Project Milestone #2 - 4+1 View Model + Database

Course: SFWRTECH 4SA3 Software Architecture

Professor: Kevin Browne

Written By: Tyler Thong Nguyen - nguyeb65 - 400610270

Version: 1.0

Last Edited: Oct 26, 2025

Table of Contents

1. Introduction **3**
- 1.1 Application Overview 3
- 1.2 Technologies 3
- 1.3 Documentation Overview 5
- 1.4 Application Usage 5

2. Requirements **6**

3. Scenario View **8**

4. Physical View **9**

5. Development View **11**

6. Process View **13**

7. Logical View **17**

8. Database Schema **21**

1. Introduction

This document is to provide an overview of the software architecture for the JobTrust application.

1.1 Application Overview

JobTrust is an AI-powered platform offered both as a browser widget and a fully-featured dashboard web application. It helps job seekers detect fake job postings, match jobs to their unique profiles and resumes, and get actionable resume or skill improvement advice. All features are delivered through modular, secure interfaces that stay synchronized between the widget and dashboard.

Users can analyze job postings for fraud using NLP-based detection, receive personalized job matching scores based on their resume and profile, and get AI-powered resume improvement recommendations. The platform implements a credit-based system where users receive 50 free credits upon registration, with credits consumed per feature usage. Additional credits can be purchased securely through Stripe payment integration.

Target Audience:

- Students, new graduates, and active job seekers.
- Professionals looking to switch careers or upskill.
- Anyone searching for jobs online and concerned about scams and fake postings.

1.2 Technologies

The following technologies are used to implement the JobTrust application:

Technology	Role	Sources
React.js	Frontend framework for dashboard and browser widget UI	https://reactjs.org/
Manifest V3	Browser extension architecture for widget packaging	https://developer.chrome.com/docs/extensions/mv3/

Python (FastAPI)	Backend framework for API development, business logic, and algorithm implementation	https://fastapi.tiangolo.com/
PostgreSQL	Relational database for structured data including users, transactions, and job analyses	https://www.postgresql.org/
Neon for PostgreSQL	The database developers trust, on a serverless platform designed to help you build reliable and scalable applications faster.	https://neon.com/
Ruvia Trust API	Third-party API for job fraud analysis and verification	https://ruvia.io/trust-api
Stripe API	Secure payment processing and credit purchase management	https://stripe.com/docs/api
LinkedIn/Indeed APIs	Job feed aggregation and job posting data retrieval	https://developer.linkedin.com/
spaCy	NLP library for resume parsing and text analysis	https://spacy.io/
HuggingFace Transformers	Pre-trained models for job matching and fraud detection	https://huggingface.co/
OAuth2	Authentication protocol supporting Google and LinkedIn login	https://oauth.net/2/
AWS / Vercel	Cloud hosting infrastructure for backend and frontend deployment	https://aws.amazon.com/ https://vercel.com/

1.3 Documentation Overview

In section 2 we document the requirements for the JobTrust application. In sections 3-7 we use the 4+1 View Model to document the JobTrust application architecture, which involves the construction of 5 views focusing on different concerns about the software architecture. We use different types of UML diagrams to document each view.

1.4 Application Usage

Dashboard Web Application:
The dashboard can be accessed through a web browser at the deployed URL. Users authenticate using OAuth2 (Google or LinkedIn), then access features including profile management, credit balance viewing, job analysis, and resume recommendations

Browser Widget:
The browser extension widget can be installed from the Chrome Web Store. Once installed, users can navigate to any job posting website (LinkedIn, Indeed, etc.) and click the JobTrust widget icon to analyze jobs directly from the current page. All data synchronizes with the dashboard in real-time.

Configuration File:
The backend can be run with the command `python3 main.py`. The backend requires a configuration file (config.cfg or environment variables) with the following settings:

```
[Database]
postgresql_url=postgresql://user:password@localhost:5432/jobtrust

[APIs]
ruvia_api_key=your_ruvia_api_key_here
stripe_secret_key=sk_test_your_stripe_secret_key
linkedin_api_key=your_linkedin_api_key
indeed_api_key=your_indeed_api_key

[OAuth2]
google_client_id=your_google_client_id
google_client_secret=your_google_client_secret
linkedin_client_id=your_linkedin_client_id
```

linkedin_client_secret=your_linkedin_client_secret

[Logging]

file=TRUE

database=TRUE

console=FALSE

log_filename=jobtrust.log

2. Requirements

The JobTrust application was built to satisfy the following requirements:

	Functional Requirements
R01	Users shall be able to authenticate using OAuth2 providers (Google and LinkedIn) with secure token-based session management.
R02	Users shall be able to analyze job postings for fraud by providing a job URL or description, receiving a fraud score (0-100) with detailed fraud indicators.
R03	Job fraud analysis shall utilize both ML-based models and the Ruvia Trust API to detect suspicious patterns including language analysis, company verification, salary anomalies, and contact information validity.
R04	Users shall be able to calculate personalized job matching scores (0-100) by comparing their resume and profile against job requirements, with detailed breakdowns of skill alignment, experience compatibility, and education matching
R05	Users shall receive AI-powered resume and skill improvement recommendations based on target job requirements, including skill gap identification, content improvements, formatting suggestions, and ATS optimization.
R06	Each user account shall be initialized with 50 free credits upon registration, with credits consumed as follows: fraud detection (2 credits per job), job matching (3 credits per job), and resume recommendations (5 credits per request).
R07	Users shall be able to purchase additional credits through Stripe payment integration with packages of 100 credits ($9.99), 500 credits ($39.99), and 1000 credits ($69.99).
R08	Users shall be able to bookmark jobs, add personal notes, update application status, and view their complete job tracking history.
R09	The browser widget and dashboard shall maintain real-time data synchronization with

	updates propagating within 2 seconds.
R10	Users shall be able to upload and manage multiple resume versions with automatic NLP-based parsing of skills, experience, and education.
R11	The system shall aggregate job postings from LinkedIn and Indeed APIs with automatic daily updates.
R12	All user data including resumes, profiles, and job history shall be encrypted at rest using AES-256 encryption and in transit using TLS 1.3.
R13	The application shall log its activity to any combination of console, file, and database for debugging and audit purposes
R14	A configuration file shall be used to specify database connection strings, API keys, OAuth2 credentials, and logging settings.
	Performance Requirements
R15	Job fraud analysis requests shall complete within 3 seconds for 95% of requests (95th percentile latency).
R16	Job matching score calculations shall complete within 2 seconds for resumes up to 10 pages in length.
R17	The dashboard shall load initial page content within 1.5 seconds on broadband connections.
R18	The system shall support at least 1000 concurrent users without performance degradation.
	Performance Requirements
R19	All API endpoints shall require valid JWT authentication tokens with 24-hour expiration times.
R20	Payment processing shall comply with PCI DSS standards through Stripe's secure payment gateway.
R21	The application shall implement rate limiting of 100 requests per minute per user to prevent abuse.
R22	All third-party API keys shall be stored securely using environment variables or secret management services, never hardcoded in source code.
	Scalability Requirements
R23	The database schema shall support horizontal scaling to accommodate growth to 100,000+ users.

