

DIGT2107: Practice of Software Development  
Project Iteration 4: Full Documentation, Prototype, Presentation  
OpenBid

Team 1: Tyler      Mani      Yanness      Alaister

Due: November 30, 2025

**Course:** DIGHT2107 – 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 initial prototype, testing the prototype, and updating the documentation. The main highlight of this iteration is delivering a prototype of the OpenBid application.

## 2 Iteration Goals and Objectives

- **Post a job** and **place a bid** core bidding functionality should be fully implemented in the prototype.
- Core authentication and authorization functionality should be updated in the prototype- specifically authentication and security flows: **auth + Duo 2FA** and **KYC gate** (stubbed via Stripe Identity sandbox token).
- Develop and run additional unit tests mapped to requirements (frontend component tests and backend handler tests) \*as required. Conduct manual e2e testing.
- Update documentation- reiterate user stories and user cases, update functional and non-functional requirements, and provide a user manual.
- Deliver a prototype demonstrating requirement ↔ test case links.

## 3 Detailed User Stories

### Contractors

**US-1: Browse Nearby Jobs (High, 3 pts)** As a **Contractor**, I want to see jobs near me with filters so I can quickly find relevant work.

- **Acceptance:** Map/list render; radius/category/budget/date filters; open job details view.

**US-2: Place a Bid (High, 5 pts)** As a **Contractor**, I want to place a bid with an amount, note, and ETA so the Job Poster can evaluate me.

- **Acceptance:** Bid saved; Job Poster notified; contractor can edit/withdraw prior to award.

**US-3: Complete KYC (High, 3 pts)** As a **Contractor**, I want to verify my identity (document + selfie) so I can post jobs safely.

- **Acceptance:** Stripe Identity flow completes; app shows verified badge; bidding features unlock.

**US-4: Duo 2FA (Medium, 2 pts)** As a **Contractor**, I want Duo phone confirmation so my account is protected during sign-in.

- **Acceptance:** 2FA challenge required; recovery path documented.

**US-5: Job Thread Messaging (Medium, 3 pts)** As a **Contractor**, I want a per-job chat so I can coordinate details with job posters.

- **Acceptance:** Send/receive text; attachments; report/block; notifications.

**US-6: Leave Reviews (Medium, 2 pts)** As a **Contractor**, I want to rate and review job posters after completion so future users can trust matches.

- **Acceptance:** 1–5 stars plus comment; appears on bidder profile; single review per job.

**US-7: Create an account (High, 5 pts)** As a **Contractor**, I want to create an account using my email.

- **Acceptance:** Login page present with a seamless sign up process.

**US-8: Personalize Profile (High, 5 pts)** As a **Contractor**, I want to personalize my profile to showcase my skills, reviews, experience etc.

- **Acceptance:** Editable contractor profile page to change profile picture, skills, experience etc.

## Job Posters

**US-9: Safety Score Guidance (Medium, 3 pts)** As a **Contractor**, I want safety guidance at post-time so I can choose safer options (e.g., daylight/public meetup).

- **Acceptance:** Score shown with tips; low-score flows add friction or require verification.

**US-10: Accept a Bid & Fund Escrow (High, 4 pts)** As a **Job Poster**, I want to accept a bid and put funds on hold so payment is guaranteed on completion.

- **Acceptance:** Stripe hold succeeds; job moves to **awarded**; chat thread opens with winner.

**US-11: Complete KYC (High, 3 pts)** As a **Job Poster**, I want to verify my identity (document + selfie) so I can post jobs safely.

- **Acceptance:** Stripe Identity flow completes; app shows verified badge; posting features unlock.

**US-12: Duo 2FA (Medium, 2 pts)** As a **Job Poster**, I want Duo phone confirmation so my account is protected during sign-in and payouts.

- **Acceptance:** 2FA challenge required; recovery path documented.

**US-13: Job Thread Messaging (Medium, 3 pts)** As a **Job Poster**, I want a per-job chat so I can coordinate details with Contractors.

- **Acceptance:** Send/receive text; attachments; report/block; notifications.

**US-14: Leave Reviews (Medium, 2 pts)** As a **Job Poster**, I want to rate and review Contractors after completion so future users can trust contractors.

