University of Toronto
Faculty of Applied Science and Engineering

MIE 350
Design and Analysis of Information Systems

Final Report

Team Number	Group 2
Topic Category	Consumer Information Site
Project Name	CampusTech Connect

Team Members		
Name	**Role**	**Email**
Ali Ahmad	FrontEnd	itsali.ahmad@mail.utoronto.ca
Kuval Brar	Presenter #1, FrontEnd, Editor #1	kuval.brar@mail.utoronto.ca
Nithilan Suresh	Presenter #2, BackEnd, Editor #2	nithilan.suresh@mail.utoronto.ca
Samrath Singh	Presenter #3, FrontEnd, Editor #3	samrathsingh@mail.utoronto.ca
Michael Furlano	Scrum Master, FrontEnd	michael.furlano@mail.utoronto.ca
Fengrui Yang	BackEnd	fengrui.yang@mail.utoronto.ca
Peter Bing	BackEnd	p.bing@mail.utoronto.ca
Isabella Liang	BackEnd	i.liang@mail.utoronto.ca

1.0 Executive Summary

CampusTech Connect is a peer-to-peer web application designed to address a common pain point among university students: the difficulty in accessing affordable, reliable laptops suited for academic use. Motivated by survey data showing that nearly half of students experience laptop malfunctions and over 39% struggle to run essential software, the team set out to create a trusted, campus-specific marketplace. Unlike general platforms such as Kijiji or Facebook Marketplace, CampusTech Connect offers academic relevance, student verification through university email, tailored filters based on program needs, and an intelligent comparison tool backed by real-time market data.

The development process followed a structured and user-centered methodology. The team began with stakeholder analysis, interviews, surveys, and internal brainstorming to define both functional and non-functional requirements. These included features such as user registration, secure authentication, wishlist tracking, laptop comparisons, and listing management. The system was designed with usability, security, performance, and maintainability in mind, supported by clear UML models, statecharts, activity diagrams, and use case definitions to represent system behavior and logic.

The backend was built using Java Spring Boot with frontend components implemented in a browser-based interface, AppSmith. While the application currently performs well on laptops, mobile optimization is planned for the next development phase, including a shift to mobile-first design and Progressive Web App (PWA) capabilities.

Testing efforts covered all major user workflows through manual and Spring mockMvc-based integration tests, ensuring endpoints performed as intended. These tests included registration, login, listing CRUD operations, and wishlist management, all of which passed successfully. The team developed a clear roadmap for future QA initiatives, including usability testing, security assessments, load simulation, and broader device compatibility. Informal usability testing with students helped the team refine the user interface and identify practical areas for improvement.

Throughout the project, the team encountered and overcame a range of technical and collaborative challenges such as JWT-based authentication, external API integration, version control issues, and feature misalignment among team members. The team learned to manage complexity through scoped planning, clearer communication, and regular check-ins.

Ultimately, this project served as a comprehensive learning experience in software engineering, covering requirement elicitation, system design, implementation, testing, and documentation. Beyond building a working product, CampusTech Connect helped the team strengthen their development skills, embrace user-centered thinking, and grow as collaborators and engineers.

2.0 Overview of web application: purpose, intended audience, content summary

CampusTech Connect is a web application created as a peer-to-peer laptop marketplace tailored for university students. The purpose of this platform is to connect student buyers and sellers of used laptops in a trusted, campus-focused environment. It was inspired by the recognition that many students struggle with inadequate devices, nearly half of students have faced laptop malfunctions, and 39% have been unable to run required academic software [1]. These challenges, coupled with 42% of students reporting financial difficulties upgrading devices highlight the need for a student-centric marketplace that bridges gaps in functionality, affordability, and trust [2]. The challenges students face with laptops is presented in Figure 1.

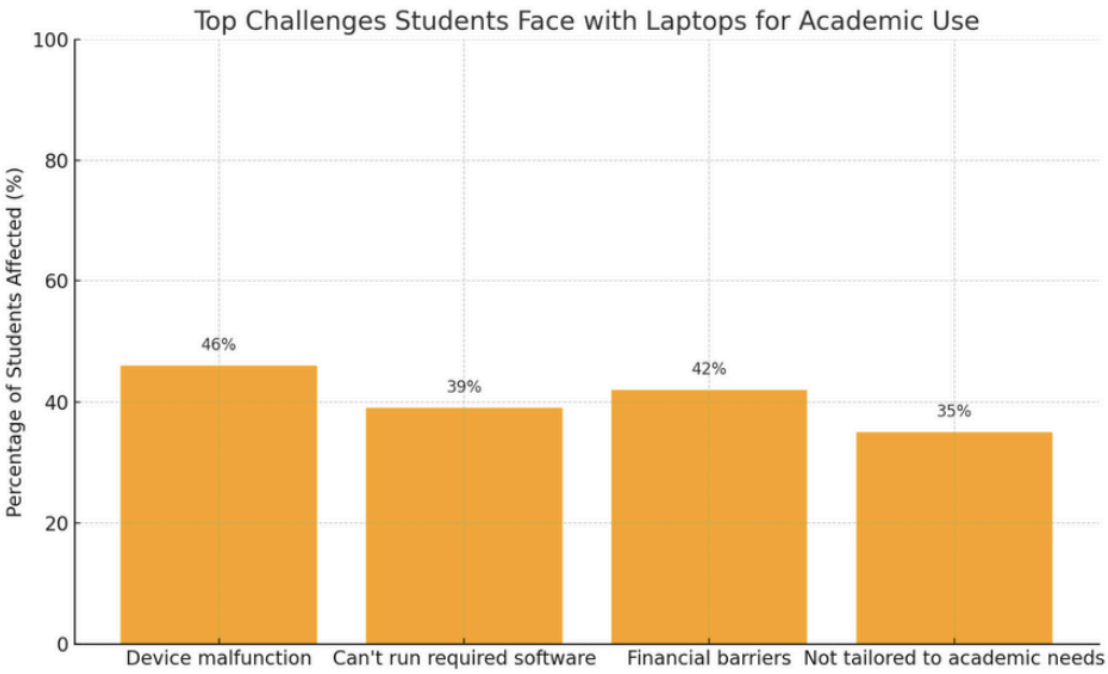

Figure 1. Top Challenges Students Face with Laptops for Academic Use.
(Data adapted from [1], [2])

Figure 1 illustrates that 46% of students experience device malfunctions during coursework and 39% cannot run required academic software. Additionally, 42% report financial constraints when purchasing or upgrading devices, and 35% feel their laptops aren't tailored to academic tasks. These systemic challenges justify the need for a targeted solution like CampusTech Connect which is designed specifically for students and their academic workflows.

The intended audience for CampusTech Connect is the university student community, specifically for students who want secure, budget-conscious, and performance-matched laptop options. Unlike general audiences on public marketplaces, the users share an academic context. The platform's content and features are designed with students' needs in mind. For example, a

computer science student might seek a laptop that can run programming IDEs and virtual machines, whereas a design student might need a high-resolution display and graphics capability. CampusTech Connect allows students to find devices suited to such academic requirements in a safe, peer-driven setting.

In terms of content and features, CampusTech Connect offers a comprehensive set of tools for buying and selling laptops. All users register with a verified university email, ensuring that only students can participate, which creates a foundation of trust. Sellers can create detailed laptop listings including specifications (CPU, RAM, storage, etc.), condition, and price. Buyers can browse and search listings with filters for specifications and price range, making it easy to find devices that meet specific course or program needs. A key feature is the smart comparison tool that lets users compare multiple listings side by side and even check against external market prices, helping students make informed decisions. Buyers can also save items to a wishlist and set target prices. For instance, a student can bookmark a laptop and indicate the price they are willing to pay, helping them track potential deals. The application provides an interactive dashboard for each user (buyer or seller) to manage their posts or saved items. In summary, the content of the app spans user accounts, laptop listings, wishlists, and comparison features, all presented through a user-friendly web interface.

CampusTech Connect was conceived to improve on the shortcomings of general marketplaces like Kijiji, Facebook Marketplace, or Craigslist for this use case. Those existing platforms are generic marketplaces that are not specifically designed for academics so they lack filters for student-specific requirements or academic needs. For example, on Kijiji or Craigslist, a buyer cannot filter used laptops by criteria like “suitable for engineering software” or easily compare detailed specs across postings. This platform fills that gap by offering filtering and recommendations tailored to academic use (e.g., by major or software requirements).

Trust is another major differentiator. Listings on open marketplaces typically aren’t verified or university-specific, leading to buyer skepticism. CampusTech Connect addresses this by requiring university email verification for all users and focusing on on-campus transactions, which creates a built-in layer of trust and accountability. Students can feel safer knowing they are dealing with fellow students rather than anonymous sellers. Moreover, features like the price comparison tool and the inclusion of only campus-related users give CampusTech Connect an academic relevance not found on Facebook Marketplace or similar platforms.

In essence, while sites like Craigslist or Facebook Marketplace cast a wide net for all kinds of items and users, CampusTech Connect reimagines how students buy and sell tech on campus with a targeted approach, providing a niche, trusted service that aligns with university life.

