EECS4314

Project Proposal

Group 6

Rajendra Brahmbhatt - 217925157

Mai Komar - 218590935

Harnaindeep Kaur - 217621137

Michael Byalsky - 219026202

Lakshan Kandeepan - 216440521

Daniel Chahine - 219598994

Table of Contents

Q1. Name of your product 3
Q2. Briefly describe the core concept of your application in terms of user functionality 3
Q3. List the names of each person in the project group 3
Q4. List programming languages and web development frameworks that you know or want to use 3
Q5. Provide a rough diagram or description of your system architecture, which project requirements you strive to implement, and how this fulfils the learning goals for this course 4
Core Requirements 5
Optional Requirements 5
Architecture and Risk Management 6
More About the Product 7

Q1. Name of your product.

YU Trade

Q2. Briefly describe the core concept of your application in terms of user functionality.

Brief description

The application will be an online marketplace designed specifically for all current **York University** members (Students, Faculty, Professors). The app allows valid members to post items for sale, and have the items be purchased by other members of the website, similar to Facebook Marketplace.

We can limit access to verified York University users by enforcing registration through @my.yorku.ca (students) and @yorku.ca (faculty) email addresses. Once authenticated, users can create listings, browse items posted by others, and communicate directly with sellers through private messaging that is associated with a product.

Main Objectives

- Basic Login/Registration & profile/portfolio view
- Create item listings with title and product fields. Fields will include price (firm amount, negotiable), product description, item status (Available, Pending Sale, Sold) and item pictures.
- Buyers can message the seller for the product (messages should be tied to a specific product)

Q3. List the names of each person in the project group.

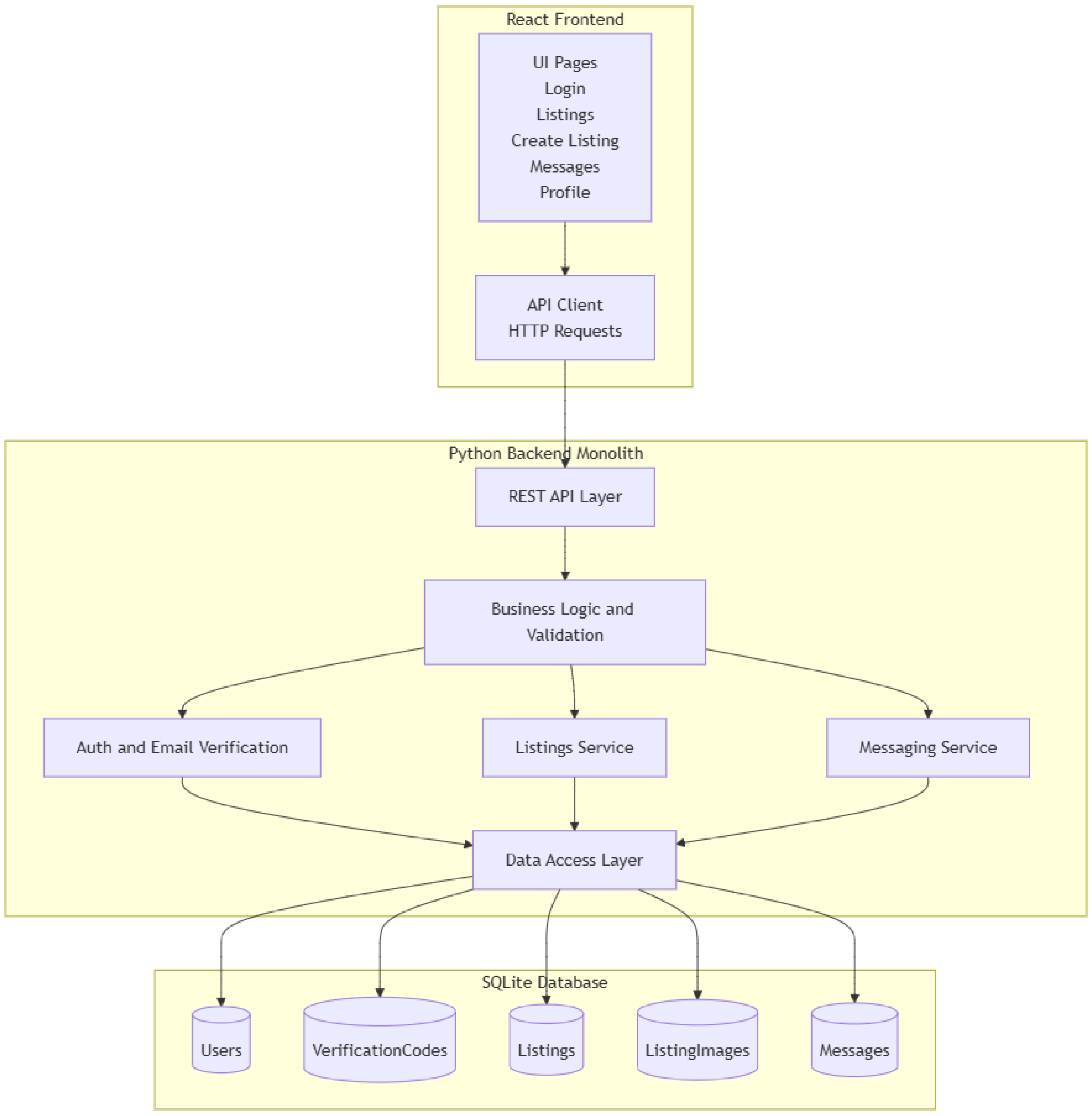
- Rajendra Brahmbhatt | raj1308@my.yorku.ca | 217925157
- Michael Byalsky | mickeyb@my.yorku.ca | 219026202
- Daniel Chahine | chahined@my.yorku.ca | 219598994
- Lakshan Kandeepan | lakshan@my.yorku.ca | 216440521
- Harnaindeep Kaur | harnain33@gmail.com | 217621137
- Mai Komar | maik@my.yorku.ca | 218590935