- **Acceptance:** 1–5 stars plus comment; appears on Contractor profile; single review per job.

**US-15: Create an account (High, 5 pts)** As a **Job Poster**, I want to create an account using my email.

- **Acceptance:** Login page present with a seamless sign up process.

### **Job Browsers (Not KYC-Verified Users)**

**US-16: Browse Jobs Read-Only (Medium, 2 pts)** As a **browser**, I want to explore posted jobs on a map in read-only mode, so I can understand the platform's services before committing.

- **Acceptance:** Map/list view available; limited job details visible; registration prompt when trying to post/bid.

**US-17: View Limited Job Details (Medium, 1 pt)** As a **browser**, I want to view limited job details (e.g., title, category, approximate location, budget range) without registering, so I see the platform's value.

- **Acceptance:** Basic job info visible; contact details and full descriptions hidden; registration required for full access.

**US-18: Registration Prompt (Medium, 1 pt)** As a **browser**, I want to be prompted to create an account if I try to post a job or place a bid, so I know registration is required.

- **Acceptance:** Clear registration modal appears; KYC/2FA requirements explained; seamless signup flow.

**US-19: KYC/2FA Instructions (Medium, 1 pt)** As a **newly registered user**, I want clear instructions that I must complete KYC verification and enable Duo 2FA before being allowed to post jobs or bid, so I understand the trust policy.

- **Acceptance:** Step-by-step guidance; progress indicators; help resources available.

**US-20: Access Support Resources (Low, 1 pt)** As a **browser**, I want access to support or FAQ pages, so I can ask questions or learn more before verifying my account.

- **Acceptance:** Support contact form; FAQ section; help documentation accessible without full registration.

### **Admin**

**US-21: Review Reports (High, 3 pts)** As an **Admin**, I want to review user reports and disputes so I can maintain platform safety and integrity.

- **Acceptance:** Report dashboard; case management tools; resolution tracking.

**US-22: Manage Categories (Medium, 2 pts)** As an **Admin**, I want to manage job categories so the platform stays organized and relevant.

- **Acceptance:** Category CRUD operations; validation; audit logging.

**US-23: Handle Disputes (High, 4 pts)** As an **Admin**, I want to handle payment and job disputes so conflicts are resolved fairly.

- **Acceptance:** Dispute intake; evidence review; resolution workflow; communication tools.

**US-24: System Monitoring (Medium, 2 pts)** As an **Admin**, I want to monitor system health and performance so I can ensure platform reliability.

- **Acceptance:** Dashboard with metrics; alert system; performance monitoring.

## 4 Functional and Non-Functional Requirements

### 4.1 Functional Requirements

1. **Account & Identity (→ US-3, US-7, US-11, US-15, US-18, US-19):** Email/password sign-up and sign-in (handled via Firebase Auth in production). Mandatory KYC before Contractors can bid or Posters can create jobs (handled via Stripe Identity in production).
2. **Viewing Jobs (→ US-16, US-17):** Job browsers can view jobs after signing up an account.
3. **Contractor Profiles (→ US-8):** Contractors can create and update basic profiles. In production, contractors can enhance their profiles by including skills, description, photos, rates, availability, and their location. Support for deletion.
4. **Discovery (→ US-1):** Contractors can browse for jobs using a list. In production, there will be an added map view and job filters (radius, category/skill, budget, availability date).
5. **Bidding (→ US-2):** KYC verified Contractors can place, edit, and withdraw bids (amount, note, ETA).
6. **Posting Jobs (→ US-10):** KYC verified Job Posters can place, edit, and delete jobs (amount, location, description, etc.). A Job Poster can also accept a bid on a job.
7. **Two-Factor Authentication (2FA) (→ US-4, US-12):** Duo 2FA required for sign-in.
8. **Payments (Escrow) (→ US-10):** Job Posters fund escrow on offer acceptance; payouts via Stripe Connect on completion.
9. **Messaging: (→ US-5, US-13):** Secure in-offer messaging with attachments and report/block options.
10. **Reviews (→ US-6, US-14):** Both parties leave ratings and written reviews.
11. **Safety Score (→ US-9):** System calculates a 0–100 location safety score and applies friction controls.
12. **Admin Dashboard: (→ US-21, US-22, US-23, US-24)** Admins manage reports, categories, and disputes.