3.0 Summary of Requirements

The development of CampusTech Connect was guided by a clear set of functional and non-functional requirements. These requirements were identified through a structured, iterative, and user-centered approach aimed at designing a solution that directly reflects the needs of university students.

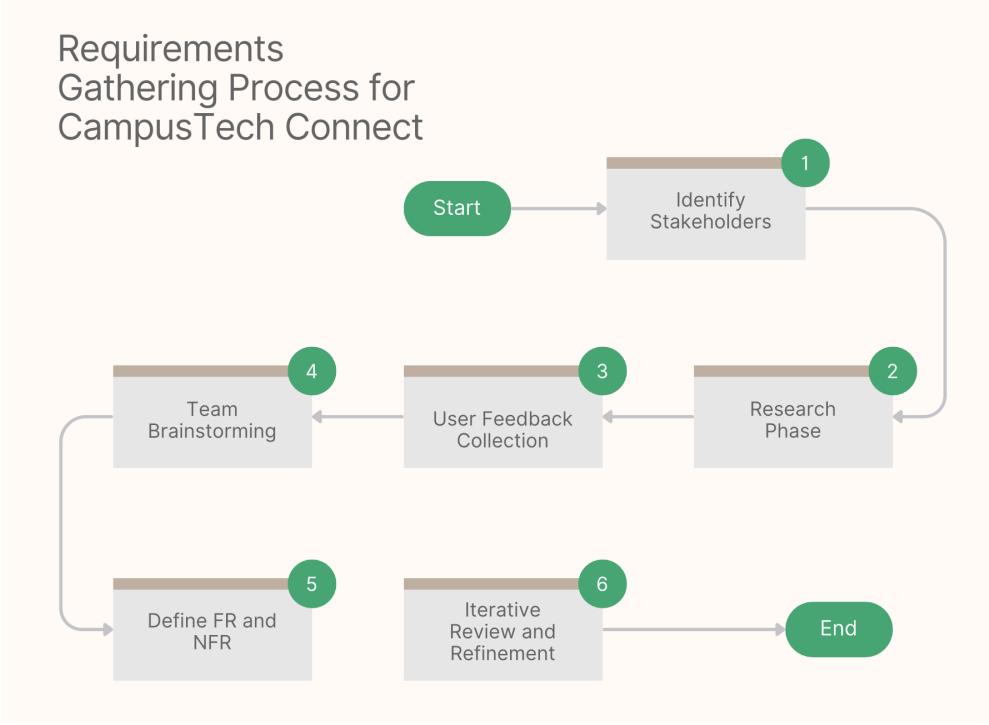

Figure 2. Graphical View of Requirements Gathering Process (Refer to Appendix A for Details).

These requirements were derived using a user-centered requirements gathering process (see Figure 2 and Appendix A) tailored to university students. The team began by researching student technology pain points and observing how students currently buy/sell used devices. This included reviewing survey data about student laptop challenges. For example, the high rates of device malfunctions and software compatibility issues students reported in Figure 1. This also involved informally interviewing fellow students about their experiences with platforms like Kijiji (see Appendix B).

Common requirements elicitation techniques such as interviews, surveys, and team brainstorming were applied to capture diverse user needs [3]. For instance, the team conducted a short survey of students across different faculties to identify what features would make them trust a campus marketplace (see Appendix C). The team also held a brainstorming session within the team (as all members are part of the target user demographic) to pool insights and prioritize key features (see Appendix D).

Based on this thorough requirements gathering process, the team clearly distinguished between what the system must do (functional requirements) and how the system must behave (non-functional requirements). This approach ensured that the final system architecture, user experience, and platform features were aligned with actual student expectations and technical feasibility.

3.1 Functional requirements:

The functional requirements describe the core features and behaviors of the CampusTech Connect system. In other words, the services it provides to its users in order to facilitate a trusted, student-focused marketplace.

	Feature	User Need Addressed
0	University Email Login	Trust in peer-to-peer transactions (Interview 1, 3)
1	Spec-based Filtering	Ease of finding suitable devices (Survey Q1, Interview 2)
2	Recommendation Engine	Support for non-tech-savvy users (Interview 2, Survey Q2)
3	Wishlist & Target Price	Budget planning & deal tracking (Brainstorming, Survey Q1)
4	Comparison Tool	Smarter decision-making, price awareness (Survey Q2)

Figure 3. Mapping of Key System Features to Identified User Needs, Based on Interviews, Surveys, and Brainstorming Sessions.

Key functional requirements include:

A. User Account Management:

The system needs to allow users to register for an account using their university email and a password, and subsequently log in with those credentials. Email verification (e.g., confirming the address ends in an approved university domain) is required to ensure only legitimate students join the platform. Users should also be able to choose a role (Buyer or Seller) upon registration (though a single user can perform both roles).

B. Secure Authentication:

Upon successful registration, the user needs to be able to log in and receive a secure session (the team implemented JWT-based authentication for session management). This ensures that pages and actions are protected and only accessible to authenticated users.

C. Laptop Listing Management (for Sellers):

Sellers need to be able to create new laptop listings by inputting details such as the laptop's brand, model, specifications (CPU, RAM, storage, etc.), condition, and asking price. The system

will validate the input (for example, required fields cannot be empty) and store the listing in the database. Sellers should also be able to view a dashboard of all their listings, edit/update a listing's details if needed (e.g. adjust the price or add information), and delete a listing when it's no longer available. Each listing created by a seller is tagged as "pending verification" or similar until it meets the trust criteria (for instance, the team design envisions an admin or automated check to mark certain listings as verified to avoid fraudulent posts).

D. Listing Search and Filters (for Buyers):

Buyers need to be able to browse available listings. The system should provide search functionality (by keyword or model) and filtering options, for example, filtering by price range, by hardware specifications (min RAM, specific GPU, etc.), or by other criteria relevant to students (perhaps filtering by campus or pickup location). This makes it easy for a buyer to find laptops that meet their needs and budget without sifting through irrelevant offers.

E. Laptop Recommendation Engine:

The application should offer personalized recommendations to buyers based on their academic profile or stated preferences. For instance, during signup or in their profile, a user might specify their major or the primary intended use of the laptop (e.g. gaming, graphic design, programming). The system can then suggest listings that match those needs (or even suggest not-yet-listed popular models that fit their criteria to encourage sellers to post such items). This feature aims to guide less tech-savvy students toward appropriate choices (e.g., a design student would be pointed to laptops with better GPUs and high-res screens, while a business major might get recommendations for portability and battery life).

F. Wishlist and Target Price Tracking:

Buyers need to be able to maintain a wish list of laptops. This means a buyer can add a listing to their wishlist (or even add a specific laptop model that they are interested in, whether or not it's currently listed). For each wishlist item, the user can set a target price they'd be willing to pay. The system stores this information and could notify the buyer if a listing's price drops to their target or if a new listing at or below that price appears. Buyers should be able to view all laptops in their wishlist, update the target price for each, and remove items from the wishlist when they are no longer interested. This functionality helps students keep track of potential deals and plan purchases within their budget.

G. Laptop Comparison Tool:

The platform needs to allow users to compare multiple laptops side-by-side. A buyer should be able to select two or more listings and trigger a comparison view that displays their specifications and prices in a table or comparative format. Additionally, the requirement was to integrate external pricing data, for example, fetching the current average price of the same model from an external source (such as eBay or a retail site) to give context on how fair each seller's price is. This comparison tool is intended to help students make smarter purchasing decisions by easily visualizing which laptop gives better value or suits their needs best.

H. Transaction and Communication Support: (Planned scope)

While the initial implementation focuses on listings and discovery, a functional requirement the team noted for completeness is to support the transaction process. This includes allowing interested buyers to contact sellers (perhaps via an internal messaging system or by providing contact info) to arrange the sale. Although full e-commerce features (like online payments) were out of scope for the project, the system should at least facilitate communication between buyer and seller in a safe manner (e.g., through in-app messaging using university emails to keep personal contact info private).

These functional requirements ensure the system provides all necessary capabilities for a student-oriented marketplace. Each requirement was traced back to a user need identified during the requirements gathering phase (for example, the trust issues observed led directly to the email verification requirement, and the difficulty in comparing specs led to the comparison tool requirement).

3.2 Non-functional requirements

In addition to what the system does, the team defines how the system should behave and the qualities it must uphold. Non-functional requirements (NFRs) cover the performance, security, usability, and other quality attributes of CampusTech Connect.

Key non-functional requirements include:

A. Security and Privacy:

The application must be secure since it involves user accounts and personal data. This entails using secure communication (HTTPS for all data transfer), storing passwords securely (hashed and salted in the database), and preventing unauthorized access to user data. Only authenticated users can perform actions like creating listings or viewing contact information. The team also ensures that only users with verified university emails gain access, adding a layer of security by limiting the user base. Privacy is considered by storing minimal personal information (the team

primarily uses email and names, avoiding sensitive data) and never exposing emails publicly without consent.