R24	The backend API shall be stateless to enable load balancing across multiple server instances.
	Traceability Requirements
R25	At least 70% of all application methods should create a log entry at runtime.

3. Scenario View

Concerns: Understanding the central functionality of the system
Stakeholders: All stakeholders, but particularly the end user
Modelling Techniques: UML Use Case Diagram

The primary user actions in JobTrust include analyzing job postings for fraud, calculating job match scores, getting resume recommendations, managing credits, and tracking job applications. Each of these use cases requires user authentication via OAuth2 as a prerequisite.

Primary Use Cases:
Analyze Job Posting: Users provide a job URL or description to receive a fraud score (0-100) with detailed indicators. This use case includes "Authenticate User" and "Check Fraud Indicators" as dependent operations.
Calculate Job Match Score: Users compare their resume against job requirements to receive a personalized compatibility score. This includes "Authenticate User" and "Compare Skills & Experience" as included operations.
Get Resume Recommendations: Users receive AI-powered suggestions for improving their resume based on target job requirements. This includes "Authenticate User" and "Generate Improvement Suggestions" as included operations.
Manage Credits: Users view their credit balance, purchase history, and buy additional credits through Stripe. This includes "Authenticate User" as a prerequisite, with "Purchase Credits" as an extending operation.
Track Job Applications: Users bookmark jobs, update application status, and monitor their job search history. This includes "Authenticate User" with "Bookmark Jobs" and "Update Application Status" as extending operations.

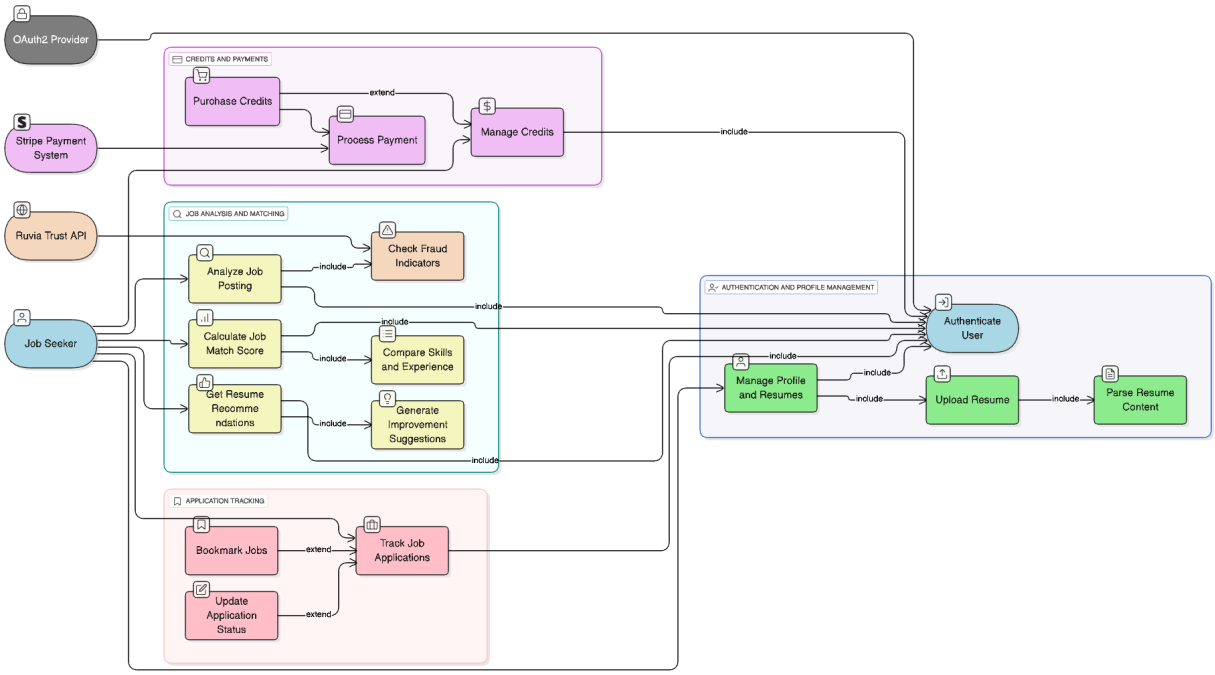

Figure 1: Use Case Diagram (Scenario View)

4. Physical View

Concerns: Mapping of software to hardware, communication protocols and modules related to communication

Stakeholders: Software architect, software developers

Modelling Techniques: UML Deployment Diagram

The JobTrust application follows a cloud-native architecture with clear separation between client-side, server-side, database, and external service components.

Client Browser Node: Hosts the React dashboard (web application) and browser extension widget (Manifest V3 package). The client communicates with the backend server exclusively over HTTPS (port 443) with TLS 1.3 encryption.

AWS/Vercel Server Node: Hosts the FastAPI backend which serves as the central orchestration layer for business logic, API routing, credit management, and authentication. The server is deployed on either AWS (EC2/Lambda) or Vercel platform with auto-scaling capabilities

Database Server Node: Uses PostgreSQL for relational data (users, transactions, job analyses) and MongoDB Atlas for document storage (resumes, job descriptions, detailed reports). Communication between the application server and databases occurs over TCP with SSL/TLS encryption on ports 5432 (PostgreSQL) .

