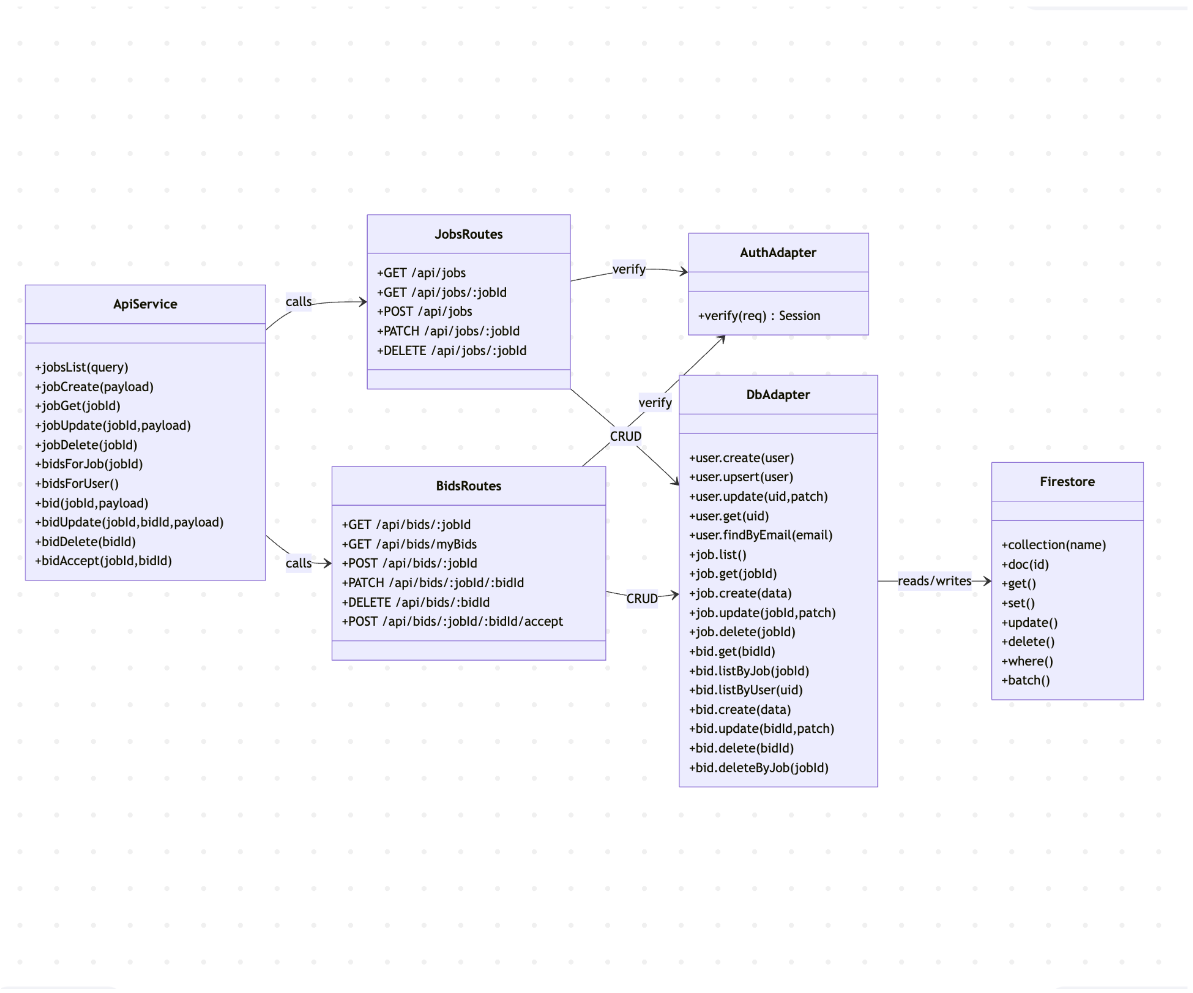


Figure 12: Jobs/Bids API Routers and Firestore DB Adapter Class Diagram

Server Routes and DB Adapter Structure

Sequence Diagrams

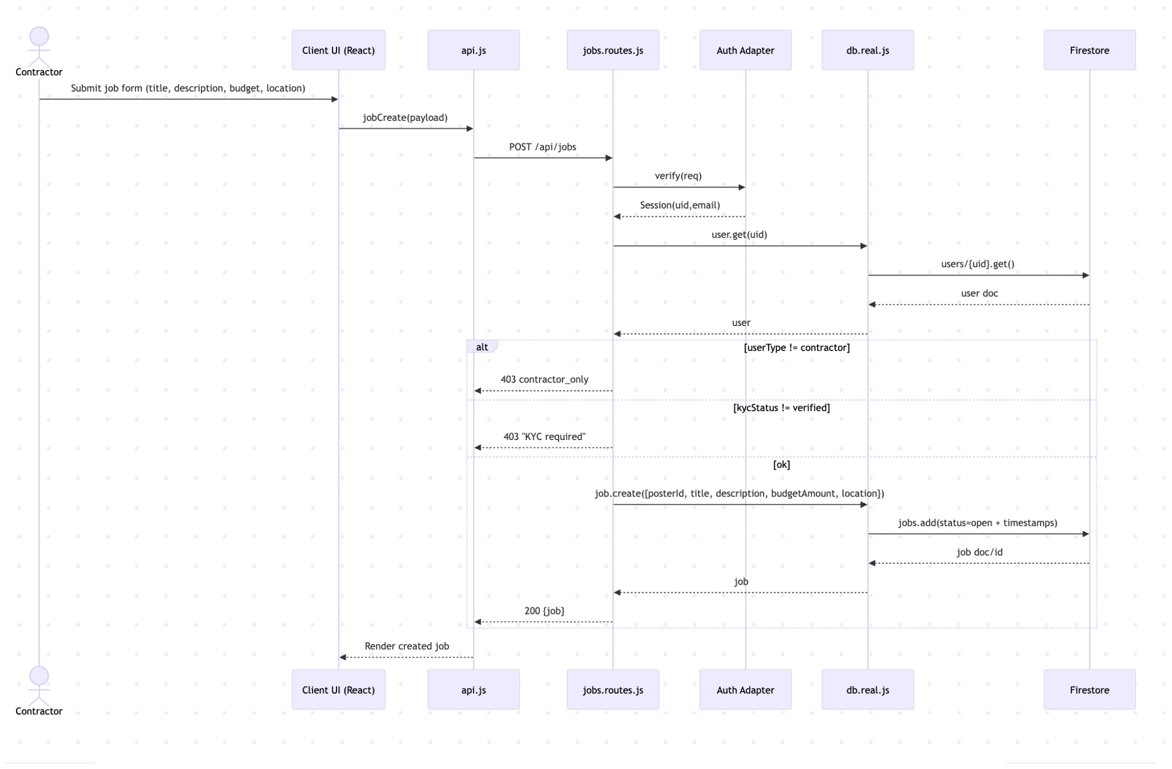


Figure 13: Sequence Diagram: Contractor Creates a Job

Create Job (Contractor) Flow Description:

1. Contractor submits job details from the client UI.
2. Client calls `api.jobCreate()` which sends `POST /api/jobs`.
3. Server verifies the session and confirms the user is a contractor with `kycStatus=verified`.
4. Server writes a new job document to Firestore via `db.job.create()`.
5. Server returns the created job to the client for immediate rendering.