B. Reliability and Data Integrity:

The system must operate reliably without crashes or data loss. For example, when a listing is created or a wishlist entry is added, the data must be correctly saved and remain consistent even if multiple users are using the system at once. The system requires that database transactions are handled properly to avoid corruption. Uptime should be high (as a campus service, ideally available 24/7) so that users can access the marketplace whenever needed. In case of any server failure, proper backup and recovery mechanisms should be in place (the team regularly backed up the database during development to protect data).

C. Usability and Accessibility:

Since the target users are busy students, the interface must be intuitive and easy to navigate without requiring a technical learning curve. This means clear navigation menus, logical organization of features, and instructions or tool-tips where necessary (for instance, explaining how to add to a wishlist or compare items). The team aimed for a clean, responsive UI design so that even those accessing from a phone or tablet can use it (more on mobile support in the next section). Accessibility considerations include readable font sizes, sufficient color contrast, and avoiding reliance on any one input method (e.g., all functions should be accessible by mouse or touch, not keyboard only). A student should be able to register, find a laptop, and contact a seller in just a few straightforward steps.

D. Performance:

CampusTech Connect should offer reasonably fast response times. As a non-functional goal, searches and page loads should happen within a couple of seconds on a typical campus internet connection. Even with dozens of listings, the system should be able to filter and display results quickly. The team needed to optimize database queries (for example, using indexes on key fields like price or model for faster search) to meet this requirement. Performance also ties into scalability: as more students join and more listings are added over time (or if expanding to multiple campuses), the system should still perform well. The team considered using pagination or infinite scroll for listings to ensure the interface remains snappy and doesn't overload the client by rendering too much data at once.

E. Compatibility and Maintainability:

The web application must be compatible with common browsers (Chrome, Firefox, Safari, etc.) and not require any special software on the client side aside from a browser. This is to accommodate all students, whether they use Windows, Mac, or mobile devices. Maintainability is also important; as a non-functional requirement, the code and documentation was structured so that future developers can update the system. This includes clear modular separation (for instance, separating frontend and backend logic, using descriptive naming and comments in code) and using standard frameworks so new team members can quickly understand the stack. This will help when adding new features or scaling the system to other universities.

All these requirements were documented and reviewed to ensure they aligned with the project goals. By following a valid requirements gathering process (stakeholder input, research, and iteration) and distinguishing functional vs. non-functional needs, the team established a solid foundation for the system's design. This careful planning is reflected in the implementation of CampusTech Connect, which meets the identified requirements within the scope of our project.

4.0 Use Cases

The outlined use cases define the core functionalities of the system, ensuring a smooth and efficient experience for buyers and sellers on the CampusTech Connect platform. These use cases represent essential user interactions and are selected based on importance, frequency of use, and impact on user experience.

4.1 User Registration and Login

User authentication is foundational for system access and security. The **User Registration** use case **(Use Case ID: 1)** allows new users to create an account by submitting their email, password, and role (Buyer/Seller). The system ensures uniqueness by checking for duplicate emails and confirming eligibility through domain-based email verification.

The **User Login** use case **(Use Case ID: 2)** enables existing users to access the platform securely. Login is protected using JWT-based session tokens to ensure only authenticated users can interact with the platform's protected functionalities.

These use cases are described in detail below:

Table 1. Use Case ID 1 - User Registration

Use Case Name: User Registration	**ID:** 1	**Importance Level:** High
Primary Actor: User (Buyer/Seller)		
Brief Description: This use case allows a new user to register for an account.		

Trigger: A user submits a registration request to access the system.
Type: External

Normal Flow of Events:	**Information for Steps:**
1. User submits a registration request.	→ User email request
2. User provides a system with email.	← User email
3. System validates user email.	← Existing users information
4. System requests role information from the user.	→ User role request
5. User provides the system with a role.	← User role
6. System requests password from user.	→ User password request
7. User provides the system with a password.	← User password → Registration success confirmation
8. System creates and stores the user as a buyer or a seller.	

Alternative/Exceptional Flows	**Information for Steps:**
3. Invalid email or user already exists a. System provides email error message to user	← User email → Email error message
5. Invalid role a. System provides role error message to user	← User role → Role error message

Summary of Inputs

Description	**Source**
User email	User
Existing users information	User Repository
User role	User
User password	User

Summary of Outputs

Description	**Destination**

User email request	User
Email error message	User
User role request	User
Role error message	User
User password request	User
Registration success confirmation	User

Table 2. Use Case ID 2 - User Login

Use Case Name: User Login	**ID:** 1	**Importance Level:** High
Primary Actor: User (Buyer/Seller)		
Brief Description: This use case allows an existing user to login to the system.		
Trigger: A user submits a login request to access the system. **Type:** External		
Normal Flow of Events: 1. User submits a login request. 2. User provides the system with email. 3. System validates user email. 4. System requests password from user. 5. User provides the system with a password. 6. System generates and returns a JWT token for the session.	**Information for Steps:** → User email request ← User email ← Existing users information → User password request ← User password → JWT Token → Login success confirmation	
Alternative/Exceptional Flows 3. Invalid email or user already exists b. System provides email error message to user 5. Invalid password b. System provides password error message to user	**Information for Steps:** ← User email → Email error message ← User password → Password error message	

Summary of Inputs	
Description User email Existing users information User password	**Source** User User Repository User
Summary of Outputs	
Description User email request Email error message User password request Password error message JWT Token Login success confirmation	**Destination** User User User User User User

4.2 Laptop Listing Management for Sellers

Sellers play a vital role in the marketplace by adding and managing laptop listings. The **Seller Laptop Listing Management** use case **(Use Case ID: 3)** captures this functionality. Sellers can create new listings, view their existing ones, update listing details, and delete entries that are no longer valid. The system ensures that only the listing owner can perform modifications or deletions, preserving data integrity and ownership accountability. This use case is described in further detail in Table 3.

Table 3. Use Case ID 3 - Seller Laptop Listing Management

Use Case Name: Seller Laptop Listing	**ID:** 3	**Importance Level:** High
Primary Actor: Seller		
Brief Description: This use case allows a seller who is logged in to create, view, update, verify, and delete laptop listings on the platform.		
Trigger: A seller sends a request to add/modify/delete a laptop listing. **Type:** External		
Normal Flow of Events: 1. System retrieves seller identity from JWT token. 2. System gets desired action from seller - create laptop listing, view laptop listings, or delete laptop listings	**Information for Steps:** ← JWT Token	

a. If create laptop listing, perform subflow S-1 b. If view laptop listings, perform subflow S-2 c. If delete laptop listing, perform subflow S-3	
SubFlows: S-1: Create new laptop listing 1. System creates a laptop listing. 2. System requests laptop specifications. 3. Seller provides laptop specifications. 4. System saves laptop listing information. S-2: View seller's listed laptops 1. System provides all of the seller's laptop listings. S-3: Seller deletes laptop listing 1. Seller provides ID of laptop to be deleted. 2. System verifies that the laptop is currently owned by the seller. 3. Remove laptop listing and provide confirmation to the seller.	 → Seller laptop specifications request ← Laptop specifications → New laptop listing ← All existing laptop listings → All of seller's laptops ← Laptop ID ← All existing laptop listings → Laptop removal confirmation
Alternative/Exceptional Flows 1. Seller not found a. System provides not found error message to user S-1, 3. Invalid laptop specification detail a. System provides error message to user and requests information again S-3, 2. Laptop not owned by seller	**Information for Steps:** ← JWT Token → Seller not found error message ← Laptop specifications → Laptop specifications error message ← All existing laptop listings

a. System provides non-ownership error message to user	→ Non-ownership error message
Summary of Inputs	
Description JWT Token Laptop specifications All existing laptop listings Laptop ID	**Source** Seller Seller Laptop Repository Seller
Summary of Outputs	
Description Seller not found error message Seller laptop specifications request Laptop specifications error message New laptop listing All of seller's laptop Non-ownership error message Laptop removal confirmation	**Destination** Seller Seller Seller Laptop Repository Seller Seller Seller

4.3 Laptop Comparison Tool

To help buyers make informed purchasing decisions, the system includes a **Compare Laptops** use case **(Use Case ID: 4)**. This feature allows users to compare selected listings side-by-side, along with real-time price comparisons from external sources such as eBay. It supports academic decision-making by evaluating technical specs relevant to student needs. The nuances of this use case is described in Table 4.

Table 4. Use Case ID 4 - Compare Laptops

Use Case Name: Compare Laptops	**ID:** 4	**Importance Level:** Medium
Primary Actor: Buyer		
Brief Description: This use case allows a user to compare multiple laptops based on all available attributes (price, processor, RAM, storage, etc.) using real-time eBay listings to make an informed purchase/sale decision.		
Trigger: User submits a request to compare laptops. **Type:** External		
Normal Flow of Events:	**Information for Steps:**	

1. User selects a laptop for comparison. 2. System verifies if a laptop exists. 3. System requests comparison criteria. 4. User provides a system with comparison criteria. 5. System finds similar laptops based on eBay listings. 6. Provide the user with the list of similar laptops.	← Laptop ID ← All existing laptop listings → Comparison criteria request ← Comparison criteria → Comparison criteria ← Similar laptops list → Similar laptops list
Alternative/Exceptional Flows 2. Laptop not found a. System provides not found error message to user 4. No similar laptops found a. System provides no similar laptops exist message to user	**Information for Steps:** ← Laptop ID → Laptop not found error message ← Laptop ID → No similar laptops exist error message

Summary of Inputs	
Description	**Source**
Laptop ID	User
All existing laptop listings	Laptop Repository
Comparison criteria	User
Similar laptops list	EBay API

Summary of Outputs	
Description	**Destination**
Laptop not found error message	User
Comparison criteria request	User
Comparison criteria	EBay API
No similar laptops exist error message	User
Similar laptops list	User

4.4 Wishlist and Target Price Management for Buyers

The **Buyer Wishlist Management** use case **(Use Case ID: 5)** provides a personalized space for users to track potential purchases. Buyers can add laptops to their wishlist, define a target price, view saved items, update preferences, and remove entries. The system validates ownership and prevents duplication to maintain wishlist integrity. This use case is described in further detail in Table 5.