### 4.2 Non-Functional Requirements

Most non-functional requirements will only be implemented in the production application).

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

## 5 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 <b>post jobs</b> with title, description, budget, photos, and location.	TC-001, TC-002	Partially Covered
FR-002	System shall allow providers to <b>browse jobs on a map</b> with basic filters (radius, category).	TC-003a-d, TC-004	Partially Covered
FR-003	System shall allow providers to <b>place bids</b> on open jobs; posters can view and accept a bid.	TC-005, TC-006	Partially Covered
FR-004	System shall enforce <b>KYC for all users</b> before posting or bidding.	TC-007, TC-008a-b, TC-009a-b, TC-015a-e	Covered
FR-005	System shall support <b>user authentication</b> with <b>Duo 2FA</b> for sensitive actions.	TC-008	Covered
FR-006	System shall provide <b>in-thread messaging</b> per job after acceptance.	TC-010	Planned
FR-007	System shall record <b>ratings and reviews</b> 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 job poster 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

Req ID	Requirement Description	Test Case IDs	Coverage
FR-011	Only KYC-approved job posters 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 job poster accepts an offer.	TC-018a-b	Covered
FR-013	System shall allow users to <b>manage their profile</b> including avatar upload and personal information.	TC-016a-d	Covered

### Non-Functional Requirements (NFR)

Req ID	Requirement Description	Test Case IDs	Coverage
NFR-001	<b>Performance:</b> Search/browse should return results within 800 ms P95 for a metro with 5k open jobs (stub data).	TC-011	Planned
NFR-002	<b>Security:</b> Only KYC-verified users can hit write endpoints for jobs/bids.	TC-007, TC-008b, TC-009b, TC-013, TC-016b	Partially Covered
NFR-003	<b>Usability:</b> Map view and list view maintain accessible contrast and keyboard navigation for core actions. Application shall be mobile responsive.	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.
- At this time, it was determined that it would be difficult to implement tests for the scalability, reliability and privacy non-FRs. Test cases will be planned and implemented in a future iteration.

## 6 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

<b>Test Case ID</b>	TC-001
<b>Title</b>	Post Job - Valid Inputs
<b>Requirement</b>	FR-001
<b>Preconditions</b>	User is logged in, KYC status = verified.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. Navigate to <code>/new-job</code>.</li><li>2. Enter valid title, description, budget, and pick a map location (Google Maps widget).</li><li>3. Upload a photo (sample file).</li><li>4. Click <b>Post</b>.</li></ol>
<b>Expected Result</b>	Job document is created in Firestore; UI redirects to Job Detail.
<b>Actual Result</b>	To be filled after execution.
<b>Status</b>	Planned
<b>Priority</b>	High

### TC-002

<b>Test Case ID</b>	TC-002
<b>Title</b>	Post Job - Validation Errors
<b>Requirement</b>	FR-001
<b>Preconditions</b>	Logged in, KYC verified.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. Navigate to <code>/new-job</code>.</li><li>2. Leave title empty; click <b>Post</b>.</li></ol>
<b>Expected Result</b>	Client-side validation shows error; no write occurs.
<b>Actual Result</b>	To be filled after execution.
<b>Status</b>	Planned
<b>Priority</b>	High

### TC-003a

<b>Test Case ID</b>	TC-003a
<b>Title</b>	The haversineFormulakm function accurately calculates distance between 2 coordinates.
<b>Requirement</b>	FR-002
<b>Preconditions</b>	Valid coordinates are input.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. Coordinates are input into the function.</li><li>2. Function calculates the distance between the coordinates.</li><li>3. Function returns the distance between the coordinates.</li></ol>
<b>Expected Result</b>	Returns distance in km of the 2 coordinates, returns infinity when there is invalid input.
<b>Actual Result</b>	Passed - Returns distance accurately within a specific tolerance. Coordinates tested are up to 250km apart.
<b>Status</b>	Passed
<b>Priority</b>	High