Q4. List programming languages and web development frameworks that you know or want to use.

- Frontend: React (TypeScript)
- Backend: Python (FastAPI or Flask)
- Database: SQLite

Q5. Provide a rough diagram or description of your system architecture, which project requirements you strive to implement, and how this fulfils the learning goals for this course.

Component Diagram: Layered Architecture Diagram



YU Trade follows a monolithic web architecture. The React frontend communicates with a single Python back-end through REST APIs. The back-end encompasses all core application logic, including authentication, listing management, and private messaging. Persistent data is stored in a SQLite database.

This architecture is illustrated through UML use-case, class, and sequence diagrams, which model user interactions, data relationships, and request flows between the frontend, backend, and data store.

The monolithic approach reduces integration overhead, simplifies deployment, and aligns with the course objective of understanding architectural trade-offs in a controlled, small-team environment.

Why not Microservices?

Microservice is not well-suited for this project because it would introduce unnecessary complexity for a small team working within a limited academic timeframe. It would require managing multiple independently deployed services, how the services communicate with each other, difficult debugging, and data consistency across separate components, which significantly increases development and testing overhead.

Since YU Trade's core features, authentication, listings, and product-specific messaging, are tightly coupled and built using a single technology stack, the benefits of microservices (such as independent scaling and deployment) would not be fully realized. As a result, a microservice approach would increase project risk and slow feature delivery, making a monolithic architecture a more appropriate and manageable choice for this course project.