Table 5. Use Case ID 5 - Buyer Wish List Management

<table>
<tr><td colspan="2">Use Case Name: Buyer Wish List Addition</td><td>ID: 5</td><td>Importance Level: High</td></tr>
<tr><td colspan="4">Primary Actor: Buyer</td></tr>
<tr><td colspan="4">Brief Description: This use case allows a buyer who is logged in to add, view, update, and delete laptops to their wishlist.</td></tr>
<tr><td colspan="4">Trigger: A buyer sends a request to add/view/modify/delete their wishlist.
Type: External</td></tr>
<tr><td colspan="2">Normal Flow of Events:

1. System retrieves buyer identity from JWT token.

2. System gets desired action from buyer - add laptop to wishlist, view wishlist, delete laptop from wishlist, or update laptop in wishlist
a. If add laptop to wishlist, perform subflow S-1
b. If view wishlist, perform subflow S-2
c. If delete laptop from wishlist, perform subflow S-3
d. If update laptop in wishlist, perform subflow S-4</td><td colspan="2">Information for Steps:

← JWT Token</td></tr>
<tr><td colspan="2">SubFlows:

S-1: Add new laptop to wishlist.
1. System requests laptop ID for laptop to be added to wishlist.

2. Buyer provides a laptop ID.</td><td colspan="2">

→ Laptop ID request

← Laptop ID</td></tr>
</table>

3. System verifies that laptop is not already in wishlist	
4. System requests target price.	→ Target price request
5. Buyer provides target price for laptop.	← Target price
6. System creates and stores the laptop to the wishlist.	→ New wishlist item
S-2: View buyer's wishlist 1. System provides a buyer's wishlist to the buyer.	← All existing wishlists → Buyer's wishlist
S-3: Seller deletes laptop in wishlist 1. Seller provides ID of laptop to be deleted.	← Laptop ID
2. System verifies that the laptop is currently in buyer's wishlist	← All existing wishlists
3. Remove laptop from wishlist and provide confirmation to the seller.	→ Laptop removal confirmation
S-4: Seller updates laptop in wishlist 1. Seller provides ID of laptop to be deleted.	← Laptop ID
2. System verifies that the laptop is currently in buyer's wishlist	← All existing wishlists
3. System requests target price.	→ Target price request
4. Buyer provides target price for laptop.	← Target price
5. System updates the laptop to wishlist.	→ Updated wishlist item
Alternative/Exceptional Flows 1. Buyer not found a. System provides not found error message to user	**Information for Steps:** ← JWT Token → Buyer not found error message

S-1, 2. Laptop does not exist a. System provides laptop not found error message to buyer S-1, 4. Target price invalid a. System provides target price invalid error message to buyer S-3/4, 2. Laptop not found in buyer's wishlist a. System provides laptop not found error message to buyer	← Laptop ID → Laptop does not exist error message ← Target price → Target price invalid error message ← Laptop ID → Laptop not found error message

Summary of Inputs	
Description	**Source**
JWT Token	Buyer
Laptop ID	Buyer
Target price	Buyer
All existing wishlists	Wishlist Repository

Summary of Outputs	
Description	**Destination**
Buyer not found error message	Buyer
Laptop ID request	Buyer
Target price request	Buyer
New wishlist item	Wishlist Repository
Buyer's wishlist	Buyer
Laptop removal confirmation	Buyer
Updated wishlist item	Wishlist Repository

Although user registration, login, seller listing, and buyer wishlist management are the most critical and frequently used features, the addition of a comparison tool enhances buyer confidence and decision-making. These five use cases collectively provide a balanced and secure marketplace experience, fulfilling both buyer and seller needs while ensuring robust session control, data protection, and academic relevance.

5.0 Data Flow Diagrams

The use cases presented in Section 4 thoroughly describe the primary processes within the Laptop Marketplace System. These include step-by-step flows, the information involved, its sources and destinations, and various alternative or exceptional scenarios. However, while textual descriptions are detailed, visual representations like data flow diagrams (DFDs) offer an intuitive understanding of system structure and behavior. To that end, three DFDs have been

created: a Context Diagram to provide an overall system overview, a Level 0 DFD to detail interactions with the database, and a Level 1 DFD focused specifically on the Compare Laptops process (Process 4). The context diagram is presented in Figure 4 below.

5.1 Context Diagram

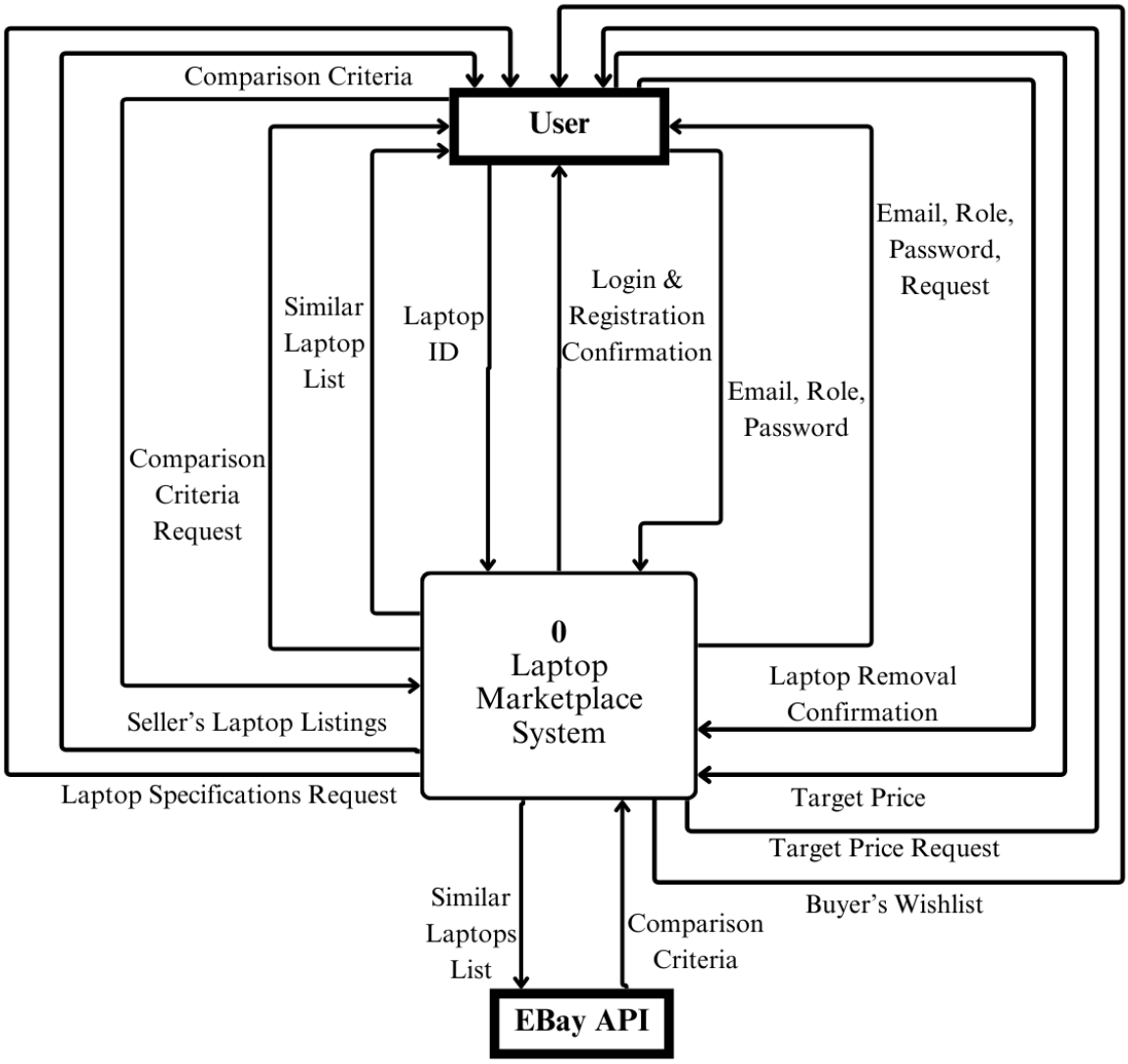

Figure 4. Context Diagram for Laptop Marketplace System.

