DIGT2107: Practice of Software Development

Project Iteration 4: Full Documentation, Prototype, Presentation

OpenBid

Team 1: Tyler Mani Yanness Alaister

Due: November 30, 2025

Course: DIGT2107 – Fall Term 2025 | **Instructor:** Dr. May Haidar

1 Introduction

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

Document Overview This document outlines the deliverables for Iteration 4- focusing on improving the inital prototype, adding additional documentation for usability, adding tests as required, and brute force testing the prototype. The main highlight of this iteration is testing and fixing a working prototype. Goals: update core functionality based on feedback and testing, add additional unit tests as required, and update the documentation with the requirements for this iteration.

2 Iteration Goals and Objectives

- Update core flows: **auth + Duo 2FA** and **KYC gate** (stubbed via Stripe Identity sandbox token) in the prototype.
- **Post a job**, **browse on map**, **place a bid** core functionality should be fully implemented in the prototype.
- Develop and run additional unit tests mapped to requirements (frontend component tests and backend handler tests) *as required.
- Update documentation to ensure ease of setup and usability while adhering to the iteration requirements.
- Deliver a prototype demonstrating requirement ↔ test case links.

3 Functional and Non-Functional Requirements

3.1 Functional Requirements

Note that all prototype features will also be included in the production app (with potential modifications).

3.1.1 Prototype Features (Focus for this iteration)

1. **Account & Identity:** Users shall be able to sign up and sign in via email+password. Mandatory KYC must be completed before Contracters can bid or Posters can create jobs.
2. **Contractor Profiles:** Contractors can create and update their profiles with basic information.
3. **Discovery:** Contracters shall browse jobs in a list view.
4. **Bidding:** Contractors shall place, edit, and withdraw bids (amount, note, ETA).

3.1.2 Production Features

1. **Updated Account & Identity:** Users shall be able to sign up and sign in via email+password (authentication to be handled by firebase auth). Mandatory KYC must be completed before Contracters can bid or Posters can create jobs.

2. **Updated Contractor Profiles:** Contractors shall create, update, and delete professional profiles with details including skills, description, photos, rate (fixed/hourly), availability, and location (map pin/address).
3. **Updated Discovery:** Contractors shall browse jobs in list and map views, with filters by radius, category/skill, budget range, and availability date.
4. **Bidding:** Contractors shall place, edit, and withdraw bids (amount, note, ETA). Posters cant accept a winning bid.
5. **Two-Factor Authentication (2FA):** Users shall confirm sign-in with Duo 2FA for sensitive actions (login, payouts).
6. **Payments (Escrow):** When a Contractor accepts an offer, the Poster shall fund an escrow hold. On job completion, funds shall be captured and paid out to the Contractor via Stripe Connect.
7. **Messaging:** Contractors and Bidders shall exchange secure messages within the context of an offer, with support for attachments and report/block controls.
8. **Reviews:** Both parties shall leave ratings and text reviews after completion.
9. **Safety Score:** The system shall compute a location safety score (0–100) and apply graduated friction (tips, daylight default, verified-only, or manual review).
10. **Admin:** Admins shall review reports, manage categories, and handle disputes via a basic dashboard.

3.2 Non-Functional Requirements

Non-functional requirements will be implemented in the production application, not in the prototype.

- **Performance:** Map/list browse and search shall return results in $\leq$ 500 ms p95 for target metro; prototype may use simplified filters.
- **Usability:** Mobile-first, accessible (WCAG 2.1 AA where practical); clear copy for KYC/2FA and escrow steps.
- **Security:** All traffic over TLS; short-lived JWTs; server-side validation; reCAPTCHA on signup/post/bid.
- **Scalability:** Stateless API (Node/Express) behind Firebase/Cloud Run; Firestore as primary data store; storage via Firebase Cloud Storage.
- **Reliability:** Error tracking via Sentry; observability with OpenTelemetry; automated CI checks on PRs.
- **Privacy:** Approximate location shown pre-accept; exact address released to the accepted job assignee only; phone masking post-accept.

4 Requirement to Test Case Traceability

Tables below map each requirement to the test cases that validate it. Coverage status will be updated as testing completes.