### TC-003b

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

### TC-003c

<b>Test Case ID</b>	TC-003c
<b>Title</b>	The degToRad function accurately converts a number in degrees to radians.
<b>Requirement</b>	FR-002
<b>Preconditions</b>	Input should be a number.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. a number in degrees is input into the function.</li><li>2. Function checks input for validity then returns the input value in radians.</li></ol>
<b>Expected Result</b>	Returns the input number converted into radians if the input is valid, returns infinity otherwise.
<b>Actual Result</b>	Passed - accurately converts a valid input to radians and returns infinity otherwise.
<b>Status</b>	Passed
<b>Priority</b>	High

### TC-003d

<b>Test Case ID</b>	TC-003d
<b>Title</b>	The createMap function loads a google map. (uses api mocking)
<b>Requirement</b>	FR-002
<b>Preconditions</b>	The lat, lng, ref, and apiKey inputs are valid. The lat and lng are tested for validity before the function call and the logic outside the function affirms that apiKey and ref are valid.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. The google maps api loader is initialized.</li><li>2. The markers array input is filtered for valid markers or is set to an empty array if the markers input is bad.</li><li>3. The google maps api loader is used to load the api.</li><li>4. A new google map is created and valid markers are attached to the map.</li></ol>
<b>Expected Result</b>	Creates a map with valid markers attached to it.
<b>Actual Result</b>	Passed - creates a map with valid markers attached to it, invalid markers are discarded and a bad markers input is gracefully handled.
<b>Status</b>	Passed
<b>Priority</b>	High

### TC-004

<b>Test Case ID</b>	TC-004
<b>Title</b>	Map Browse - Category Filter
<b>Requirement</b>	FR-002
<b>Preconditions</b>	User is on map browse page with multiple job categories available.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. Select a specific job category from the filter dropdown.</li><li>2. Observe the jobs displayed on the map.</li></ol>
<b>Expected Result</b>	Only jobs of selected category are shown.
<b>Actual Result</b>	To be filled after execution.
<b>Status</b>	Planned
<b>Priority</b>	Medium

### TC-005

<b>Test Case ID</b>	TC-005
<b>Title</b>	Place Bid - Valid
<b>Requirement</b>	FR-003
<b>Preconditions</b>	Provider is logged in, KYC verified, job is open.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. Navigate to job detail page.</li><li>2. Enter valid bid amount and optional message.</li><li>3. Click "Submit Bid" button.</li></ol>
<b>Expected Result</b>	Bid document created; poster sees bid in Job Detail.
<b>Actual Result</b>	To be filled after execution.
<b>Status</b>	Planned
<b>Priority</b>	High

## TC-006

<b>Test Case ID</b>	TC-006
<b>Title</b>	Accept Bid - Poster Flow
<b>Requirement</b>	FR-003
<b>Preconditions</b>	Contractor is logged in, has posted a job, and has received at least one bid.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. Navigate to job detail page with bids.</li><li>2. Select a bid to accept.</li><li>3. Click "Accept Bid" button.</li><li>4. Confirm acceptance in modal dialog.</li></ol>
<b>Expected Result</b>	Job status transitions to <b>awarded</b> ; winning bid marked <b>accepted</b> .
<b>Actual Result</b>	To be filled after execution.
<b>Status</b>	Planned
<b>Priority</b>	High

## TC-007

<b>Test Case ID</b>	TC-007
<b>Title</b>	KYC Gate - Block Unverified Writes
<b>Requirement</b>	FR-004, NFR-002
<b>Preconditions</b>	User KYC status = pending.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. Attempt to post a new job through UI.</li><li>2. Attempt to place a bid on an existing job.</li><li>3. Attempt direct API calls to create job/bid endpoints.</li></ol>
<b>Expected Result</b>	Attempts to post job or bid are rejected by Firestore rules/API; UI shows KYC required.
<b>Actual Result</b>	To be filled after execution.
<b>Status</b>	Planned
<b>Priority</b>	High