The context diagram (Figure 4) shows the high-level architecture of the Laptop Marketplace System and its external entities. There are two major entities interacting with the system: the User and the eBay API. The User entity is responsible for initiating nearly all system interactions, and while represented as a single block in the context diagram, it encompasses both Buyers and Sellers. The system requests information from the User in many instances, such as during account registration, login, listing creation, wishlist management, or when a laptop is

selected for comparison. In turn, the system provides the User with confirmations or desired data, such as a list of their laptop listings, their wishlist items, and comparable devices. For every completed operation, the system also sends confirmation messages, like successful registration, login success, item deletion, or wishlist update.

The system also communicates with the eBay API. This occurs during the laptop comparison process, where the system sends specific comparison criteria (e.g., price range, processor type, RAM) to the API and receives a list of similar laptops. This integration provides users with additional context when evaluating a purchase or sale. Since the User entity encapsulates both buyers and sellers, this distinction is unpacked further in the Level 0 DFD (Figure 5) for clearer process mapping.

5.2 Level 0 DFD

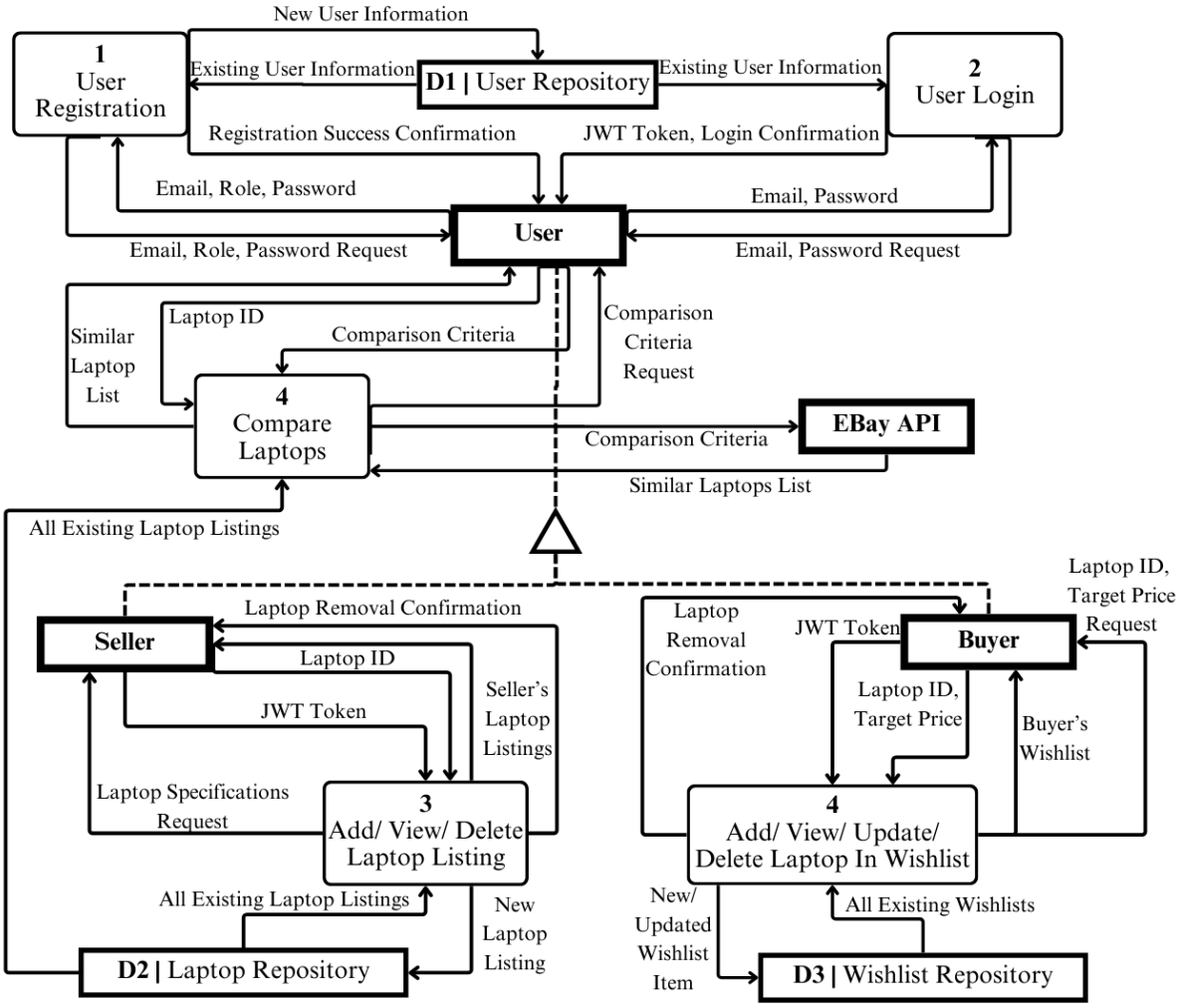

Figure 5. Level 0 DFD for the Laptop Marketplace System.

The Level 0 DFD (Figure 5) builds upon the context diagram by breaking down the system into individual processes and revealing internal data interactions. It introduces three core databases: Database 1 (User Repository), Database 2 (Laptop Repository), and Database 3 (Wishlist Repository). It also explicitly separates Buyer and Seller roles under the User entity to demonstrate how their actions differ in terms of system flow and data operations.

In this diagram, Process 1 (User Registration) and Process 2 (User Login) are shown interacting with Database 1. These processes involve retrieving existing user information to check for duplicates or validate credentials, as well as storing new user information upon successful registration. Process 3 (Add/View/Delete Laptop Listing) and Process 4 (Compare Laptops) both interact with Database 2. Specifically, these processes retrieve existing laptop listings from the Laptop Repository. Process 3 also writes data to the repository when a seller creates or updates a listing. These interactions allow the system to verify listing ownership or validate entries.

Additionally, Process 5 (Add/View/Update/Delete Wishlist Item) is linked to Database 3 – the Wishlist Repository. This process involves reading from the repository to retrieve a user's wishlist and writing to it when a new item is added, updated, or removed. This allows the system to dynamically present and modify each user's wishlist based on their input and preferences.

Among all the processes, Process 4 (Compare Laptops) is unique because it interacts with more than one external entity. In addition to drawing data from the User and Laptop Repository, it also interfaces with the eBay API to enhance comparison functionality. This multi-entity interaction justifies a deeper analysis, which is addressed using a Level 1 DFD shown in Figure 6.

5.3 Level 1 DFD for Process 4

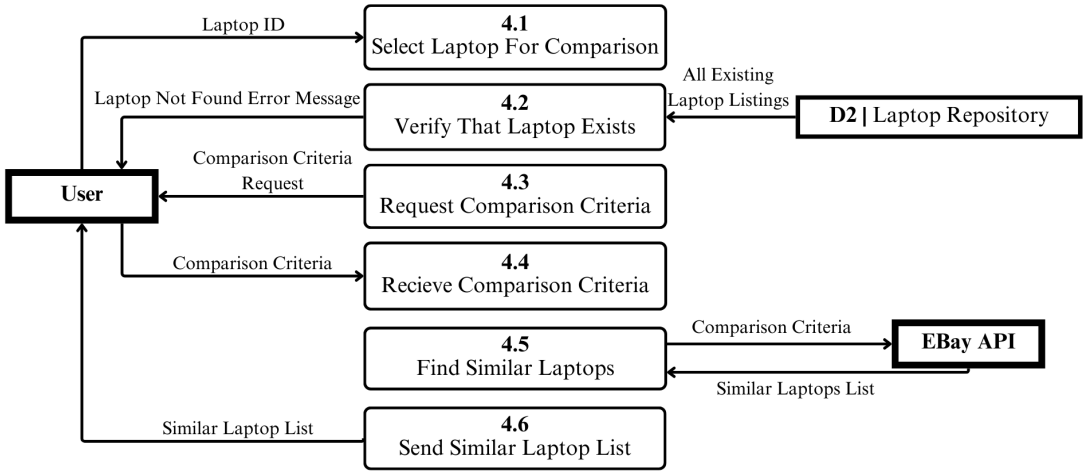

Figure 6. Level 1 DFD for Laptop Comparison Tool (Process 4).

The Level 1 DFD for Process 4 (Figure 6) illustrates the detailed workflow of the comparison feature. It begins when a user selects a laptop by providing its unique ID. The system uses this ID to verify the laptop's existence and attributes using the Laptop Repository. Next, the system requests comparison criteria from the user, such as budget, RAM, processor, or brand preference. Once this data is collected, the system sends the criteria to the eBay API and receives a list of similar laptops. These results are then presented to the user for further evaluation. This diagram helps clarify how the system integrates internal database validation with external API responses to create a cohesive feature.

In summary, the three DFDs (context, Level 0, and Level 1) collectively demonstrate how user interactions, data repositories, and external services are orchestrated within CampusTech Connect. They reinforce the data-driven nature of the system while highlighting the unique flows tied to each role and function.

6.0 UML Class Diagram

The UML Class Diagram for CampusTech Connect provides a structural blueprint of the platform's object-oriented design, representing key entities, their attributes, methods, and relationships. The class diagram focuses on encapsulating the core roles of buyers and sellers, the management of laptop listings, and the personalized features such as wishlists and recommendations.