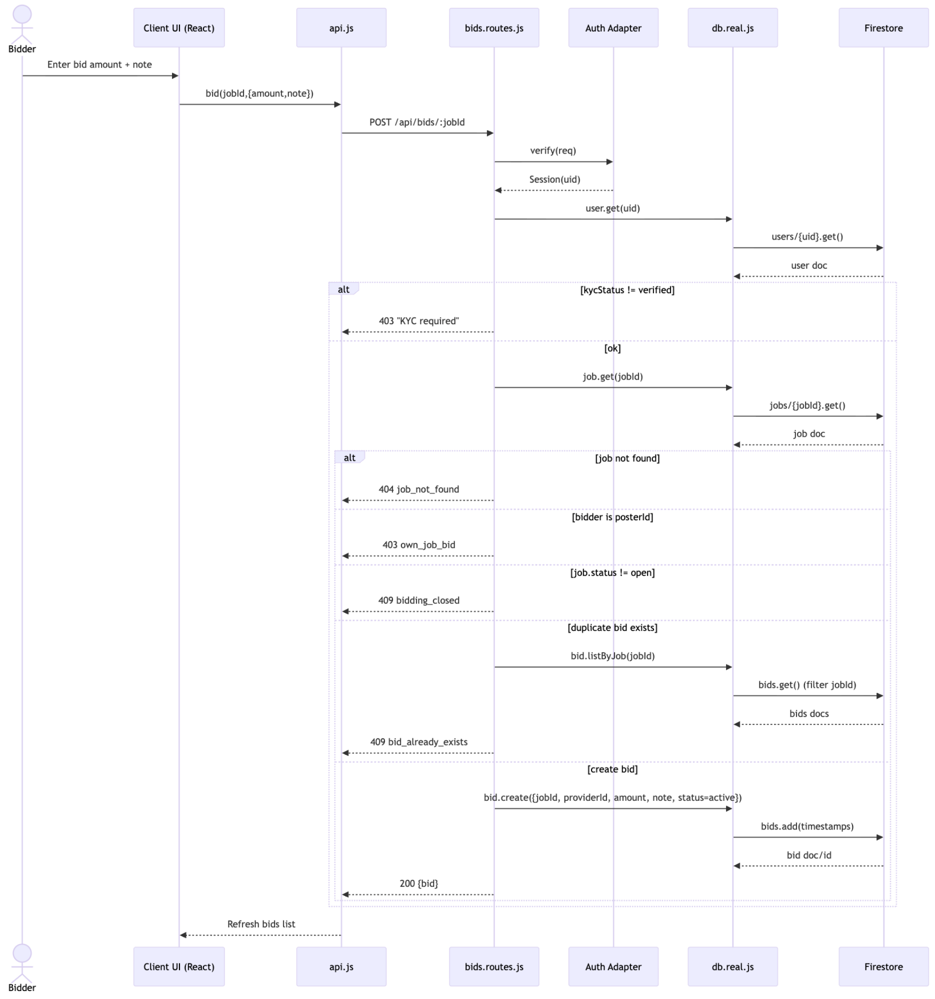


Figure 14: Sequence Diagram: Bidder Places a Bid on a Job

Place Bid (Bidder) Flow Description:

1. Bidder enters bid amount and note on a job detail page.
2. Client calls `api.bid()` which sends `POST /api/bids/:jobId`.
3. Server verifies the session and confirms the bidder's `kycStatus=verified`.
4. Server loads the job and validates: job exists, job is open, bidder is not the poster, and no existing bid exists.
5. Server creates a bid document in Firestore via `db.bid.create()` and returns the bid to the client.