Functional Requirements (FR)

Req ID	Requirement Description	Test Case IDs	Coverage
FR-001	System shall allow authenticated users to **post jobs** with title, description, budget, photos, and location.	TC-001, TC-002	Partially Covered
FR-002	System shall allow providers to **browse jobs on a map** with basic filters (radius, category).	TC-003a-d, TC-004	Partially Covered
FR-003	System shall allow providers to **place bids** on open jobs; posters can view and accept a bid.	TC-005, TC-006	Partially Covered
FR-004	System shall enforce **KYC for all users** before posting or bidding.	TC-007, TC-008a-b, TC-009a-b, TC-015a-e	Covered
FR-005	System shall support **user authentication** with **Duo 2FA** for sensitive actions.	TC-008	Covered
FR-006	System shall provide **in-thread messaging** per job after acceptance.	TC-010	Planned
FR-007	System shall record **ratings and reviews** after job completion.	TC-011	Planned
FR-008	System must create Firebase accounts and keep users out until they click the email verification link.	TC-014a-d	Covered
FR-009	System must let a signed-in person switch between bidder and contractor and keep their requirement flags accurate.	TC-015a-b	Covered
FR-010	System must save the session in local storage and log the user out after two minutes with no activity.	TC-016a-c	Covered
FR-011	Only KYC-approved contractors can create, edit, or delete their own jobs in Firestore.	TC-017a-i	Covered
FR-012	System must block self-bids and lock both the job and bids once a contractor accepts an offer.	TC-018a-b	Covered
FR-013	System shall allow users to **manage their profile** including avatar upload and personal information.	TC-016a-d	Covered

Non-Functional Requirements (NFR)

Req ID	Requirement Description	Test Case IDs	Coverage
NFR-001	**Performance:** Search/browse should return results within 800 ms P95 for a metro with 5k open jobs (stub data).	TC-011	Planned
NFR-002	**Security:** Only KYC-verified users can hit write endpoints for jobs/bids.	Covered	
NFR-003	**Usability:** Map view and list view maintain accessible contrast and keyboard navigation for core actions.	TC-013	Planned

Notes

- Each requirement is validated by one or more test cases. Gaps are flagged as *Planned*.
- NFR coverage status is informative only (not required for grading), but we track it to guide future work.

5 Detailed Test Case Descriptions

For brevity we include the highest-priority cases now; the full catalog will live in the repo under `tests/`.

TC-001

Test Case ID	TC-001
Title	Post Job - Valid Inputs
Requirement	FR-001
Preconditions	User is logged in, KYC status = verified.
Steps	1. Navigate to `/new-job`. 2. Enter valid title, description, budget, and pick a map location (Google Maps widget). 3. Upload a photo (sample file). 4. Click `Post`.
Expected Result	Job document is created in Firestore; UI redirects to Job Detail.
Actual Result	To be filled after execution.
Status	Planned
Priority	High

TC-002

Test Case ID	TC-002
Title	Post Job - Validation Errors
Requirement	FR-001
Preconditions	Logged in, KYC verified.
Steps	1. Navigate to `/new-job`. 2. Leave title empty; click `Post`.
Expected Result	Client-side validation shows error; no write occurs.
Actual Result	To be filled after execution.
Status	Planned
Priority	High

TC-003a

Test Case ID	TC-003a
Title	The haversineFormulakm function accurately calcuates distance between 2 coordinates.
Requirement	FR-002
Preconditions	Valid coordinates are input.
Steps	1. Coordinates are input into the function. 2. Function calculates the distance between the coordinates. 3. Function returns the distance between the coordinates.
Expected Result	Returns distance in km of the 2 coordinates, returns infinity when there is invalid input.
Actual Result	Passed - Returns distance accurately within a specific tolerance. Coordinates tested are up to 250km apart.
Status	Passed
Priority	High

TC-003b

Test Case ID	TC-003b
Title	The isValidCoords formula accurately determines whether the given inputs are valid latitude and longitude values.
Requirement	FR-002
Preconditions	None.
Steps	1. Coordinates are input into the function. 2. Function checks input for validity then checks whether the input values are valid coordinates. 3. Function returns whether the coordinates are valid.
Expected Result	Returns true or false to depending on whether the input is valid coordinates.
Actual Result	Passed - accurately surmises whether the input coordinates are valid.
Status	Passed
Priority	High

TC-003c

Test Case ID	TC-003c
Title	The degToRad function accurately converts a number in degrees to radians.
Requirement	FR-002
Preconditions	Input should be a number.
Steps	1. a number in degrees is input into the function. 2. Function checks input for validity then returns the input value in radians.
Expected Result	Returns the input number converted into radians if the input is valid, returns infinity otherwise.
Actual Result	Passed - accurately converts a valid input to radians and returns infinity otherwise.
Status	Passed
Priority	High