The complete UML Class Diagram is presented in Figure 7, illustrating inheritance, aggregation, and association relationships among the system's classes. It captures the fundamental architecture of the CampusTech Connect platform and supports the implementation of its core features.

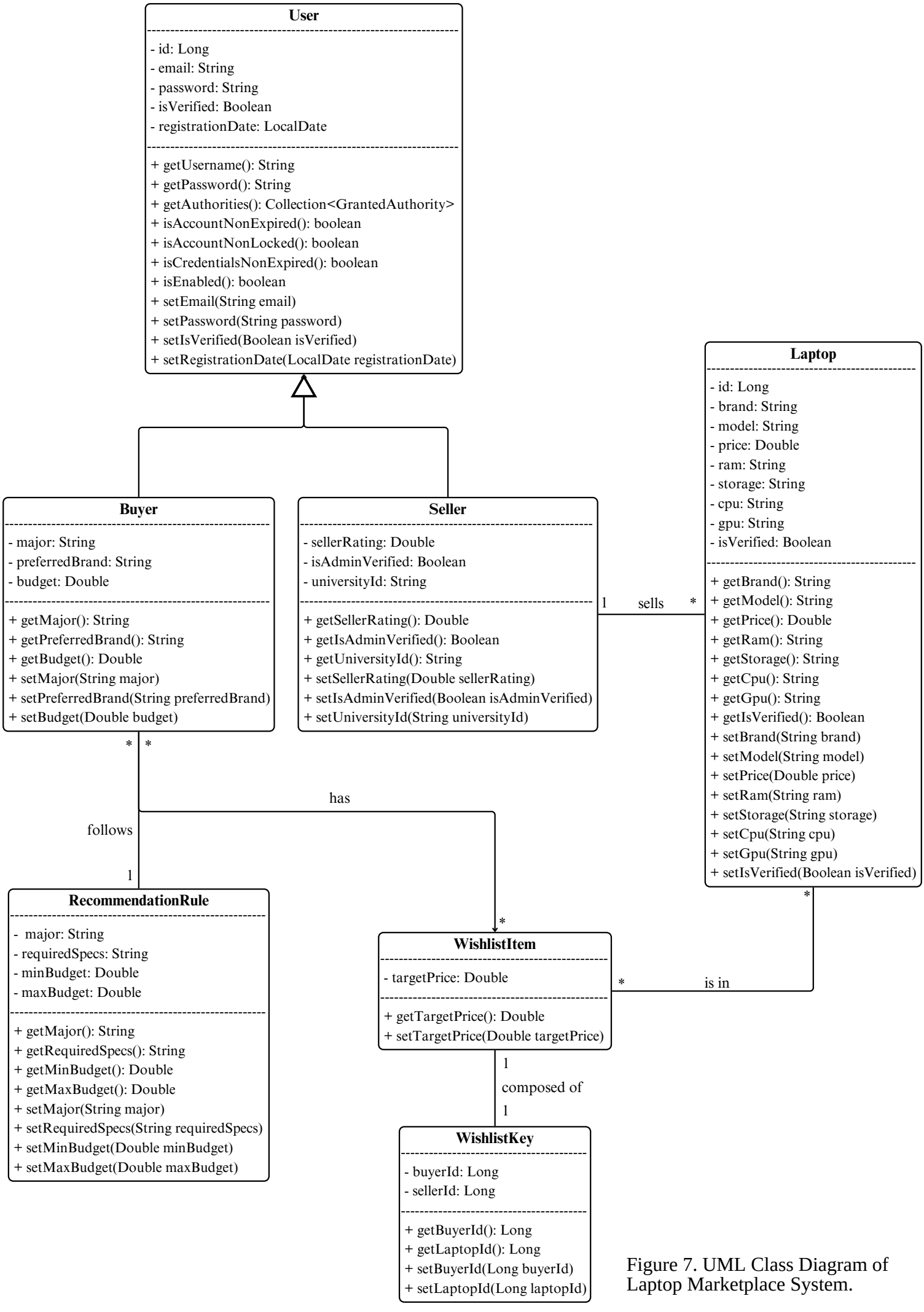

Figure 7. UML Class Diagram of Laptop Marketplace System.

Figure 7 shows that the foundation of the system is the User class, which serves as a generalization for both Buyer and Seller subclasses. All shared attributes and methods related to system access, such as email, password, and authentication mechanisms, are defined in the User class and inherited by both the Buyer and Seller classes. This abstraction ensures that both user types share a common identity structure while allowing for role-specific behaviors and associations.

Sellers are associated with Laptop objects, which they initially own and can offer for sale. Each laptop listing includes specifications such as CPU, RAM, storage, and condition. Sellers are able to create, update, and delete their listings, and the system verifies these actions through ownership validation.

Buyers interact with the WishlistItem class, which represents laptops that users have saved for future consideration. The WishlistItem class uses a composite key, implemented as WishlistKey, combining both the buyer's ID and the laptop ID to ensure uniqueness and traceability. This allows buyers to track target prices and manage multiple wishlist entries efficiently.

In addition to wishlist functionality, buyers also benefit from a RecommendationRule class. This component provides intelligent laptop suggestions tailored to the buyer's academic program or major. By associating specific recommendation rules with buyers, the system enhances user experience through relevant suggestions that align with each student's computing needs, whether for programming, design, or general use.

7.0 UML State Chart

Statechart diagrams are used to represent the dynamic behavior of entities by illustrating how they transition between various states over time. In the context of the CampusTech Connect platform, the Laptop entity stands out as the most dynamic and complex. Its life cycle is influenced by the actions of both buyers and sellers, as well as automated system behaviors. To capture this complexity, a UML Statechart Diagram (Figure 8) is used to describe how a laptop moves through different states during its existence on the platform.

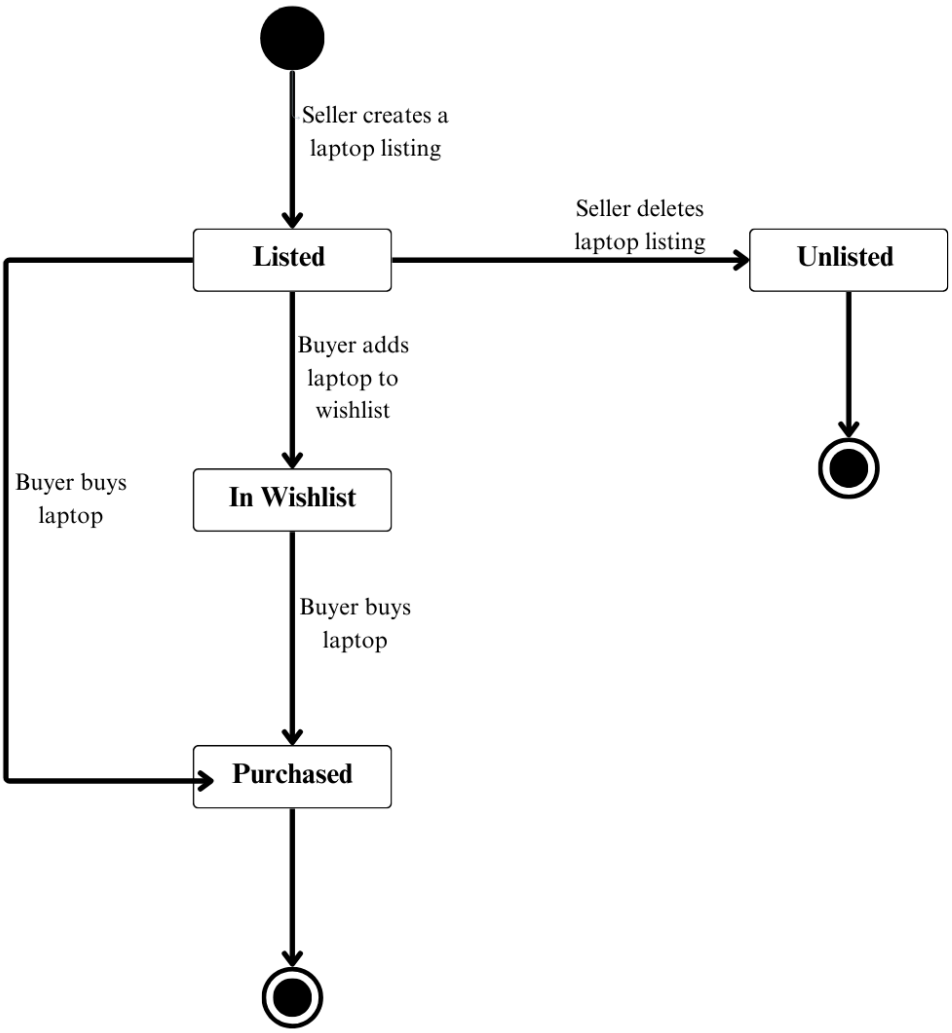

Figure 8. UML Statechart Diagram of Laptop Marketplace System.