## TC-008a

<b>Test Case ID</b>	TC-008a
<b>Title</b>	KYC Verification - Create Stripe Verification Session
<b>Requirement</b>	FR-004
<b>Preconditions</b>	User is authenticated; KYC status = <b>pending</b> ; real KYC mode enabled.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. Send POST request to <code>/api/kyc/verification</code> with valid auth JWT token.</li><li>2. System calls Stripe Identity API to create verification session.</li><li>3. System stores session ID in user record.</li><li>4. System returns Stripe verification URL and session ID.</li></ol>
<b>Expected Result</b>	Returns Stripe verification URL ( <code>https://verify.stripe.com/*</code> ) and session ID ( <code>vs_*</code> )
<b>Actual Result</b>	Passed - Returns mocked Stripe URL and session ID.
<b>Status</b>	<b>Passed</b>
<b>Priority</b>	High

### TC-008b

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

### TC-009a

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

### TC-009b

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

## TC-010

<b>Test Case ID</b>	TC-010
<b>Title</b>	Messaging After Acceptance
<b>Requirement</b>	FR-006
<b>Preconditions</b>	Job has been awarded to a bidder; both parties are authenticated.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. Navigate to awarded job detail page.</li><li>2. Enter message in the messaging thread.</li><li>3. Submit message.</li><li>4. Other party logs in and views the job detail page.</li></ol>
<b>Expected Result</b>	Parties can exchange messages in job thread; messages persist in Firestore.
<b>Actual Result</b>	To be filled after execution.
<b>Status</b>	Planned
<b>Priority</b>	Medium

## TC-011

<b>Test Case ID</b>	TC-011
<b>Title</b>	Submit Review on Completion
<b>Requirement</b>	FR-007
<b>Preconditions</b>	Job has been marked as completed; user is authenticated as job poster or winning bidder.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. Navigate to completed job detail page.</li><li>2. Click "Leave Review" button.</li><li>3. Enter rating (1-5 stars) and review text.</li><li>4. Submit review.</li><li>5. Attempt to submit a second review for the same job.</li></ol>
<b>Expected Result</b>	Review saved and visible on profile; duplicate review blocked.
<b>Actual Result</b>	To be filled after execution.
<b>Status</b>	Planned
<b>Priority</b>	Medium

## TC-012

<b>Test Case ID</b>	TC-012
<b>Title</b>	Performance P95 - Map Query
<b>Requirement</b>	NFR-001
<b>Preconditions</b>	Test environment with 5,000 stub job records in database; performance monitoring enabled.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. Load map view with default filters.</li><li>2. Change category filter to a specific job type.</li><li>3. Measure time from filter change to results rendering.</li><li>4. Repeat test 100 times to calculate P95 metric.</li></ol>
<b>Expected Result</b>	P95 end-to-end from filter change to results render $\leq 800$ ms on stub dataset.
<b>Actual Result</b>	To be filled after execution.
<b>Status</b>	Planned
<b>Priority</b>	Low

### TC-013

<b>Test Case ID</b>	TC-013
<b>Title</b>	Ruleset Audit - No Write Without Claims
<b>Requirement</b>	NFR-002
<b>Preconditions</b>	Firebase project with security rules deployed; test environment configured.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. Authenticate as user without KYC verification.</li><li>2. Attempt to write to jobs collection.</li><li>3. Attempt to write to bids collection.</li><li>4. Verify rule rejection behavior.</li></ol>
<b>Expected Result</b>	Firestore rules reject writes missing <code>kycVerified == true</code> .
<b>Actual Result</b>	To be filled after execution.
<b>Status</b>	Planned
<b>Priority</b>	High

### TC-014

<b>Test Case ID</b>	TC-014
<b>Title</b>	Accessibility - Keyboard Nav on Map/List
<b>Requirement</b>	NFR-003
<b>Preconditions</b>	Application running with jobs loaded in map and list views.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. Navigate to map/list view page.</li><li>2. Use Tab key to navigate through UI elements.</li><li>3. Verify focus indicators are visible on all interactive elements.</li><li>4. Test keyboard operation of filters, job selection, and primary actions.</li></ol>
<b>Expected Result</b>	Tabbing reaches filters, list items, and primary actions; visible focus outlines present.
<b>Actual Result</b>	To be filled after execution.
<b>Status</b>	Planned
<b>Priority</b>	Low