External Services Node: Integrates with Ruvia Trust API (fraud detection), Stripe API (payments), LinkedIn/Indeed APIs (job aggregation), and OAuth2 providers (Google/LinkedIn authentication). All external service communication uses HTTPS/REST protocols with API key authentication.

Real-time Updates: WebSocket connections (WSS) enable synchronization between the dashboard and widget.

See Figure 2 for the Deployment Diagram.

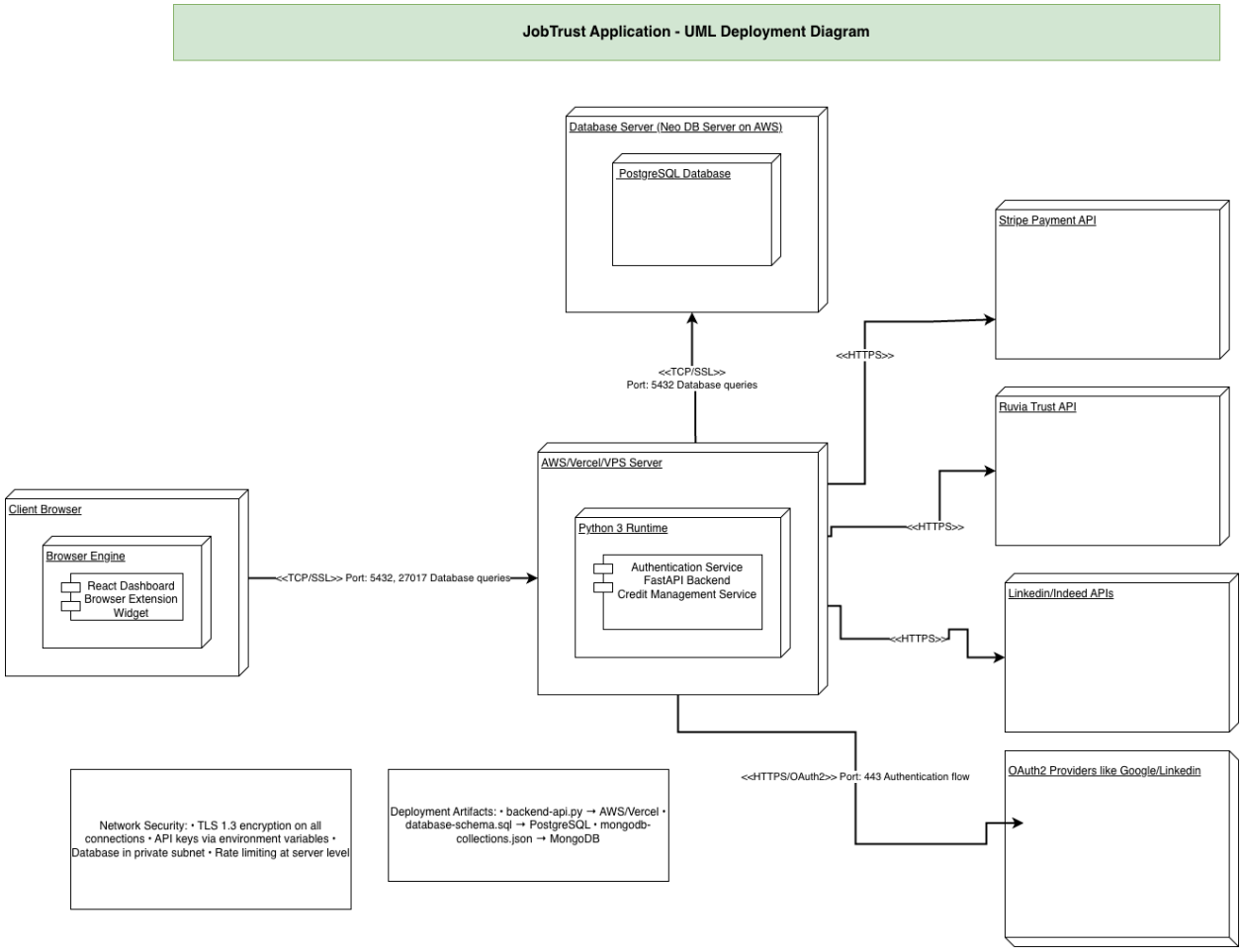

Figure 2: UML Deployment Diagram (Physical View)

5. Development View

Concerns: Organization of software modules
Stakeholders: Software developers, manager
Modelling Techniques: UML Class Diagram

The **App** singleton object is used to read the configuration file as well as keep track of all read-only application state, including database connections (PostgreSQL and MongoDB), Stripe API client, and a reference to the chain-of-loggers used by the rest of the classes to log application activity. The chain-of-loggers is implemented using a **Chain-of-Responsibility** pattern, with each logger instantiated and referencing the next logger according to whether the configuration file indicates the logger should be used.

The **Model-View-Controller** pattern is used to handle user interaction, with the view presenting dashboard/widget interfaces and returning user input to the controller. The view has no state in terms of instance or class variables, so its methods are implemented as static methods to recognize and help enforce this property. The controller executes view methods to obtain user input and calls model functions to handle any database interactions

The **Strategy** pattern is implemented for fraud analysis and job matching algorithms. The FraudAnalyzer interface defines the contract for fraud detection, with MLFraudAnalyzer (machine learning-based) and APIFraudAnalyzer (Ruvia Trust API-based) as concrete implementations. This allows the system to switch between detection methods based on credit availability, API status, or accuracy requirements. Similarly, JobMatcher uses the Strategy pattern with MatchingStrategy implementations (SkillBasedMatcher, ExperienceBasedMatcher, ComprehensiveMatcher) for flexible job matching algorithms.

Design Pattern Summary

Pattern	Role	Classes
Singleton	Maintains application read-only state, including connections to databases, APIs, and logger	App
Chain-of-Responsibility	Logging to database, console and/or file	Logger, ConsoleLogger, FileLogger, DatabaseLogger
Strategy	Pluggable fraud detection and job matching algorithms	FraudAnalyzer, MLFraudAnalyzer, APIFraudAnalyzer, MatchingStrategy, SkillBasedMatcher, ExperienceBasedMatcher, ComprehensiveMatcher
Model-View-Controller	User interaction	Model, View, Controller

See Figure 3 for the UML Class Diagram.