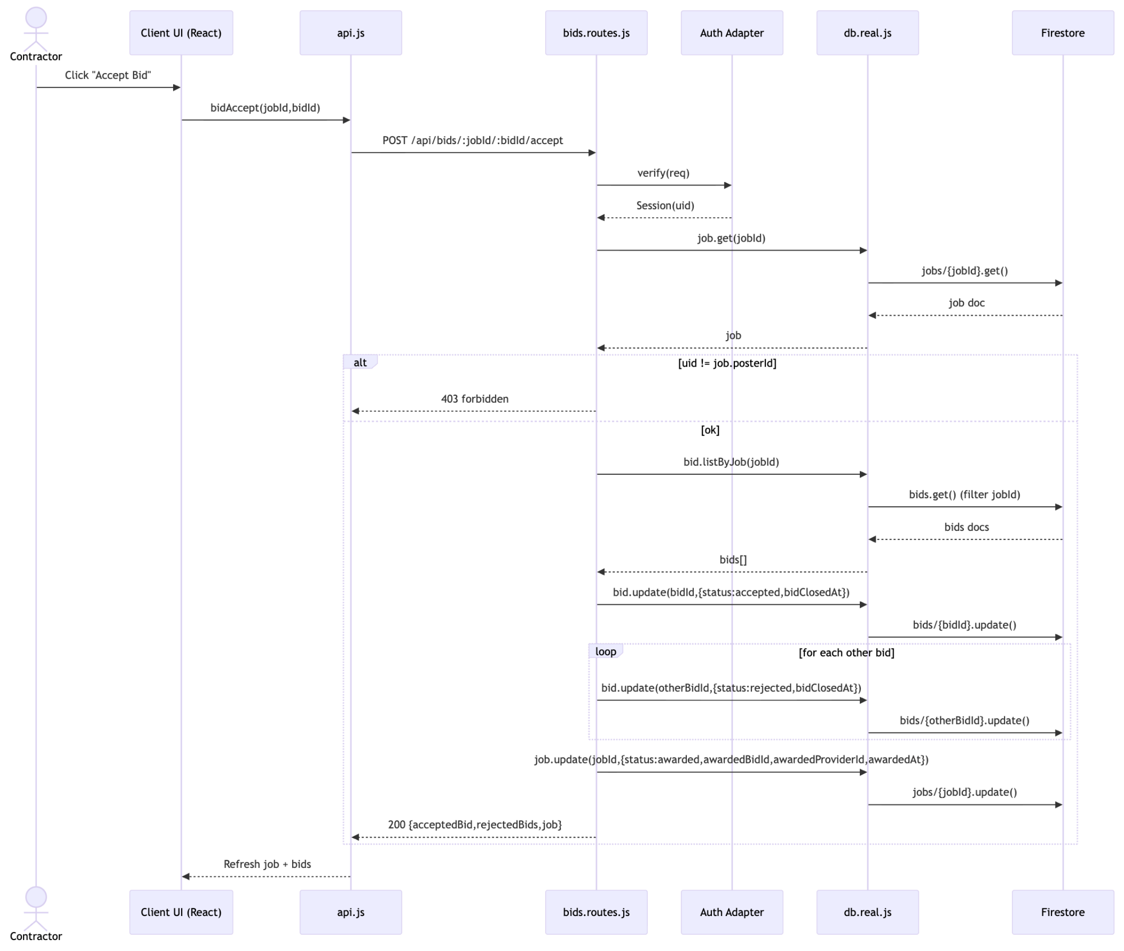


Figure 15: Sequence Diagram: Contractor Accepts a Bid and Awards the Job

Accept Bid (Contractor Awards Job)