### TC-015a

<b>Test Case ID</b>	TC-015a
<b>Title</b>	Profile Page - Display KYC Pending Status
<b>Requirement</b>	FR-004
<b>Preconditions</b>	User authenticated, KYC status = <code>pending</code> .
<b>Steps</b>	<ol style="list-style-type: none"><li>1. Navigate to <code>/profile</code>.</li><li>2. Profile component renders and displays account status section.</li><li>3. System retrieves user KYC status from session.</li></ol>
<b>Expected Result</b>	KYC status displays "Pending" tag with appropriate styling.
<b>Actual Result</b>	Passed - Profile page correctly displays pending status.
<b>Status</b>	Passed
<b>Priority</b>	High

### TC-015b

<b>Test Case ID</b>	TC-015b
<b>Title</b>	Profile Page - Display KYC Verified Status
<b>Requirement</b>	FR-004
<b>Preconditions</b>	User authenticated, KYC status = <b>verified</b> .
<b>Steps</b>	<ol style="list-style-type: none"><li>1. Navigate to <b>/profile</b>.</li><li>2. Profile component renders with verified KYC status.</li></ol>
<b>Expected Result</b>	KYC status displays "Verified" tag; no action buttons shown.
<b>Actual Result</b>	Passed - Profile page displays verified status correctly.
<b>Status</b>	<b>Passed</b>
<b>Priority</b>	High

### TC-015c

<b>Test Case ID</b>	TC-015c
<b>Title</b>	Profile Page - Complete KYC Button Displayed
<b>Requirement</b>	FR-004
<b>Preconditions</b>	User authenticated, KYC status = <b>pending</b> .
<b>Steps</b>	<ol style="list-style-type: none"><li>1. Navigate to <b>/profile</b>.</li><li>2. System checks KYC status is not verified.</li><li>3. Profile component renders action buttons.</li></ol>
<b>Expected Result</b>	"Complete KYC" and "Refresh Status" buttons are visible.
<b>Actual Result</b>	Passed - Action buttons displayed for pending status.
<b>Status</b>	<b>Passed</b>
<b>Priority</b>	High

### TC-015d

<b>Test Case ID</b>	TC-015d
<b>Title</b>	Profile Page - Initiate KYC Verification
<b>Requirement</b>	FR-004
<b>Preconditions</b>	User authenticated, KYC pending, production mode.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. User clicks "Complete KYC" button.</li><li>2. System calls <b>/api/kyc/verification</b> endpoint.</li><li>3. System receives Stripe verification URL.</li><li>4. System opens URL in new browser tab.</li></ol>
<b>Expected Result</b>	Stripe Identity verification opens in new tab; notification displayed.
<b>Actual Result</b>	Passed - Verification URL opened correctly.
<b>Status</b>	<b>Passed</b>
<b>Priority</b>	High

### TC-015e

<b>Test Case ID</b>	TC-015e
<b>Title</b>	Profile Page - Refresh KYC Status
<b>Requirement</b>	FR-004
<b>Preconditions</b>	User authenticated, KYC pending.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. User clicks "Refresh Status" button.</li><li>2. System calls <code>/api/kyc/status</code> endpoint.</li><li>3. System receives updated status.</li><li>4. System updates UI and session with new status.</li></ol>
<b>Expected Result</b>	KYC status refreshed; UI updated with current status.
<b>Actual Result</b>	Passed - Status refresh works correctly.
<b>Status</b>	Passed
<b>Priority</b>	High

### TC-016a

<b>Test Case ID</b>	TC-016a
<b>Title</b>	Profile Page - Upload Avatar Successfully
<b>Requirement</b>	FR-008 (User Profile Management)
<b>Preconditions</b>	User authenticated.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. User clicks "Change Avatar" button.</li><li>2. User selects valid image file (PNG, &lt; 5MB).</li><li>3. System uploads file to storage.</li><li>4. System updates user profile with avatar URL.</li></ol>
<b>Expected Result</b>	Avatar uploaded; preview displayed; success notification shown.
<b>Actual Result</b>	Passed - Avatar upload successful.
<b>Status</b>	Passed
<b>Priority</b>	High