TC-003d

Test Case ID	TC-003d
Title	The createMap function loads a google map. (uses api mocking)
Requirement	FR-002
Preconditions	The lat, lng, ref, and apikey inputs are valid. The lat and lng and tested for validness before the function call and the logic outside the function affirms that apiKey and ref are valid.
Steps	1. The google maps api loader is initialized. 2. The markers array input is filtered for valid markers or is set to an empty array if the markers input is bad. 3. The google maps api loader is used to load the api. 4. A new google map is created and valid markers are atached to the map.
Expected Result	Creates a map with valid markers attached to it.
Actual Result	Passed - creates a map with valid markers attached to it, invalid markers are discarded and a bad markers input is gracefully handled.
Status	Passed
Priority	High

TC-004

Test Case ID	TC-004
Title	Map Browse - Category Filter
Requirement	FR-002
Preconditions	User is on map browse page with multiple job categories available.
Steps	1. Select a specific job category from the filter dropdown. 2. Observe the jobs displayed on the map.
Expected Result	Only jobs of selected category are shown.
Actual Result	To be filled after execution.
Status	Planned
Priority	Medium

TC-005

Test Case ID	TC-005
Title	Place Bid - Valid
Requirement	FR-003
Preconditions	Provider is logged in, KYC verified, job is open.
Steps	1. Navigate to job detail page. 2. Enter valid bid amount and optional message. 3. Click "Submit Bid" button.
Expected Result	Bid document created; poster sees bid in Job Detail.
Actual Result	To be filled after execution.
Status	Planned
Priority	High

TC-006

Test Case ID	TC-006
Title	Accept Bid - Poster Flow
Requirement	FR-003
Preconditions	Contractor is logged in, has posted a job, and has received at least one bid.
Steps	1. Navigate to job detail page with bids. 2. Select a bid to accept. 3. Click "Accept Bid" button. 4. Confirm acceptance in modal dialog.
Expected Result	Job status transitions to `awarded`; winning bid marked `accepted`.
Actual Result	To be filled after execution.
Status	Planned
Priority	High

TC-007

Test Case ID	TC-007
Title	KYC Gate - Block Unverified Writes
Requirement	FR-004, NFR-002
Preconditions	User KYC status = pending.
Steps	1. Attempt to post a new job through UI. 2. Attempt to place a bid on an existing job. 3. Attempt direct API calls to create job/bid endpoints.
Expected Result	Attempts to post job or bid are rejected by Firestore rules/API; UI shows KYC required.
Actual Result	To be filled after execution.
Status	Planned
Priority	High

TC-008a

Test Case ID	TC-008a
Title	KYC Verification - Create Stripe Verification Session
Requirement	FR-004
Preconditions	User is authenticated; KYC status = `pending`; real KYC mode enabled.
Steps	1. Send POST request to `/api/kyc/verification` with valid auth JWT token. 2. System calls Stripe Identity API to create verification session. 3. System stores session ID in user record. 4. System returns Stripe verification URL and session ID.
Expected Result	Returns Stripe verification URL (https://verify.stripe.com/*) and session ID (vs_*)
Actual Result	Passed - Returns mocked Stripe URL and session ID.
Status	Passed
Priority	High

TC-008b

Test Case ID	TC-008b
Title	KYC Verification - Unauthorized Access
Requirement	FR-004, NFR-002
Preconditions	User not authenticated or invalid token.
Steps	1. Send POST request to `/api/kyc/verification` without auth token or with invalid token.
Expected Result	Returns 401 Unauthorized with error message.
Actual Result	Passed - Returns 401 with `{error: "unauthorized"}`.
Status	Passed
Priority	High

TC-009a

Test Case ID	TC-009a
Title	KYC Status Check
Requirement	FR-004
Preconditions	User is authenticated.
Steps	1. Send GET request to `/api/kyc/status` with valid auth JWT token. 2. System retrieves user KYC status from database. 3. If verification session exists and status is `pending`, checks Stripe for updated status. 4. System returns KYC status (`pending`, `verified`, or `failed`).
Expected Result	Returns KYC status (pending, verified, or failed).
Actual Result	Passed - Returns valid status from database.
Status	Passed
Priority	High

TC-009b

Test Case ID	TC-009b
Title	KYC Status - Unauthorized Access
Requirement	FR-004, NFR-002
Preconditions	User not authenticated.
Steps	1. Send GET request to `/api/kyc/status` without auth token.
Expected Result	Returns 401 Unauthorized.
Actual Result	Passed - Returns 401 with error message.
Status	Passed
Priority	High