Activity Diagrams

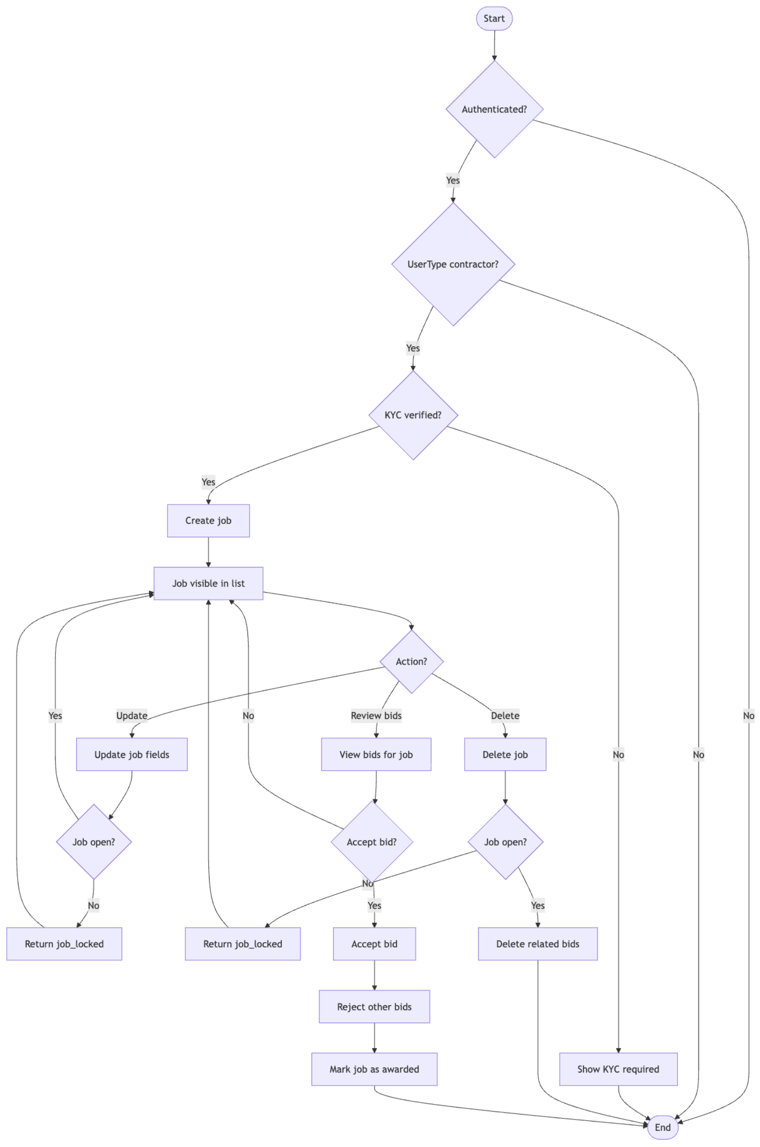


Figure 16: Activity Diagram: Job Lifecycle (Create, Update, Delete, Award)

Job Lifecycle Workflow

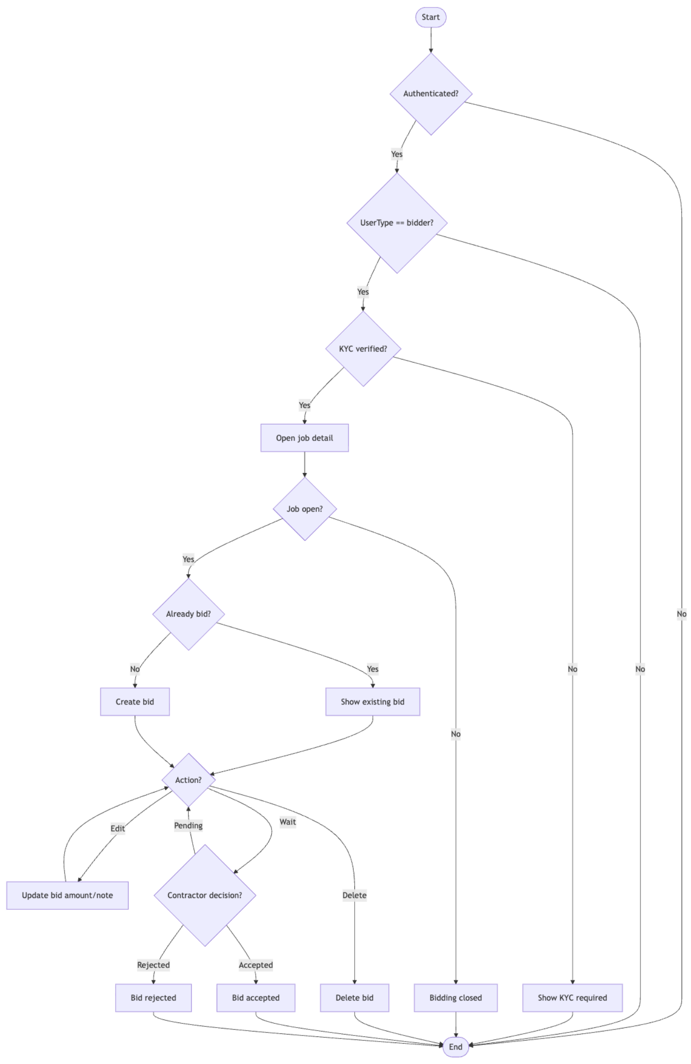


Figure 17: Activity Diagram: Bid Lifecycle (Place, Edit, Delete, Accept/Reject)

Bid Lifecycle Workflow

Component Summary (Jobs & Bids + Firestore)

Component	Type	Technology	Responsibility
`server/src/adapters/db.real.js`	Adapter	Firebase Admin SDK (Firestore)	CRUD for users, jobs, bids; timestamps; cascading deletes for job bids
`server/src/adapters/db.mock.js`	Adapter	In-memory	Prototype/mock persistence for local development and tests