### TC-016b

<b>Test Case ID</b>	TC-016b
<b>Title</b>	Profile Page - Reject Large Avatar Files
<b>Requirement</b>	FR-008, NFR-002 (Security)
<b>Preconditions</b>	User authenticated.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. User clicks "Change Avatar" button.</li><li>2. User selects image file larger than 5MB.</li><li>3. System validates file size before upload.</li></ol>
<b>Expected Result</b>	Upload rejected; error message "Image must be smaller than 5MB" displayed.
<b>Actual Result</b>	Passed - Large files rejected correctly.
<b>Status</b>	Passed
<b>Priority</b>	High



### TC-016c

<b>Test Case ID</b>	TC-016c
<b>Title</b>	Profile Page - Handle Avatar Upload Error
<b>Requirement</b>	FR-008
<b>Preconditions</b>	User authenticated; network/server error occurs.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. User selects valid image file.</li><li>2. System attempts upload to <code>/api/avatar/upload</code>.</li><li>3. Server returns error response.</li></ol>
<b>Expected Result</b>	User-friendly error message displayed; no profile update.
<b>Actual Result</b>	Passed - Error handled gracefully.
<b>Status</b>	Passed
<b>Priority</b>	High

### TC-016d

<b>Test Case ID</b>	TC-016d
<b>Title</b>	Profile Page - Display Existing Avatar
<b>Requirement</b>	FR-008
<b>Preconditions</b>	User authenticated; avatar URL exists in profile.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. Navigate to <code>/profile</code>.</li><li>2. System retrieves user avatar URL from database.</li><li>3. Profile component renders avatar image.</li></ol>
<b>Expected Result</b>	User's avatar displayed in profile header.
<b>Actual Result</b>	Passed - Existing avatar displayed correctly.
<b>Status</b>	Passed
<b>Priority</b>	High

### TC-017

<b>Test Case ID</b>	TC-017a
<b>Title</b>	Duo 2FA Authentication Flow
<b>Requirement</b>	FR-005
<b>Preconditions</b>	User exists, Duo enabled.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. POST <code>/api/auth/login</code> with valid credentials.</li><li>2. Follow <code>mfa.startUrl</code> to begin Duo: GET <code>/api/auth/duo/start?state=...</code></li><li>3. Simulate Duo callback: GET <code>/api/auth/duo/callback?state=...&amp;code=allow-123</code>.</li><li>4. Finalize: POST <code>/api/auth/duo/finalize</code> with one-time code.</li></ol>
<b>Expected Result</b>	Login returns 202 with Duo start URL; callback redirects to <code>/login/finish?code=...</code> ; finalize returns session JSON.
<b>Actual Result</b>	Passed – Covered by Jest tests/auth.duo.test.js.
<b>Status</b>	Passed
<b>Priority</b>	High

### TC-018a

<b>Test Case ID</b>	TC-018a
<b>Title</b>	Firebase Signup - Provision Account and Send Verification Email
<b>Requirement</b>	FR-008
<b>Preconditions</b>	Email is brand new inside the Firebase project; Identity Toolkit is online.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. Send POST <code>/api/auth/signup</code> with first/last name, valid email, password, confirmPassword.</li><li>2. Let Firebase create the account and queue the verification email.</li><li>3. Read the JSON response from our API.</li></ol>
<b>Expected Result</b>	HTTP 201 with sanitized user data, email + KYC both pending, and Firebase verification email requested.
<b>Actual Result</b>	Passed – Confirmed in <code>server/src/routes/__tests__/auth.integration.real.routes.test.js</code> .
<b>Status</b>	Passed

### TC-018b

<b>Test Case ID</b>	TC-018b
<b>Title</b>	Firebase Signup - Reject Invalid Credentials
<b>Requirement</b>	FR-008
<b>Preconditions</b>	Same Firebase project as TC-014a; email still unused.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. POST <code>/api/auth/signup</code> with a broken email or a password shorter than eight characters.</li><li>2. Watch the server reject the payload before any Firebase call.</li></ol>
<b>Expected Result</b>	HTTP 400 with a clear error; Firebase signup helper never runs.
<b>Actual Result</b>	Passed – Shown in the same Jest suite for real adapters.
<b>Status</b>	Passed
<b>Priority</b>	High