TC-010

Test Case ID	TC-010
Title	Messaging After Acceptance
Requirement	FR-006
Preconditions	Job has been awarded to a bidder; both parties are authenticated.
Steps	1. Navigate to awarded job detail page. 2. Enter message in the messaging thread. 3. Submit message. 4. Other party logs in and views the job detail page.
Expected Result	Parties can exchange messages in job thread; messages persist in Firestore.
Actual Result	To be filled after execution.
Status	Planned
Priority	Medium

TC-011

Test Case ID	TC-011
Title	Submit Review on Completion
Requirement	FR-007
Preconditions	Job has been marked as completed; user is authenticated as job poster or winning bidder.
Steps	1. Navigate to completed job detail page. 2. Click "Leave Review" button. 3. Enter rating (1-5 stars) and review text. 4. Submit review. 5. Attempt to submit a second review for the same job.
Expected Result	Review saved and visible on profile; duplicate review blocked.
Actual Result	To be filled after execution.
Status	Planned
Priority	Medium

TC-012

Test Case ID	TC-012
Title	Performance P95 - Map Query
Requirement	NFR-001
Preconditions	Test environment with 5,000 stub job records in database; performance monitoring enabled.
Steps	1. Load map view with default filters. 2. Change category filter to a specific job type. 3. Measure time from filter change to results rendering. 4. Repeat test 100 times to calculate P95 metric.
Expected Result	P95 end-to-end from filter change to results render ≤ 800 ms on stub dataset.
Actual Result	To be filled after execution.
Status	Planned
Priority	Low

TC-013

Test Case ID	TC-013
Title	Ruleset Audit - No Write Without Claims
Requirement	NFR-002
Preconditions	Firebase project with security rules deployed; test environment configured.
Steps	1. Authenticate as user without KYC verification. 2. Attempt to write to jobs collection. 3. Attempt to write to bids collection. 4. Verify rule rejection behavior.
Expected Result	Firestore rules reject writes missing `kycVerified == true`.
Actual Result	To be filled after execution.
Status	Planned
Priority	High

TC-014

Test Case ID	TC-014
Title	Accessibility - Keyboard Nav on Map/List
Requirement	NFR-003
Preconditions	Application running with jobs loaded in map and list views.
Steps	1. Navigate to map/list view page. 2. Use Tab key to navigate through UI elements. 3. Verify focus indicators are visible on all interactive elements. 4. Test keyboard operation of filters, job selection, and primary actions.
Expected Result	Tabbing reaches filters, list items, and primary actions; visible focus outlines present.
Actual Result	To be filled after execution.
Status	Planned
Priority	Low

TC-015a

Test Case ID	TC-015a
Title	Profile Page - Display KYC Pending Status
Requirement	FR-004
Preconditions	User authenticated, KYC status = `pending`.
Steps	1. Navigate to `/profile`. 2. Profile component renders and displays account status section. 3. System retrieves user KYC status from session.
Expected Result	KYC status displays "Pending" tag with appropriate styling.
Actual Result	Passed - Profile page correctly displays pending status.
Status	Passed
Priority	High

TC-015b

Test Case ID	TC-015b
Title	Profile Page - Display KYC Verified Status
Requirement	FR-004
Preconditions	User authenticated, KYC status = `verified`.
Steps	1. Navigate to `/profile`. 2. Profile component renders with verified KYC status.
Expected Result	KYC status displays "Verified" tag; no action buttons shown.
Actual Result	Passed - Profile page displays verified status correctly.
Status	Passed
Priority	High

TC-015c

Test Case ID	TC-015c
Title	Profile Page - Complete KYC Button Displayed
Requirement	FR-004
Preconditions	User authenticated, KYC status = `pending`.
Steps	1. Navigate to `/profile`. 2. System checks KYC status is not verified. 3. Profile component renders action buttons.
Expected Result	"Complete KYC" and "Refresh Status" buttons are visible.
Actual Result	Passed - Action buttons displayed for pending status.
Status	Passed
Priority	High

TC-015d

Test Case ID	TC-015d
Title	Profile Page - Initiate KYC Verification
Requirement	FR-004
Preconditions	User authenticated, KYC pending, production mode.
Steps	1. User clicks "Complete KYC" button. 2. System calls `/api/kyc/verification` endpoint. 3. System receives Stripe verification URL. 4. System opens URL in new browser tab.
Expected Result	Stripe Identity verification opens in new tab; notification displayed.
Actual Result	Passed - Verification URL opened correctly.
Status	Passed
Priority	High