As shown in Figure 8, the life cycle of a laptop begins when a seller creates a listing, which transitions the laptop into the Listed state. This indicates that the laptop is now available on the marketplace and visible to potential buyers. At any point, the seller has the option to delete the listing, which causes the laptop to transition into the Unlisted state. This state signifies that the laptop is no longer available to buyers and has been removed from active listings.

Meanwhile, buyers can interact with listed laptops by adding them to their wishlist. This action transitions the laptop to the In Wishlist state, indicating interest but not yet a purchase. This state supports buyers in tracking their preferred devices and planning purchases, often in conjunction with a target price. When a buyer chooses to proceed with a purchase, the laptop transitions to the Purchased state, representing the completion of the transaction and the end of the laptop's active listing cycle.

The transitions between these states provide a clear and structured workflow, enabling smooth interactions and effective inventory management. By modeling the laptop's behavior in this way, the system ensures that both buyers and sellers experience a predictable and organized interface, ultimately improving user satisfaction and platform reliability.

8.0 UML Sequence Diagram

To further elaborate on how users interact with the laptop comparison feature, particularly through real-time integration with eBay listings (as described in **Use Case ID: 4 – Compare Laptops**), a UML Sequence Diagram is used. The diagram visually represents the flow of interactions between system components, including how the system processes valid and invalid inputs and how it connects to internal and external data sources. It serves as a detailed complement to the use case narrative by showing sequential message exchanges between the user, system components, and the eBay API.

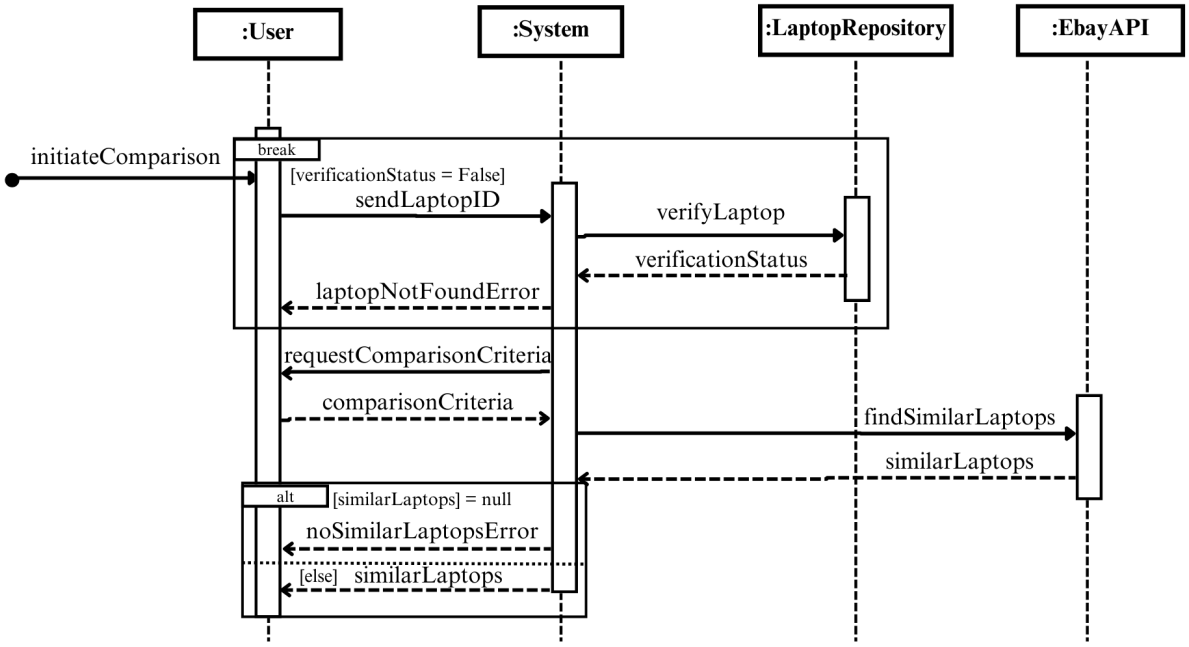

Figure 9. UML Sequence Diagram describing Use Case 4: Compare Laptops.

As illustrated in Figure 9, the comparison process begins when the user selects a laptop they wish to use as the reference point for comparing others. To do this, the user provides a LaptopID, which is passed to the system. The system's first task is to verify that the selected laptop exists in the internal database. If the provided LaptopID does not correspond to any listing in the system, the system prompts the user to provide a different LaptopID. This alternate scenario is depicted using a break frame in the sequence diagram, which separates the exception handling from the normal workflow.

Once a valid LaptopID has been confirmed, the system requests comparison criteria from the user. These criteria may include attributes such as RAM, processor type, storage, price range, or other specifications relevant to the buyer's needs. The user provides the requested criteria, and the system forwards this data to the eBay API to retrieve a list of similar laptops currently available in the broader market.

After the system receives the response from the eBay API, it must determine whether the list of similar laptops is populated or empty. This decision-making process is represented in the diagram using an alt frame, which shows two alternative flows. If similar laptops are found, the system returns the list to the user. Otherwise, it returns a message indicating that no suitable comparisons were found based on the given criteria.

This sequence diagram reinforces the robustness of the system's design by clearly modeling both standard and edge-case behaviors. It also highlights how the platform leverages external APIs to provide value-added insights, enabling users to make informed purchasing decisions.

9.0 UML Activity Diagram

While individual use cases describe specific system functions, combining them into a unified activity flow provides a broader perspective on user interaction. In the case of CampusTech Connect, a typical Buyer's workflow involves several connected use cases. For instance, a buyer will start by executing Use Case 2 – User Login to access the platform. After logging in, the buyer may perform Use Case 4 – Compare Laptops to evaluate a selected laptop against similar models available on external platforms such as eBay. If the laptop meets the buyer's expectations, they may proceed to Use Case 5 – Buyer Wishlist Management to save the laptop for future consideration. These high-level steps taken by the Buyer and the corresponding system responses can be illustrated using a UML Activity Diagram, as shown in Figure 10.

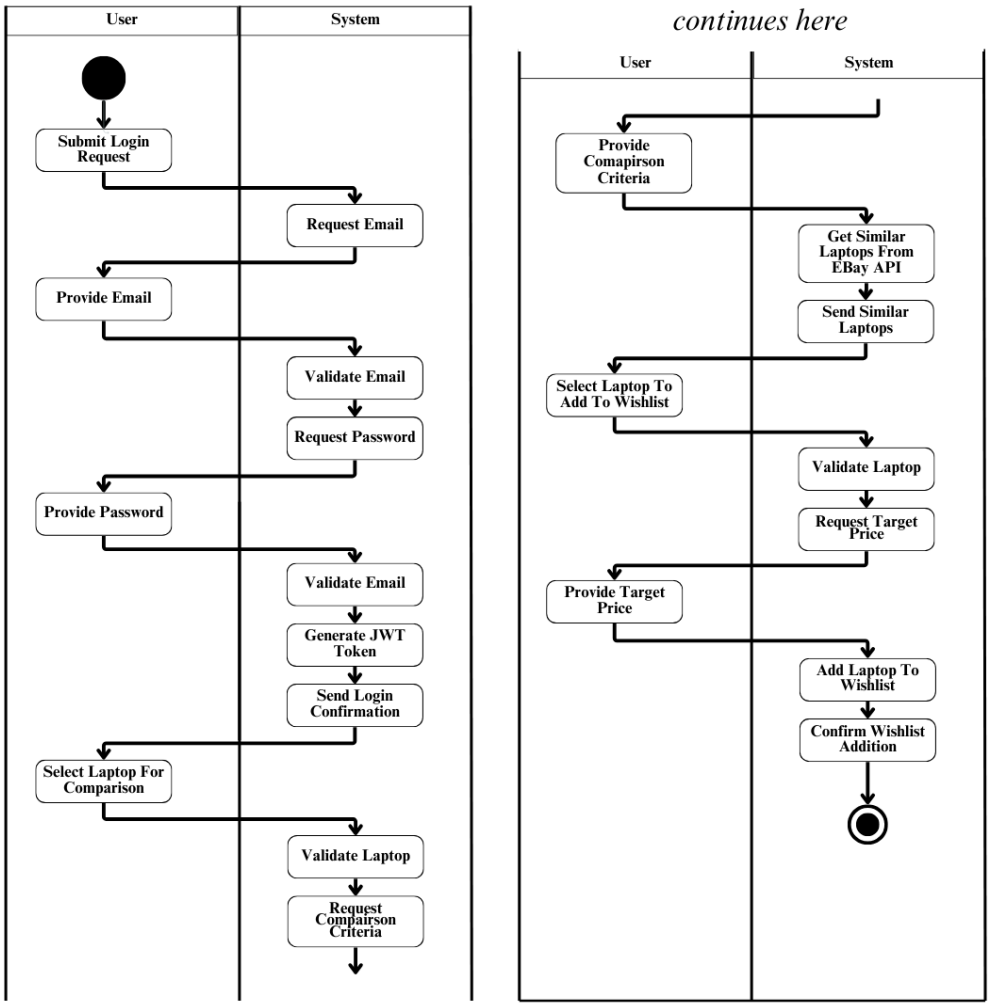

Figure 10. UML Activity Diagram Describing a Typical Buyer's Workflow.