### TC-018c

<b>Test Case ID</b>	TC-018c
<b>Title</b>	Firebase Login - Verified User Receives Session Tokens
<b>Requirement</b>	FR-008
<b>Preconditions</b>	User already exists in Firebase with both email and KYC marked verified.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. POST <code>/api/auth/login</code> with the correct email and password.</li><li>2. Inspect the response for session + requirement flags.</li></ol>
<b>Expected Result</b>	HTTP 200 with sanitized user, Firebase ID + refresh tokens, and <code>requirements = {emailVerified: true, kycVerified: true}</code> .
<b>Actual Result</b>	Passed – Assertions live in <code>auth.integration.real.routes.test.js</code> .
<b>Status</b>	Passed
<b>Priority</b>	High

### TC-018d

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

### TC-019a

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

### TC-019b

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

### TC-020a

<b>Test Case ID</b>	TC-020a
<b>Title</b>	Session Service - Persist and Notify Subscribers
<b>Requirement</b>	FR-010
<b>Preconditions</b>	Browser environment available (Vitest JSDOM); subscribers registered.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. Call <code>setSession</code> with a real Firebase-authenticated user payload.</li><li>2. Verify <code>subscribeSession</code> listener triggered.</li><li>3. Inspect <code>localStorage</code> for serialized session data.</li></ol>
<b>Expected Result</b>	Session stored under <code>openbid_session</code> ; only real-user metadata persisted; listeners invoked once.
<b>Actual Result</b>	Passed – Covered by <code>client/src/__tests__/session.test.js</code> .
<b>Status</b>	Passed
<b>Priority</b>	Medium

### TC-020b

<b>Test Case ID</b>	TC-020b
<b>Title</b>	Session Service - setUser Preserves Requirements
<b>Requirement</b>	FR-010
<b>Preconditions</b>	Existing session with requirement flags stored.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. Initialize session via <code>setSession</code>.</li><li>2. Invoke <code>setUser</code> with new user payload and explicit requirements.</li><li>3. Query <code>getRequirements</code> and <code>getSession</code>.</li></ol>
<b>Expected Result</b>	Updated user persisted; requirements reflect provided override (email + KYC flags).
<b>Actual Result</b>	Passed – Validated in <code>session.test.js</code> .
<b>Status</b>	Passed
<b>Priority</b>	Medium

### TC-020c

<b>Test Case ID</b>	TC-020c
<b>Title</b>	Session Service - Inactivity Monitor Auto Logout
<b>Requirement</b>	FR-010
<b>Preconditions</b>	Session established; Vitest fake timers.
<b>Steps</b>	<ol style="list-style-type: none"><li>1. Call <code>startInactivityMonitor</code> with spy callback.</li><li>2. Advance timers to just before the 2-minute limit; ensure no callback.</li><li>3. Advance timers past the limit; ensure callback fires and cleanup works.</li></ol>
<b>Expected Result</b>	Timeout callback executes exactly once after 2 minutes of inactivity, indicating logout.
<b>Actual Result</b>	Passed – Covered by Vitest case in <code>session.test.js</code> .
<b>Status</b>	Passed
<b>Priority</b>	Medium

### TC-021a

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

### TC-021b

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

### TC-021c

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

### TC-021d

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

### TC-021e

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

### TC-021f

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

### TC-021g

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

### TC-021h

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

### TC-021i

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

## TC-022a

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

## TC-022b

<b>Test Case ID</b>	TC-022b
<b>Title</b>	Bid Accept - Lock Job/Bid Lifecycle
<b>Requirement</b>	FR-012
<b>Preconditions</b>	Contractor owns a job; bidder submits a valid bid; Firebase tokens exist for contractor, bidder, and observer.
<b>Steps</b>	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.
<b>Expected Result</b>	Job flips to awarded with awardedBidId; further edits return job_locked / bidding_closed; observers no longer see the job.
<b>Actual Result</b>	Passed – “Accepting a bid hides the job from outsiders and locks further edits” test.
<b>Status</b>	Passed
<b>Priority</b>	High

## 7 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.

## 8 Task Allocation and Timeline

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



## Sprint Plan (4 weeks for Iteration 4)

- **Week 1 - 2:** Test and update the prototype based on feedback and user experience.
- **Week 3 - 4:** Create user manual and update 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.

## 9 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. LINE TO BE UPDATED

### Implemented in Iteration 4 (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.

## 10 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')
})
```