TC-015e

Test Case ID	TC-015e
Title	Profile Page - Refresh KYC Status
Requirement	FR-004
Preconditions	User authenticated, KYC pending.
Steps	1. User clicks "Refresh Status" button. 2. System calls `/api/kyc/status` endpoint. 3. System receives updated status. 4. System updates UI and session with new status.
Expected Result	KYC status refreshed; UI updated with current status.
Actual Result	Passed - Status refresh works correctly.
Status	Passed
Priority	High

TC-016a

Test Case ID	TC-016a
Title	Profile Page - Upload Avatar Successfully
Requirement	FR-008 (User Profile Management)
Preconditions	User authenticated.
Steps	1. User clicks "Change Avatar" button. 2. User selects valid image file (PNG, < 5MB). 3. System uploads file to storage. 4. System updates user profile with avatar URL.
Expected Result	Avatar uploaded; preview displayed; success notification shown.
Actual Result	Passed - Avatar upload successful.
Status	Passed
Priority	High

TC-016b

Test Case ID	TC-016b
Title	Profile Page - Reject Large Avatar Files
Requirement	FR-008, NFR-002 (Security)
Preconditions	User authenticated.
Steps	1. User clicks "Change Avatar" button. 2. User selects image file larger than 5MB. 3. System validates file size before upload.
Expected Result	Upload rejected; error message "Image must be smaller than 5MB" displayed.
Actual Result	Passed - Large files rejected correctly.
Status	Passed
Priority	High

TC-016c

Test Case ID	TC-016c
Title	Profile Page - Handle Avatar Upload Error
Requirement	FR-008
Preconditions	User authenticated; network/server error occurs.
Steps	1. User selects valid image file. 2. System attempts upload to `/api/avatar/upload`. 3. Server returns error response.
Expected Result	User-friendly error message displayed; no profile update.
Actual Result	Passed - Error handled gracefully.
Status	Passed
Priority	High

TC-016d

Test Case ID	TC-016d
Title	Profile Page - Display Existing Avatar
Requirement	FR-008
Preconditions	User authenticated; avatar URL exists in profile.
Steps	1. Navigate to `/profile`. 2. System retrieves user avatar URL from database. 3. Profile component renders avatar image.
Expected Result	User's avatar displayed in profile header.
Actual Result	Passed - Existing avatar displayed correctly.
Status	Passed
Priority	High

TC-017

Test Case ID	TC-017a
Title	Duo 2FA Authentication Flow
Requirement	FR-005
Preconditions	User exists, Duo enabled.
Steps	1. POST `/api/auth/login` with valid credentials. 2. Follow `mfa.startUrl` to begin Duo: GET `/api/auth/duo/start?state=...`. 3. Simulate Duo callback: GET `/api/auth/duo/callback?state=...&code=allow-123`. 4. Finalize: POST `/api/auth/duo/finalize` with one-time code.
Expected Result	Login returns 202 with Duo start URL; callback redirects to `/login/finish?code=...`; finalize returns session JSON.
Actual Result	Passed – Covered by Jest `tests/auth.duo.test.js`.
Status	Passed
Priority	High

TC-018a

Test Case ID	TC-018a
Title	Firebase Signup - Provision Account and Send Verification Email
Requirement	FR-008
Preconditions	Email is brand new inside the Firebase project; Identity Toolkit is online.
Steps	1. Send POST `/api/auth/signup` with first/last name, valid email, password, confirmPassword. 2. Let Firebase create the account and queue the verification email. 3. Read the JSON response from our API.
Expected Result	HTTP 201 with sanitized user data, email + KYC both `pending`, and Firebase verification email requested.
Actual Result	Passed – Confirmed in `server/src/routes/__tests__/auth.integration.real.routes.test.js`.
Status	Passed

TC-018b

Test Case ID	TC-018b
Title	Firebase Signup - Reject Invalid Credentials
Requirement	FR-008
Preconditions	Same Firebase project as TC-014a; email still unused.
Steps	1. POST `/api/auth/signup` with a broken email or a password shorter than eight characters. 2. Watch the server reject the payload before any Firebase call.
Expected Result	HTTP 400 with a clear error; Firebase signup helper never runs.
Actual Result	Passed – Shown in the same Jest suite for real adapters.
Status	Passed
Priority	High