Core Requirements

- User registration and login with York University email verification
- Each product listing has a status: Available, Pending, or Sold. Sellers can update the status to reflect the item's availability. This helps buyers clearly understand the status of the item.
- Ability for users to post items for sale, with fields such as title, description, item status (available, sold, pending), price (firm amount vs negotiable), and images
- Ability for users to view listings posted by others
- Product-specific messaging between buyers and sellers
- Basic database persistence for users, listings, and messages
- Search and filtering of listings

Optional Requirements

- User profile enhancements
- Improved UI/UX or performance optimizations
- Timestamped messaging with message ordering
- Basic analytics (Example: number of views per listing)
- Accessibility improvements (keyboard navigation, readable contrast)

Architecture and Risk Management

The system will primarily follow a monolithic architecture, where the frontend, backend, and database interact as a single cohesive system. This architectural choice reduces initial complexity and is appropriate for a team of 6 members working within a limited timeframe. Preliminary risks that may be encountered during this project include:

- **Time constraints:** A monolithic system requires careful coordination of core features before deployment. Delays in one component (e.g., authentication or messaging) can impact overall progress if not managed incrementally.
- **Frontend–backend integration challenges:** Because the frontend and backend are tightly coupled within a single system, mismatches in API contracts, data formats, or request handling can lead to integration issues if interfaces are not clearly defined early.
- **Uneven workload distribution:** In a monolithic codebase, some components may become more complex or central than others, increasing the risk that certain team members take on disproportionate responsibility.

To mitigate these risks:

- Development tasks will be divided evenly among team members.
- The project will be broken into smaller tasks with weekly milestones.
- Progress will be reviewed regularly to ensure deadlines are met.

Learning goals:

Our project will fulfill several learning goals in this course, including understanding the differences between different types of software architecture, identifying risks associated with a given software project & developing plans to reduce those risks. In addition, we will learn to manage a software project by identifying the sequence of tasks that will enable the project to be completed on time while delegating tasks accordingly. Our project will primarily follow a monolithic architecture, and the team will evaluate risks associated with this architecture style throughout the development period. Our project will be developed incrementally, and the development of our project will be clearly delegated amongst our team members to ensure that it's completed on time.

Learning Outcome 1: Derive models of software systems and express them in a language such as UML.

We require the system to be modelled along with implementation. Core features can be represented using UML diagrams, including use case, class, and sequence diagrams. This helps translate requirements into system components and define how the frontend, backend, and database interact. This process demonstrates the ability to formally model a software system.

Learning Outcome 2: Understand the differences between different types of software architecture.

We demonstrate software architectural understanding through the selection of a monolithic architecture. By implementing the application as a single system, we experience the pros of reduced complexity and simpler deployment for a small team. This also encourages comparison with microservices and other architectural styles to evaluate trade-offs related to scalability, maintainability, and deployment.

Learning Outcome 3: Identify risks associated with a given software project and develop plans to mitigate and manage.

We identified key project risks like time constraints, integration challenges, and uneven workload distribution. These risks are mitigated through task decomposition, weekly milestones, and balanced responsibility among team members. Regular progress reviews further help manage potential issues. This approach reflects practical risk identification and mitigation strategies used in real software projects.

Learning Outcome 4: Manage software projects by identifying the sequence of tasks needed to complete the project on time and assigning responsibility for each task.

We demonstrate effective project management by organizing development into a sequence of tasks, starting with core functionality and followed by optional enhancements. Responsibilities are distributed across team members to enable parallel development and timely completion. Milestones and incremental progress tracking ensure the project remains on schedule.

More About the Product

• What problems does it solve?

YU Trade provides the York University community with a platform to sell used clothing, books, school supplies and much more. This helps to reduce the overall waste that the York University community introduces into the environment by encouraging recycling and thrifting. Furthermore, York University students can benefit from reduced prices for expensive school supplies, supporting students who may need more financial assistance.

• What knowledge should the reader expect to gain after reading the design?

The reader will gain an understanding of how a web app is planned, structured, and managed using sound software engineering principles. The reader will learn how a monolithic web architecture can be designed for a small development team, including how frontend, backend, and database components interact within a cohesive system. Also, the reader will gain insight into how UML diagrams are used to model system behaviour and structure before implementation. The design also highlights common software project risks like time constraints, integration challenges, and workload distribution, and it also shows practical strategies for mitigating these risks through task decomposition, milestone planning, and responsibility delegation.