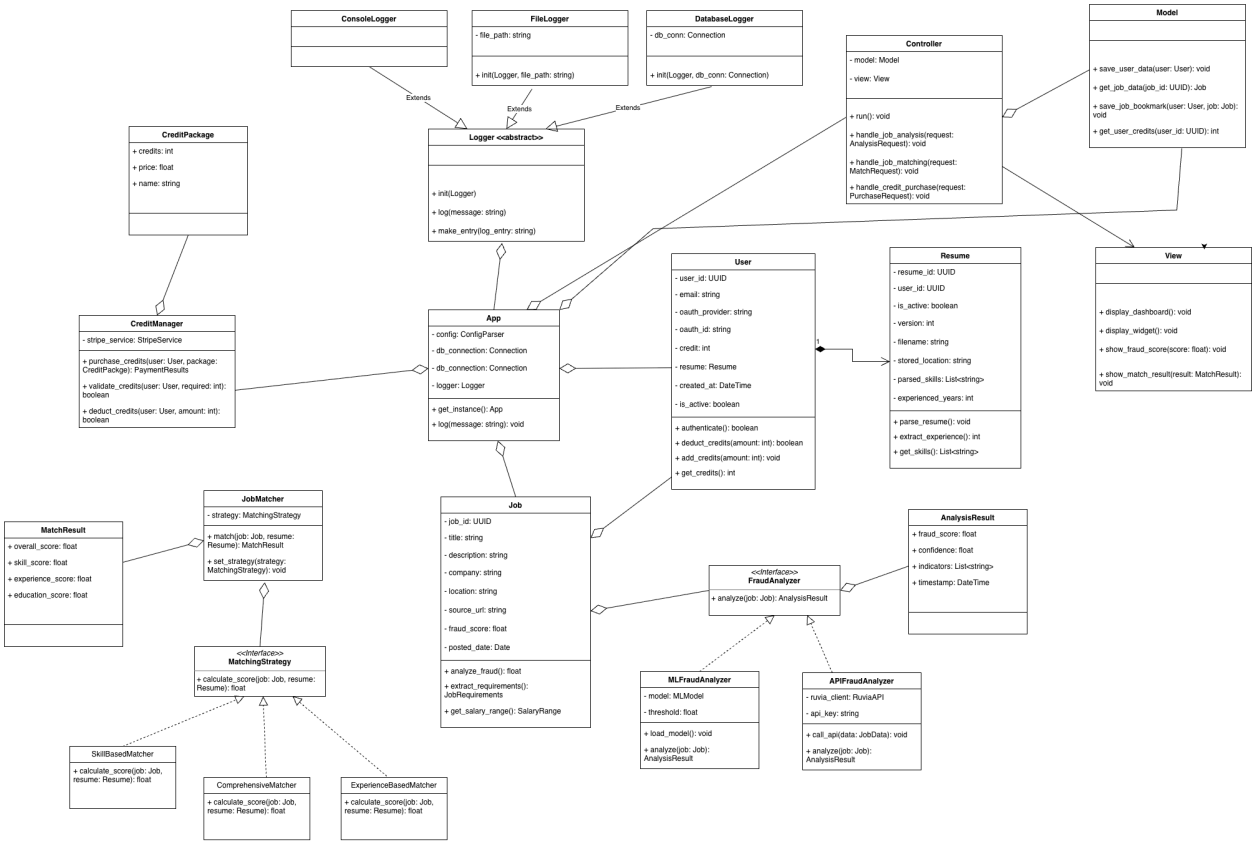

Figure 3: UML Class Diagram (Development View)

6. Process View

Concerns: Runtime communication

Stakeholders: Software architect

Modelling Techniques: UML Sequence Diagram

The most important runtime communication occurs between the application and the database systems (PostgreSQL and MongoDB), between the application and external APIs (Ruvia Trust API, Stripe API), and within the MVC architecture.

Job Fraud Analysis Sequence:

When analyzing a job posting, the following sequence occurs:

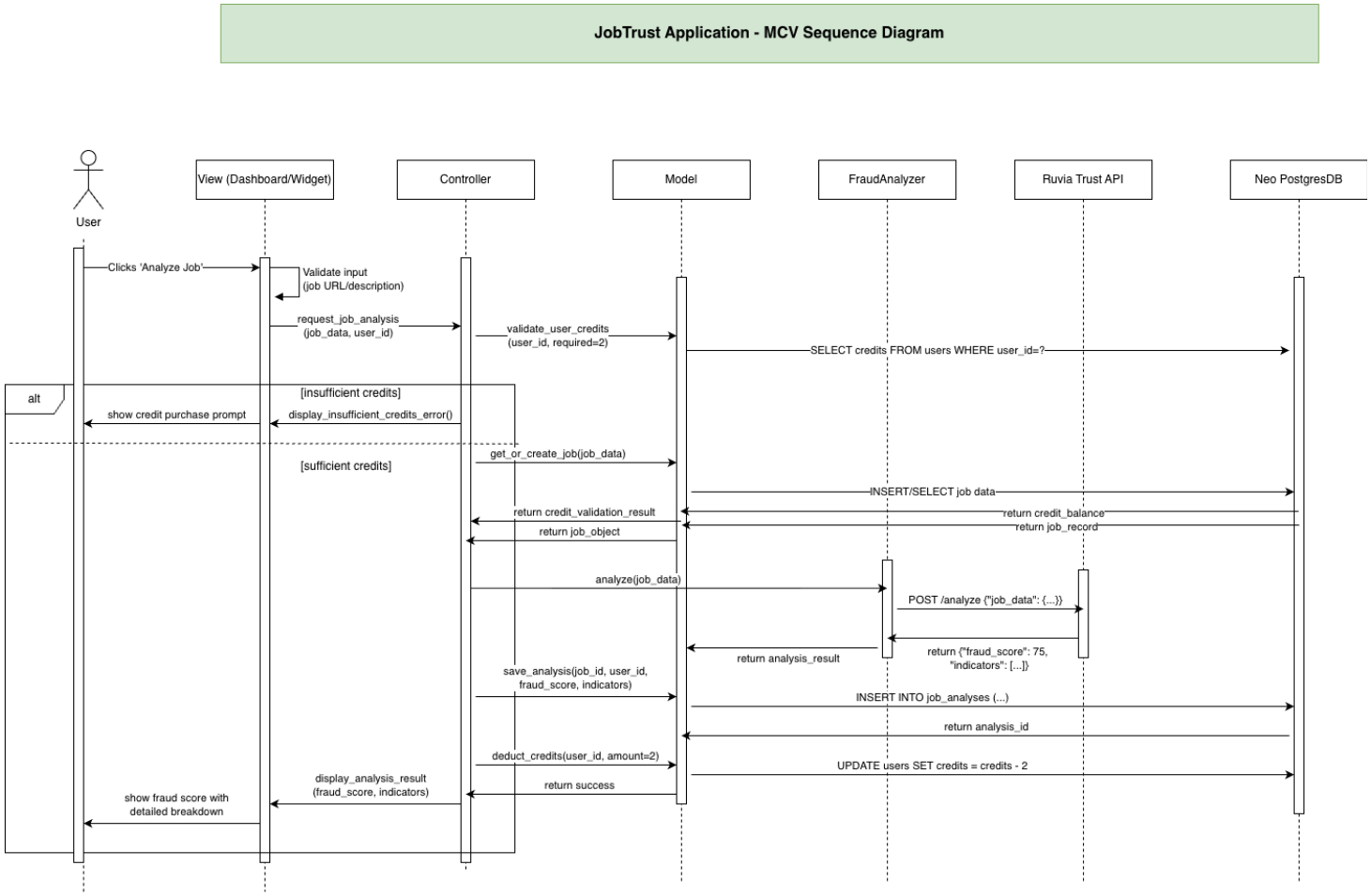

Figure 4: UML Sequence Diagram for job fraud analysis sequence

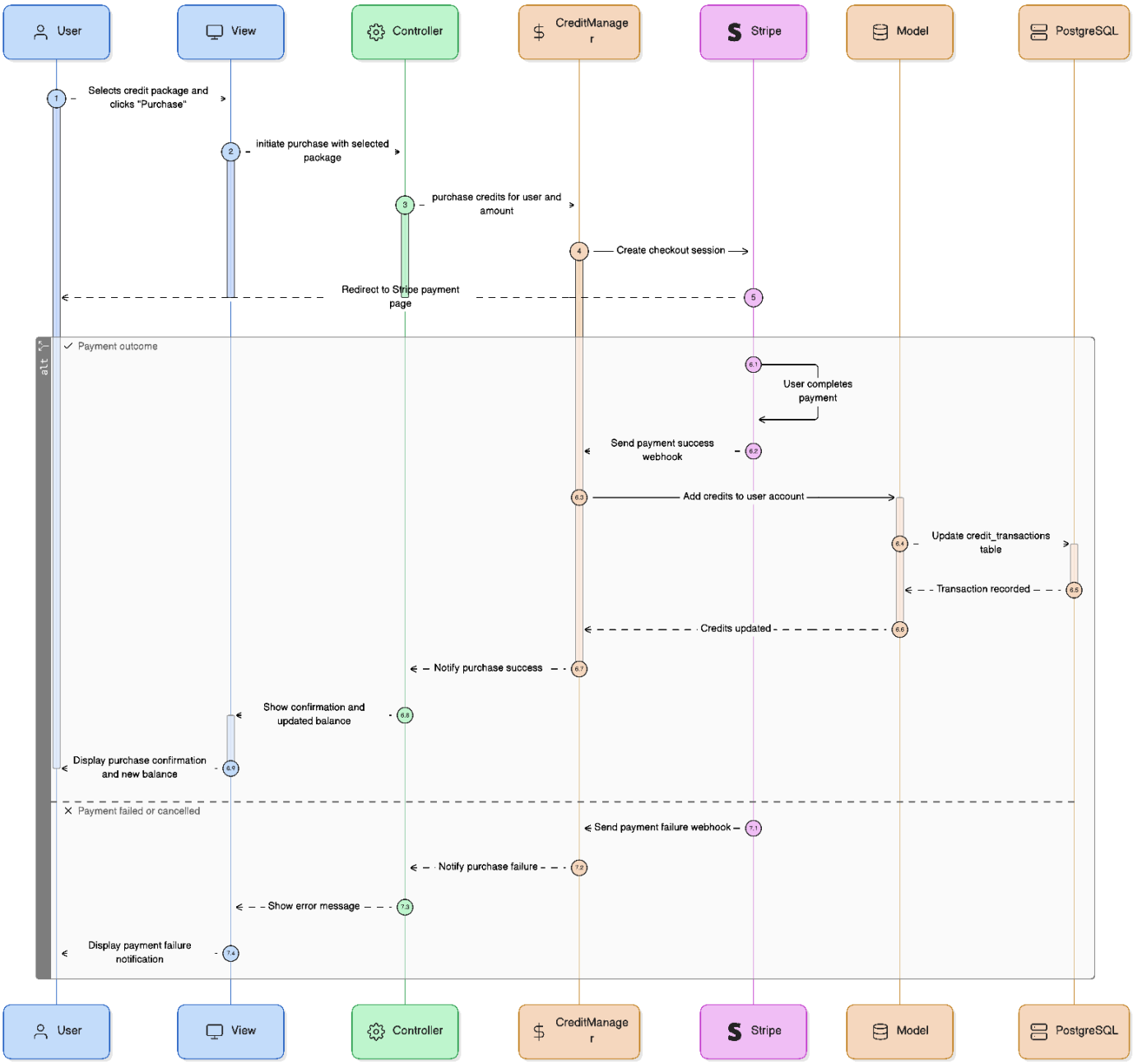

Figure 5: UML Sequence Diagram for credit purchase flow

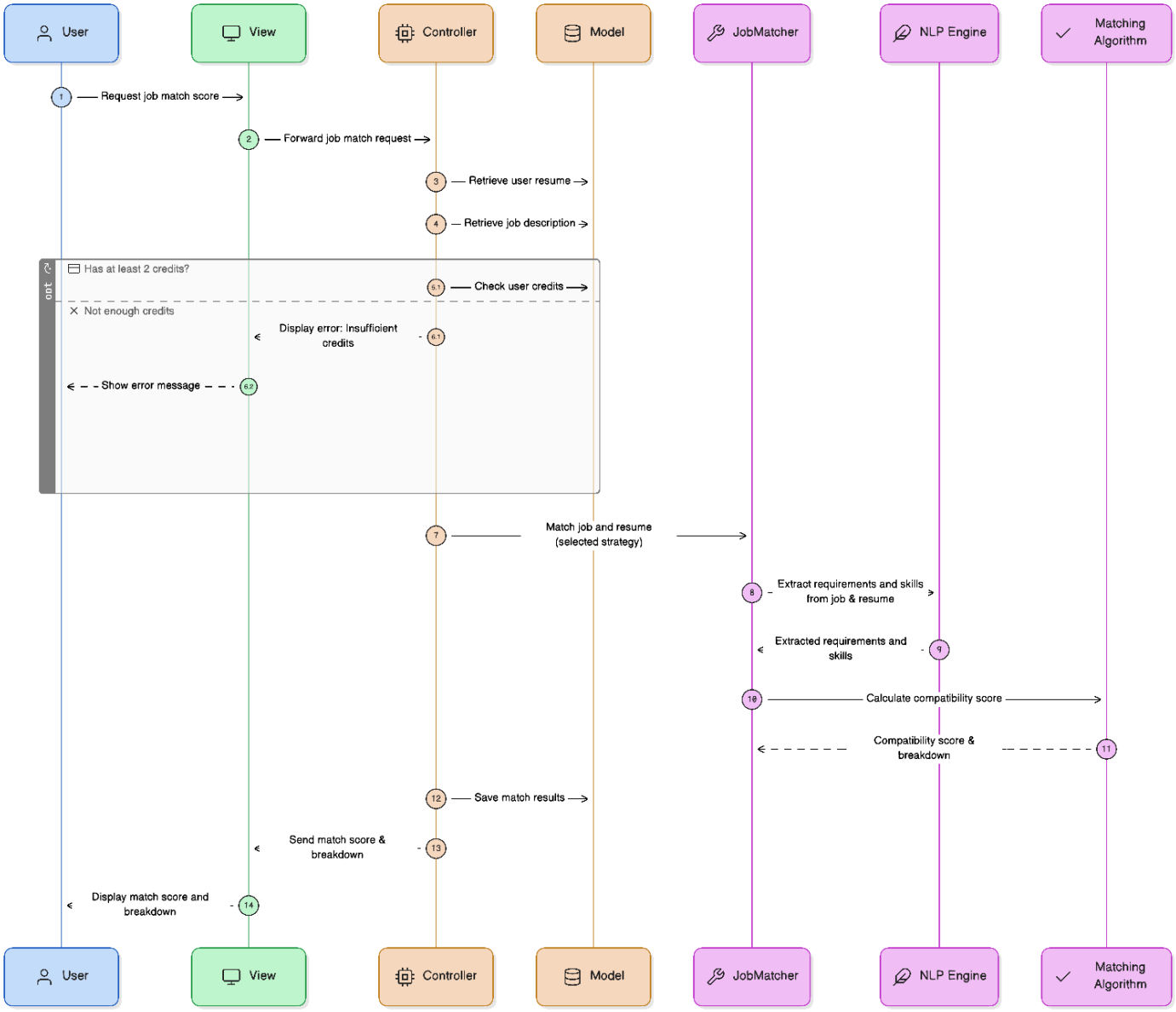

Figure 6: UML Sequence Diagram for Job Matching flow.

7. Logical View

Concerns: Functional requirements
Stakeholders: End user, software architect
Modelling Techniques: UML Activity Diagram

The use cases are realized via a series of dashboard pages and widget interfaces that allow users to choose between analyzing jobs, calculating match scores, getting resume recommendations, managing credits, and tracking applications. The primary workflow is implemented as both a web application (dashboard) and browser extension (widget), with text and graphical interfaces for all interactions.

Main User Workflow:
Users begin at the **Main Menu** (dashboard home or widget interface) where they can choose from several high-level actions:
Option 1: Analyze Job Posting

- Input job URL or description
- System checks credit balance (2 credits required)
- If insufficient credits → redirect to Purchase Credits flow
- If sufficient → select analysis type (Quick ML vs Detailed API)
- Process job analysis
- Display fraud score with indicators
- Options: Save to bookmarks, Get match score, Return to main menu

Option 2: Calculate Job Match Score

- Select job (from bookmarks or enter new)
- Select resume version
- System checks credit balance (3 credits required)
- If insufficient credits → redirect to Purchase Credits flow
- If sufficient → select matching strategy (skill/experience/comprehensive)
- Process matching algorithm
- Display match score with breakdown

- Options: Get resume recommendations, Save match report, Return to main menu

Option 3: Get Resume Recommendations

- Select resume version
- Optionally select target job
- System checks credit balance (5 credits required)
- If insufficient credits → redirect to Purchase Credits flow
- If sufficient → AI analyzes resume
- Display categorized recommendations (skills/content/formatting/ATS)
- Options: Apply suggestions, Save recommendations, Return to main menu

Option 4: Manage Credits

- View credit dashboard (balance, usage history, transactions)
- Options: View usage history, Purchase credits, Return to main menu
- Purchase Credits flow: Select package → Stripe payment → Confirmation

Option 5: Track Jobs

- View bookmarked jobs with filters/sorting
- Job actions: View details, Update status, Add notes, Remove bookmark
- View application history timeline
- Return to main menu

Option 6: Settings & Profile

- Manage profile, resumes, preferences, and security settings
- Return to main menu

Quit/Logout

- Exit application with session cleanup

The sequence of menus and options is captured in Figure 7.