TC-018c

Test Case ID	TC-018c
Title	Firebase Login - Verified User Receives Session Tokens
Requirement	FR-008
Preconditions	User already exists in Firebase with both email and KYC marked `verified`.
Steps	1. POST `/api/auth/login` with the correct email and password. 2. Inspect the response for session + requirement flags.
Expected Result	HTTP 200 with sanitized user, Firebase ID + refresh tokens, and `requirements = {emailVerified: true, kycVerified: true}`.
Actual Result	Passed – Assertions live in `auth.integration.real.routes.test.js`.
Status	Passed
Priority	High

TC-018d

Test Case ID	TC-018d
Title	Firebase Login - Reject Wrong Passwords
Requirement	FR-008
Preconditions	User exists in Firebase; test double forces Identity Toolkit to return `INVALID_PASSWORD`.
Steps	1. POST `/api/auth/login` with the wrong password. 2. Capture HTTP status + body.
Expected Result	HTTP 401 with `{error: "invalid credentials"}`; no session issued.
Actual Result	Passed – Covered in `auth.integration.real.routes.test.js`.
Status	Passed
Priority	High

TC-019a

Test Case ID	TC-019a
Title	Role Switch - Authorization Required
Requirement	FR-009
Preconditions	User is signed in and has a valid Firebase ID token.
Steps	1. PATCH `/api/auth/role` without the Authorization header. 2. Repeat with `Authorization: Bearer <idToken>` and payload `{role: "contractor"}`.
Expected Result	Unauthorized request returns 401; authorized request returns 200 with updated `userType` and refreshed requirement flags.
Actual Result	Passed – Exercised via Jest role-switch integration test.
Status	Passed
Priority	Medium

TC-019b

Test Case ID	TC-019b
Title	Auth Me - Return Sanitized User Context
Requirement	FR-009
Preconditions	Contractor user plus valid Firebase ID token.
Steps	1. GET `/api/auth/me` without Authorization header and confirm 401. 2. Repeat with Authorization header; inspect payload for sanitized user fields.
Expected Result	Unauthorized call blocked; authorized call returns sanitized `user` payload (no `passwordHash`) plus metadata.
Actual Result	Passed – Documented in `auth.integration.real.routes.test.js`.
Status	Passed
Priority	Medium

TC-020a

Test Case ID	TC-020a
Title	Session Service - Persist and Notify Subscribers
Requirement	FR-010
Preconditions	Browser environment available (Vitest JSDOM); subscribers registered.
Steps	1. Call `setSession` with a real Firebase-authenticated user payload. 2. Verify `subscribeSession` listener triggered. 3. Inspect `localStorage` for serialized session data.
Expected Result	Session stored under `openbid_session`; only real-user metadata persisted; listeners invoked once.
Actual Result	Passed – Covered by `client/src/__tests__/session.test.js`.
Status	Passed
Priority	Medium

TC-020b

Test Case ID	TC-020b
Title	Session Service - setUser Preserves Requirements
Requirement	FR-010
Preconditions	Existing session with requirement flags stored.
Steps	1. Initialize session via `setSession`. 2. Invoke `setUser` with new user payload and explicit requirements. 3. Query `getRequirements` and `getSession`.
Expected Result	Updated user persisted; requirements reflect provided override (email + KYC flags).
Actual Result	Passed – Validated in `session.test.js`.
Status	Passed
Priority	Medium

TC-020c

Test Case ID	TC-020c
Title	Session Service - Inactivity Monitor Auto Logout
Requirement	FR-010
Preconditions	Session established; Vitest fake timers.
Steps	1. Call `startInactivityMonitor` with spy callback. 2. Advance timers to just before the 2-minute limit; ensure no callback. 3. Advance timers past the limit; ensure callback fires and cleanup works.
Expected Result	Timeout callback executes exactly once after 2 minutes of inactivity, indicating logout.
Actual Result	Passed – Covered by Vitest case in `session.test.js`.
Status	Passed
Priority	Medium

TC-021a

Test Case ID	TC-021a
Title	Job Create - Verified Contractor Path
Requirement	FR-011
Preconditions	Contractor account with `kycStatus = verified` and a valid Firebase ID token.
Steps	1. POST `/api/jobs` with the Authorization header plus title, description, budget, and location. 2. Read the JSON response.
Expected Result	HTTP 200 with a job whose `posterId` matches the contractor and status `open`.
Actual Result	Passed – Verified in `jobs.integration.real.routes.test.js`.
Status	Passed
Priority	High

TC-021b

Test Case ID	TC-021b
Title	Job Create - Bidder Rejected
Requirement	FR-011
Preconditions	Bidder account logged in with Firebase token.
Steps	1. POST `/api/jobs` using the bidder's token and minimal payload.
Expected Result	HTTP 403 with `{error: "contractor_only"}`; no job written.
Actual Result	Passed – Documented in `jobs.integration.real.routes.test.js`.
Status	Passed
Priority	High