Component	Type	Technology	Responsibility
`server/src/routes/jobs.routes.js`	Express Router	Node.js, Express	Jobs endpoints: list/get/create/update/delete; contractor/KYC enforcement
`server/src/routes/bids.routes.js`	Express Router	Node.js, Express	Bids endpoints: list-by-job, list-my-bids, create/update/delete, accept bid
`client/src/services/api.js`	Client Service	Fetch API	HTTP client for `/api/jobs` and `/api/bids` calls
`client/src/pages/JobList.jsx`	React Page	React	Browse jobs, map/filtering, navigate to job details
`client/src/pages/JobDetail.jsx`	React Page	React	View job + bids, place bid (bidder), accept bid (contractor)
`client/src/pages/MyBids.jsx`	React Page	React	List and manage bidder's submitted bids

Updated Backlog with Design Stories (Future: Reviews & Portfolio)

1. **DS-REV-001: Design Review Data Model** (High, 3 pts)
   Define review schema (reviewerId, revieweeId, jobId, bidId, rating, comment, timestamps). Decide write rules: only allow reviews after job completion and escrow payout. Enforce one review per party per job.
2. **DS-REV-002: Design Review Aggregation & Reputation** (Medium, 3 pts)
   Design aggregate rating fields on user documents (avg rating, count) and the update strategy (transaction, Cloud Function trigger, or server-side recompute). Define abuse mitigation (minimum comment length, flagging).
3. **DS-PORT-001: Design Provider Portfolio Model** (High, 3 pts)
   Define portfolio entry schema (providerId, title, description, tags, links/images, createdAt). Decide storage for images (Cloud Storage) and references in Firestore.
4. **DS-PORT-002: Design Portfolio UI and Access Rules** (Medium, 3 pts)
   Design profile/portfolio views and editing flows. Restrict write access to the owner; allow public read for awarded-job providers and verified accounts as configured.

Preliminary Test Coverage (Jobs & Bids)

The following test coverage report was generated using Jest (server) with the `-coverage` flag. This report focuses on the Jobs, Bids, and Firestore persistence components.

Server (Jest) Test Results Summary

Command: `cd server && npm test - -coverage`

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

Coverage by Component (Jobs & Bids + Firestore) – Server

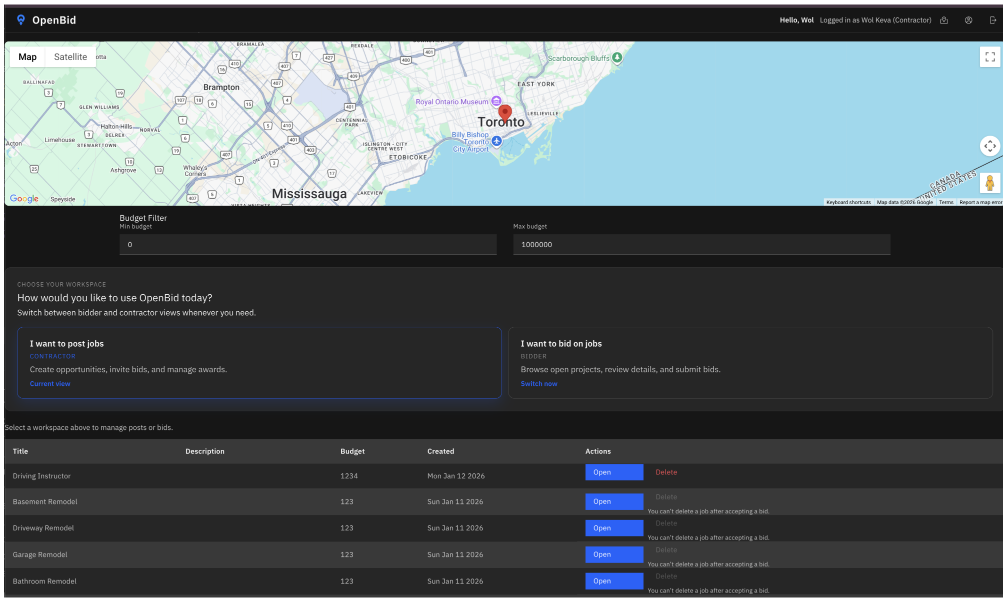
File	% Stmts	% Branch	% Funcs	% Lines	
`jobs.routes.js`	64.80	60.86	77.77	71.81	
`bids.routes.js`	42.14	29.07	40.00	44.96	
`db.real.js` (Jobs/Bids)	64.23	43.10	71.42	72.41	
Overall	53.73	46.20	57.14	56.33	