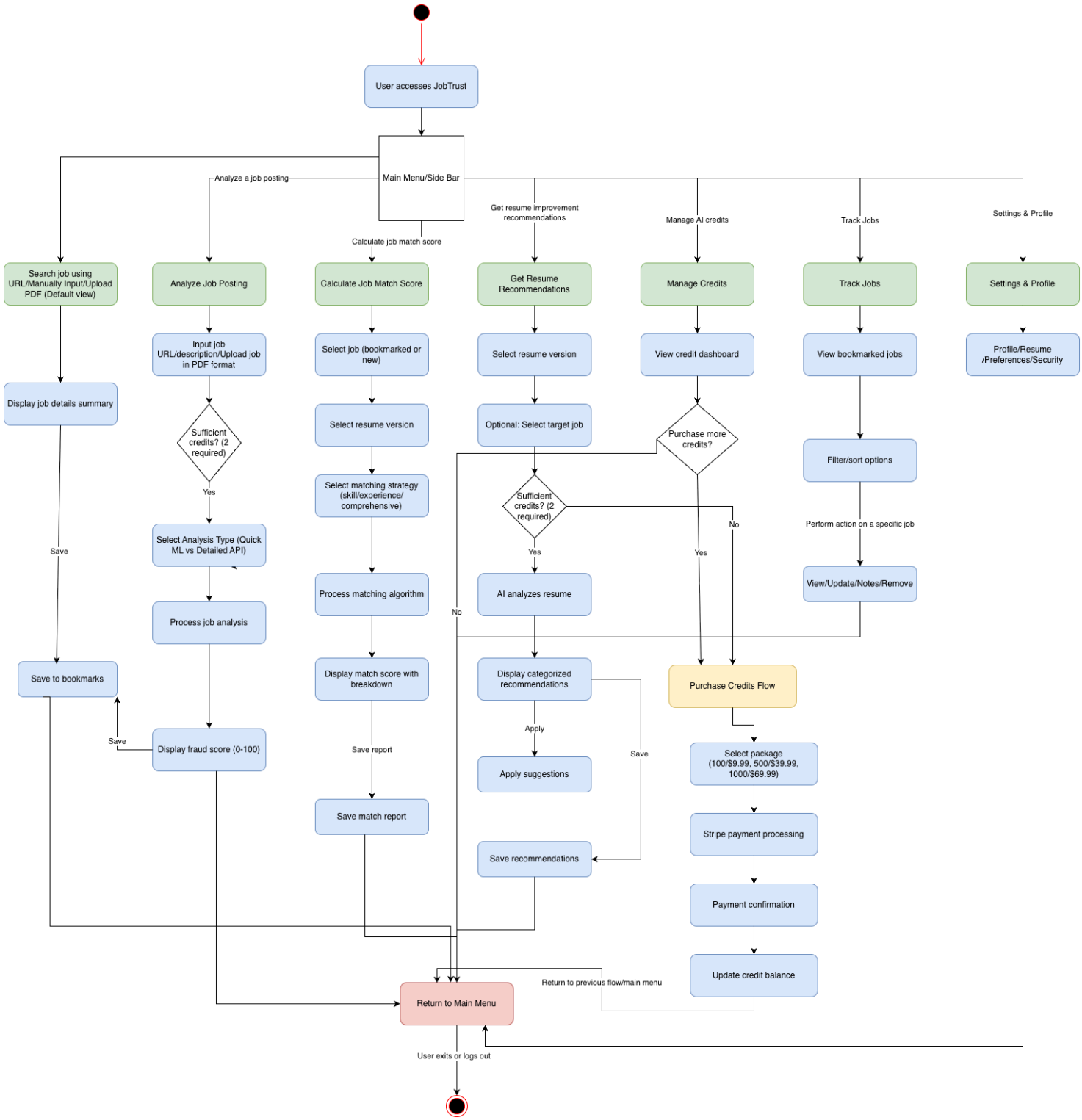

Figure 7: UML Activity Diagram

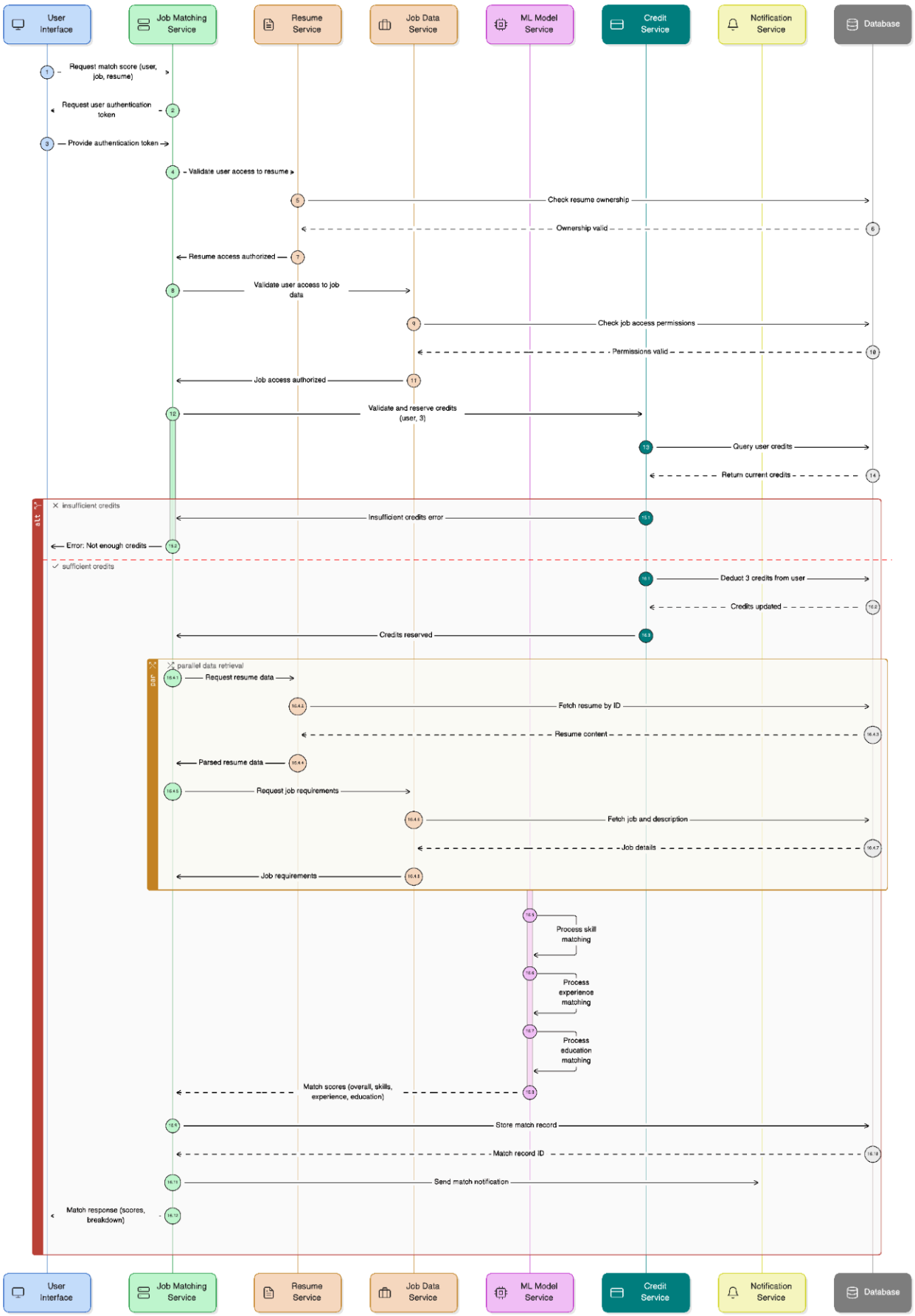
User Interface
Job Matching Service
Resume Service
Job Data Service
ML Model Service
Credit Service
Notification Service
Database
1 Request match score (user, job, resume)
2 Request user authentication token
3 Provide authentication token
4 Validate user access to resume
5 Check resume ownership
6 Ownership valid
7 Resume access authorized
8 Validate user access to job data
9 Check job access permissions
10 Permissions valid
11 Job access authorized
12 Validate and reserve credits (user, 3)
13 Query user credits
14 Return current credits
alt
insufficient credits
15.1 Insufficient credits error
15.2 Error: Not enough credits
sufficient credits
16.1 Deduct 3 credits from user
16.2 Credits updated
16.3 Credits reserved
par
parallel data retrieval
16.4.1 Request resume data
16.4.2 Fetch resume by ID
16.4.3 Resume content
16.4.4 Parsed resume data
16.4.5 Request job requirements
16.4.6 Fetch job and description
16.4.7 Job details
16.4.8 Job requirements
16.5 Process skill matching
16.6 Process experience matching
16.7 Process education matching
16.8 Match scores (overall, skills, experience, education)
16.9 Store match record
16.10 Match record ID
16.11 Send match notification
16.12 Match response (scores, breakdown)