TC-021c

Test Case ID	TC-021c
Title	Job Create - KYC Pending Block
Requirement	FR-011
Preconditions	Contractor account exists but `kycStatus = pending`.
Steps	1. POST `/api/jobs` with the contractor's token while KYC is still pending.
Expected Result	HTTP 403 with `{error: "KYC required"}`; job not created.
Actual Result	Passed – See "enforces KYC verification" Jest test.
Status	Passed
Priority	High

TC-021d

Test Case ID	TC-021d
Title	Job Update - Owner Edits Open Job
Requirement	FR-011
Preconditions	Contractor owns an `open` job and is signed in.
Steps	1. PATCH `/api/jobs/:jobId` with a new title and budget. 2. Read response body.
Expected Result	HTTP 200; returned job reflects updated fields; timestamps preserved.
Actual Result	Passed – "updates an open job" Jest test.
Status	Passed
Priority	Medium

TC-021e

Test Case ID	TC-021e
Title	Job Update - Foreign Owner Forbidden
Requirement	FR-011
Preconditions	Two contractors exist; the job belongs to contractor A; contractor B is signed in.
Steps	1. Contractor B PATCHes `/api/jobs/:jobId` owned by contractor A.
Expected Result	HTTP 403 with `{error: "forbidden"}`; job unchanged.
Actual Result	Passed – "forbids editing another contractor's job" Jest test.
Status	Passed
Priority	Medium

TC-021f

Test Case ID	TC-021f
Title	Job Update - Locked Status
Requirement	FR-011
Preconditions	Contractor owns a job already marked `awarded` or `completed`.
Steps	1. Try to PATCH the locked job.
Expected Result	HTTP 409 with `{error: "job_locked"}`.
Actual Result	Passed – "returns job_locked when status is no longer open" Jest test.
Status	Passed
Priority	Medium

TC-021g

Test Case ID	TC-021g
Title	Job Delete - Owner Removes Open Job
Requirement	FR-011
Preconditions	Contractor owns an open job.
Steps	1. DELETE `/api/jobs/:jobId` with the owner's token. 2. Try reading the job from Firestore.
Expected Result	HTTP 204; subsequent read returns null.
Actual Result	Passed – "removes an open job owned by contractor Jane Doe" Jest test.
Status	Passed
Priority	Medium

TC-021h

Test Case ID	TC-021h
Title	Job Delete - Foreign Owner Forbidden
Requirement	FR-011
Preconditions	Job owned by contractor A; contractor B signed in.
Steps	1. Contractor B DELETEs `/api/jobs/:jobId`.
Expected Result	HTTP 403 with `{error: "forbidden"}`; job remains.
Actual Result	Passed – "forbids deleting someone else's job" Jest test.
Status	Passed
Priority	Medium

TC-021i

Test Case ID	TC-021i
Title	Job Delete - Locked Status
Requirement	FR-011
Preconditions	Contractor owns a job that is no longer `open`.
Steps	1. Try to DELETE the locked job.
Expected Result	HTTP 409 with `{error: "job_locked"}`.
Actual Result	Passed – "returns job_locked when job status is not open" Jest test.
Status	Passed
Priority	Medium

TC-022a

Test Case ID	TC-022a
Title	Bid Create - Contractor Cannot Bid Own Job
Requirement	FR-012
Preconditions	Contractor owns an open job and signs in as the bidder.
Steps	1. POST `/api/bids/:jobId` with the contractor's token and a valid amount.
Expected Result	HTTP 403 with `{error: "own_job_bid"}`; no bid written.
Actual Result	Passed – "prevents contractors from bidding on their own jobs" test in `bids.integration.real.routes.test.js`.
Status	Passed
Priority	High

TC-022b

Test Case ID	TC-022b
Title	Bid Accept - Lock Job/Bid Lifecycle
Requirement	FR-012
Preconditions	Contractor owns a job; bidder submits a valid bid; Firebase tokens exist for contractor, bidder, and observer.
Steps	1. Bidder POSTs `/api/bids/:jobId` with a valid amount. 2. Contractor POSTs `/api/bids/:jobId/:bidId/accept`. 3. Contractor tries to PATCH the job; bidder tries to PATCH the accepted bid. 4. Observer GETs `/api/jobs` to check visibility.
Expected Result	Job flips to `awarded` with `awardedBidId`; further edits return `job_locked` / `bidding_closed`; observers no longer see the job.
Actual Result	Passed – "Accepting a bid hides the job from outsiders and locks further edits" test.
Status	Passed
Priority	High

6 Code Repository and Branching Strategy

Repository: https://github.com/tjung-git/OpenBid-Map-first-Bidding-Markplace *(placeholder)*

Branches:

- `main`: stable releases.
- `develop`: ongoing integration.
- `feature/{slug}`: e.g., `feature/auth`, `feature/map`, `feature/bids`, `feature/kyc`, `feature/tests`.