Analysis – Server

- **Jobs Routes:** Good coverage (71.81% lines) covering job list/create/update/delete constraints.
- **Bids Routes:** Lower coverage (44.96% lines) due to untested branches around bidding edge cases and accept flow variations.
- **Database Adapter:** Solid coverage (72.41% lines) for Firestore operations on jobs and bids, including timestamps and updates.
- **Gaps Identified:** Branch coverage remains moderate due to access-control and error-path permutations not fully exercised.

GUI Implementation Screenshots

Job List Page



Budget Filter

Min budget: 0

Max budget: 1000000

CHOOSE YOUR WORKSPACE

How would you like to use OpenBid today?

Switch between bidder and contractor views whenever you need.

I want to post jobs
CONTRACTOR
Create opportunities, invite bids, and manage awards.
Current view

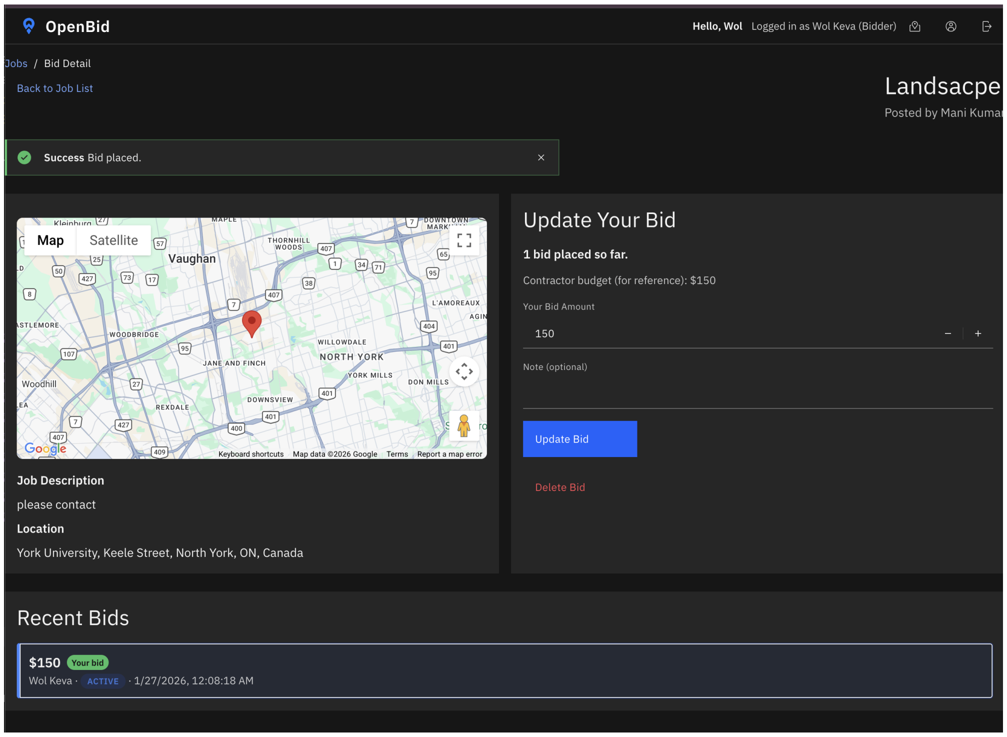
I want to bid on jobs
BIDDER
Browse open projects, review details, and submit bids.
Switch now

Select a workspace above to manage posts or bids.

Title	Description	Budget	Created	Actions
Driving Instructor		1234	Mon Jan 12 2026	Open Delete
Basement Remodel		123	Sun Jan 11 2026	Open Delete You can't delete a job after accepting a bid.
Driveway Remodel		123	Sun Jan 11 2026	Open Delete You can't delete a job after accepting a bid.
Garage Remodel		123	Sun Jan 11 2026	Open Delete You can't delete a job after accepting a bid.
Bathroom Remodel		123	Sun Jan 11 2026	Open Delete You can't delete a job after accepting a bid.