Figure 8: Services Sequence Diagram

8. Database Schema

PostgreSQL is used for structured, transactional data with strong consistency requirements.

Table: users

Column	Type	Constraints	Description
user_id	UUID	PRIMARY KEY	Unique user identifier
email	VARCHAR(255)	UNIQUE, NOT NULL	User email address
oauth_provider	VARCHAR(50)	NOT NULL	OAuth provider (google, linkedin)
oauth_id	VARCHAR(255)	NOT NULL	OAuth provider user ID
credits	INTEGER	DEFAULT 50, NOT NULL	Current credit balance
created_at	TIMESTAMP	DEFAULT NOW()	Account creation timestamp
is_active	BOOLEAN	DEFAULT TRUE	Account status

Table: credit_transactions

Column	Type	Constraints	Description
transaction_id	UUID	PRIMARY KEY	Unique transaction identifier
user_id	UUID	FOREIGN KEY → users(user_id)	User reference
transaction_type	VARCHAR(50)	NOT NULL	Type: purchase, deduction, refund

amount	INTEGER	NOT NULL	Credit amount (positive/negative)
stripe_payment_id	VARCHAR(255)		Stripe payment intent ID
created_at	TIMESTAMP	DEFAULT NOW()	Transaction timestamp

Table: job_analyses

Column	Type	Constraints	Description
analysis_id	UUID	PRIMARY KEY	Unique analysis identifier
user_id	UUID	FOREIGN KEY → users(user_id)	User who requested analysis
job_id	UUID	FOREIGN KEY → jobs(job_id)	Job being analyzed
fraud_score	DECIMAL(5,2)	CHECK (0-100)	Fraud probability score
analysis_type	VARCHAR(50)	NOT NULL	Type: ml_model, api_based
credits_used	INTEGER	DEFAULT 2	Credits consumed
created_at	TIMESTAMP	DEFAULT NOW()	Analysis timestamp

Table: job_matches

Column	Type	Constraints	Description
match_id	UUID	PRIMARY KEY	Unique match identifier
user_id	UUID	FOREIGN KEY → users(user_id)	User who requested match

job_id	UUID	FOREIGN KEY → jobs(job_id)	Job being matched
resume_id	UUID	NOT NULL	Resume version used
match_score	DECIMAL(5,2)	CHECK (0-100)	Overall match score
skill_score	DECIMAL(5,2)		Skill alignment score
experience_score	DECIMAL(5,2)		Experience match score
matching_strategy	VARCHAR(50)	NOT NULL	Algorithm used
credits_used	INTEGER	DEFAULT 3	Credits consumed
created_at	TIMESTAMP	DEFAULT NOW()	Match timestamp

Table: job_bookmarks

Column	Type	Constraints	Description
bookmark_id	UUID	PRIMARY KEY	Unique bookmark identifier
user_id	UUID	FOREIGN KEY → users(user_id)	User who bookmarked
job_id	UUID	FOREIGN KEY → jobs(job_id)	Bookmarked job
status	VARCHAR(50)	DEFAULT 'saved'	Status: saved, applied, interview, offer, rejected
notes	TEXT		User notes
created_at	TIMESTAMP	DEFAULT NOW()	Bookmark creation
updated_at	TIMESTAMP	DEFAULT NOW()	Last status update

Table: jobs

Column	Type	Constraints	Description
job_id	UUID	PRIMARY KEY	Unique job identifier
title	VARCHAR(500)	NOT NULL	Job title
company	VARCHAR(255)	NOT NULL	Company name
location	VARCHAR(255)		Job location
source	VARCHAR(100)	NOT NULL	Source: linkedin, indeed, manual
source_url	TEXT		Original job posting URL
posted_date	DATE		Job posting date
created_at	TIMESTAMP	DEFAULT NOW()	Record creation timestamp

Table: logs

Column	Type	Constraints	Description
log_id	BIGSERIAL	PRIMARY KEY	Auto-incrementing log entry ID
timestamp	TIMESTAMP	DEFAULT NOW(), NOT NULL	When the log entry was created
level	VARCHAR(20)	NOT NULL	Log level: INFO, WARNING, ERROR, DEBUG
message	TEXT	NOT NULL	The log message content
user_id	UUID		Optional user reference for user-specific actions
action	VARCHAR(100)		Action type: job_analysis, credit_purchase, login, etc.
details	JSONB		Additional structured log data