7 Task Allocation and Timeline

Pair-Programming Rotation (weekly): two pairs; driver/navigator swap daily; Scrum Master rotates weekly (Tyler → Mani → Yanness → Alaister).

Sprint Plan (3 weeks for Iteration 3)

- **Week 1:** Implement auth + Duo 2FA hook, KYC gate stub, Post Job UI/API, initial tests.
- **Week 2:** Map browse (pins, filters), Place Bid flow, tests.
- **Week 3:** Prototype hardening, accessibility pass (keyboard focus), performance harness scaffolding, test documentation.

Task Breakdown (examples)

- Frontend: NewJob form, MapView, JobList, Bid modal, validation.
- Backend (Express on Firebase Functions/Cloud Run): endpoints for jobs, bids; KYC/Duo middleware.

- Firestore rules: enforce `kycVerified == true` for job/bid writes.
- Testing: Vitest/Jest + React Testing Library for UI; supertest for API; rules-unit-testing for Firestore.
- Docs: update traceability, add test run notes.

8 Prototype Overview

Stack (MVP): React + SCSS (Firebase Hosting), Node.js + Express (Firebase Functions or Cloud Run), Firestore, Google Maps JS, Stripe (Connect/Identity), Duo 2FA.

Implemented in Iteration 3 (target)

- Auth shell with Duo challenge on sensitive action.
- KYC gate (UI + rules) using a sandbox token flow.
- Post Job UI + server write; basic Job Detail.
- Map browse with radius/category filters on seeded data.
- Place Bid basic happy path.

9 Submission Guidelines

- **GitHub:** Push code and documentation; tag release as `ITR2.1`.
- **eClass PDF:** include *Prototype Overview*, *Updated Requirements/Use Cases*, *Test Plan and (initial) Results*, and *Updated Iteration Plan/Backlog*.

Appendix A: Test Skeletons (illustrative)

Frontend (React Testing Library)

```
import { render, screen, fireEvent } from '@testing-library/react'
import NewJob from '../src/pages/NewJob'

test('TC-002: shows validation error when title empty', async () => {
  render(<NewJob />)
  fireEvent.click(screen.getByRole('button', { name: /post/i }))
  expect(await screen.findByText(/title is required/i)).toBeInTheDocument()
})
```

API (Express + supertest)

```
import request from 'supertest'
import app from '../functions/app'

test('TC-007: blocks job create when kyc not verified', async () => {
  const token = await getAuthToken({ kycVerified: false })
  const res = await request(app)
```

```
    .post('/jobs')
    .set('Authorization', `Bearer ${token}`)
    .send({ title: 'Yard help', ... })
  expect(res.status).toBe(403)
})
```

Firestore Rules (rules-unit-testing)

```
it('TC-012: rejects write without kycVerified', async () => {
  const db = authedDB({ uid: 'u1', kycVerified: false })
  const ref = db.collection('jobs').doc('j1')
  await assertFails(ref.set({ title: 'T', ... }))
})
```

KYC Routes (Express + supertest + Jest)

```
import request from 'supertest'
import express from 'express'
// Create Stripe verification session
test('should return Stripe verification URL', async () => {
  const response = await request(app)
    .post('/api/kyc/verification')
    .set('Authorization', 'Bearer fake-token')
    .expect(200)
  expect(response.body.url).toContain('https://verify.stripe.com/')
  expect(response.body.sessionId).toContain('vs')
})

// Check KYC status
test('should return KYC status', async () => {
  const response = await request(app)
    .get('/api/kyc/status')
    .set('Authorization', 'Bearer fake-token')
    .expect(200)
  expect(['verified', 'pending', 'failed'])
    .toContain(response.body.status)
})

// Unauthorized access
test('should reject unauthorized user', async () => {
  mockAuth.auth.verify.mockResolvedValue(null)
  const response = await request(app)
    .post('/api/kyc/verification')
    .expect(401)
  expect(response.body.error).toBe('unauthorized')
})
```