Figure 18: Job List page showing open jobs and filters

Job Detail + Bidding Page



OpenBid

Hello, Wol Logged in as Wol Keva (Bidder)

Jobs / Bid Detail

Back to Job List

Landsacpe

Posted by Mani Kuma

Success Bid placed. ×

Job Description

please contact

Location

York University, Keele Street, North York, ON, Canada

Update Your Bid

1 bid placed so far.

Contractor budget (for reference): $150

Your Bid Amount

150

Note (optional)

Update Bid

Delete Bid

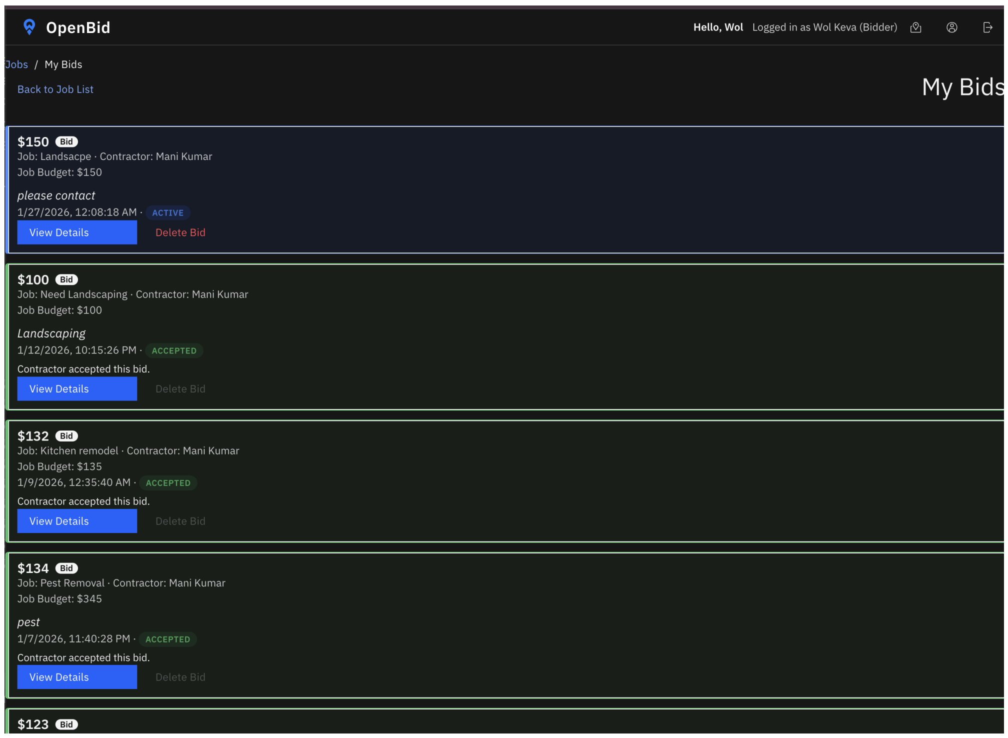
Recent Bids

$150 Your bid

Wol Keva · ACTIVE · 1/27/2026, 12:08:18 AM

Figure 19: Job Detail page showing job information and the bidding panel

My Bids Page



My Bids

$150 Bid
Job: Landsacpe · Contractor: Mani Kumar
Job Budget: $150

please contact
1/27/2026, 12:08:18 AM · ACTIVE

$100 Bid
Job: Need Landscaping · Contractor: Mani Kumar
Job Budget: $100

Landscaping
1/12/2026, 10:15:26 PM · ACCEPTED
Contractor accepted this bid.

$132 Bid
Job: Kitchen remodel · Contractor: Mani Kumar
Job Budget: $135
1/9/2026, 12:35:40 AM · ACCEPTED
Contractor accepted this bid.

$134 Bid
Job: Pest Removal · Contractor: Mani Kumar
Job Budget: $345

pest
1/7/2026, 11:40:28 PM · ACCEPTED
Contractor accepted this bid.

$123 Bid

Figure 20: My Bids page showing bidder's active bids and management actions

[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