The activity diagram (Figure 10) outlines a streamlined process that integrates authentication, comparison, and wishlist management. The process begins when the user initiates login by submitting their email and password. The system validates these credentials and, upon successful verification, generates a JWT token to authenticate the user session. With authentication complete, the buyer proceeds to compare laptops by selecting a device of interest and supplying specific comparison criteria. The system uses this information to query the eBay API and retrieves a list of similar laptops.

Once the comparison results are presented, the buyer may choose to add the currently viewed laptop to their wishlist. The system first verifies that the selected laptop exists in the platform's database. It then prompts the buyer to enter a target price for the listing. After receiving the input, the system saves the wishlist entry and provides a confirmation message to the buyer, indicating that the action was successful.

This activity flow highlights the seamless integration between multiple use cases and showcases how the system ensures validation, authentication, and feedback at each step. By visually combining these actions, the UML Activity Diagram offers a holistic view of a common buyer interaction and supports the design principle of user-centric functionality.

10.0 Access by Wireless Devices

CampusTech Connect was developed as a modern, browser-based web application designed primarily for accessibility via wireless devices such as laptops, tablets, and smartphones operating over Wi-Fi or cellular networks. At present, the platform is fully functional on laptops and has been tested in that environment. However, support for other wireless devices such as smartphones or tablets remains in its preliminary stages.

Contrary to initial intentions stated in earlier planning phases, the current implementation does not use flexible grid layouts or relative CSS units, which are commonly employed in responsive web design. As a result, although the site loads on mobile browsers, its user interface is not optimized for varying screen sizes, especially those smaller than standard laptop displays. Key components such as the multi-column comparison view, tightly spaced content blocks, and navigation structures are all designed with desktop usage in mind. Without adaptive styling or device-specific layouts, usability on mobile or tablet devices may be compromised.

At this stage, the application is best described as mobile-accessible but not mobile-optimized. The core functionality, such as browsing listings or viewing item details, may still render on mobile browsers, but the layout, touch interaction, and navigation experience have not been tailored or tested for wireless device usability. Elements like hover-based navigation or fixed-width components may hinder user experience on touch-based screens, and performance on cellular networks remains unverified.

Recognizing the growing importance of mobile accessibility in student-oriented applications, enhancing support for wireless devices is a top priority in the next development phase. A structured roadmap is outlined in Figure 11.

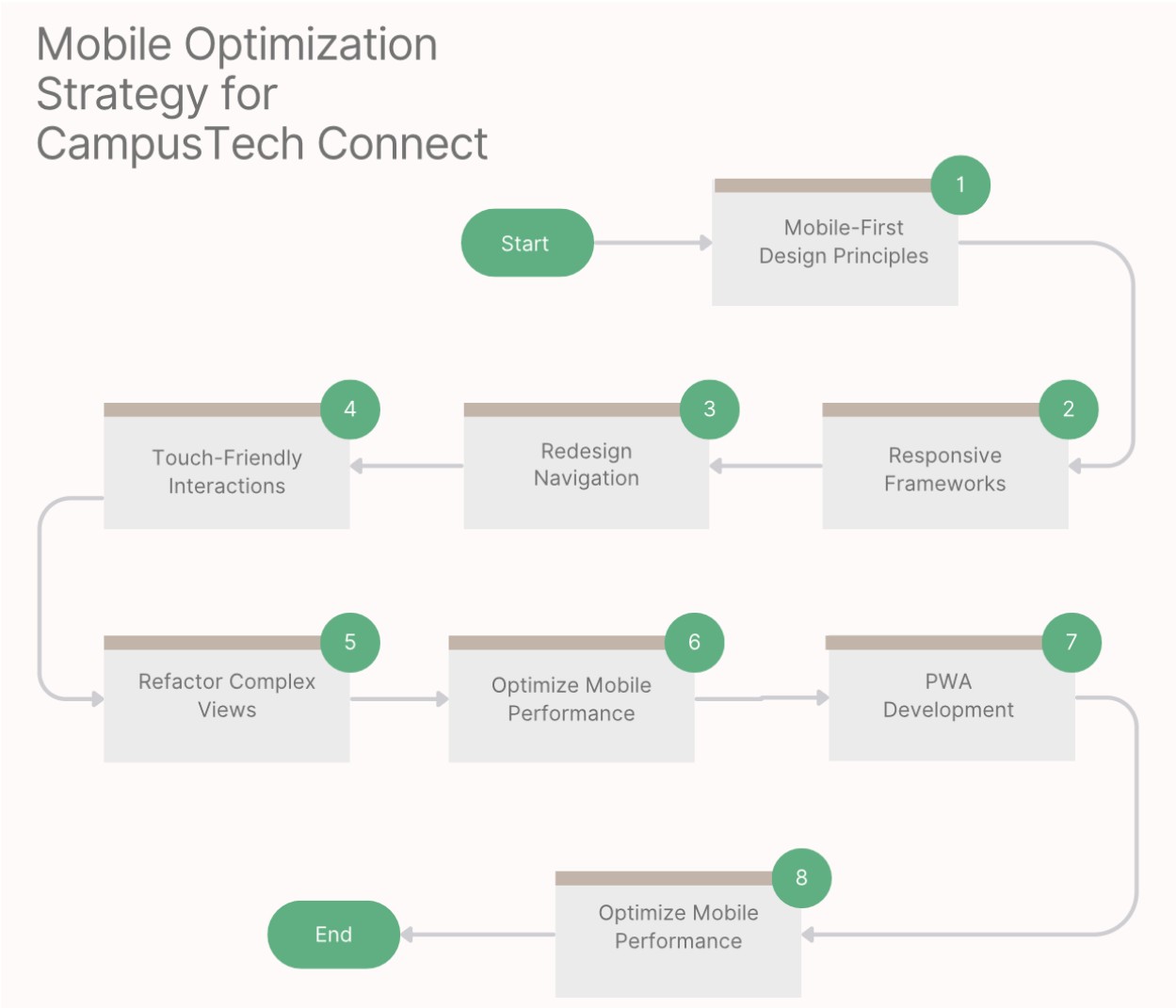

Figure 11. Mobile Optimization Strategy (Refer to Appendix E for Details).

In conclusion, while CampusTech Connect currently provides full support for laptop access, wireless device compatibility is an ongoing area of improvement. Future iterations of the application will follow a structured roadmap toward achieving mobile responsiveness, ensuring all users, regardless of their device, enjoy a seamless and intuitive experience.

11.0 Quality Assurance and Testing Procedures

Quality assurance played a vital role in the development of CampusTech Connect, as it ensured each feature of the application was functionally complete and stable under typical use. Although time constraints limited the depth of the testing, all core functionalities were verified through manual testing, with a partial implementation of automated testing. The team adopted an **iterative testing approach**, verifying each module immediately after development and resolving bugs before moving on to the next task.

For example, once the user registration and login modules were implemented, the team manually tested them by attempting to register with both valid and invalid inputs, such as a non-university email, and confirmed that proper error messages and validation logic were in place. Edge cases, like duplicate registrations or invalid passwords, were tested and handled. Similarly, the listing management feature was tested by creating laptops with and without complete data, editing existing listings, and verifying that only sellers could delete their own listings. These tests were driven by scenario-based checklists derived from the use cases and were executed manually by members of the team.

For automated testing, the team implemented **Spring-based integration tests** using the mockMvc framework. These tests allowed the team to simulate HTTP operations and verify backend logic without running a frontend. Each test invoked endpoints (e.g., /register, /login, /listings) with structured JSON payloads and verified correct HTTP responses, database persistence, and business logic enforcement.

The team developed the following test suites for CampusTech Connect:

- **RegistrationTests.java**
  Tests valid and duplicate student registration, and rejection of non-university emails.
- **LoginTests.java**
  Validates login with correct/incorrect credentials and checks for token generation.
- **ListingTests.java**
  Verifies creation of a new laptop listing, retrieval of all listings, and deletion.
- **WishlistTests.java**
  Confirms adding a laptop to a wishlist, updating its target price, and deleting it.

Each test case is detailed in Appendix F, including setup, inputs, expected outputs, and pass/fail status. All tests passed successfully.

To supplement internal testing, the team also conducted **informal usability testing** (Appendix G). A few students from the UofT campus who were not involved in the development were asked to use the platform and provide feedback. This led to actionable improvements, such as making the “Compare” button more visible and clarifying certain layout elements. This ad-hoc user testing was valuable in validating real-world usage patterns and ensuring the user interface was intuitive.

11.1 Testing Plan and Implementation Status

The team designed a comprehensive testing plan that categorized tests into unit testing, integration testing, interface testing, and acceptance testing.

- **Unit Testing**: The team wrote lightweight utility tests for isolated logic such as price comparison and string validation.
- **Integration Testing**: Implemented using Spring mockMvc, this covered end-to-end user interactions like account creation, login, and CRUD operations on laptops and wishlists.
- **UI Testing**: Conducted manually, focusing on visibility, layout correctness, and flow clarity.
- **Acceptance Testing**: A final system-wide walkthrough based on our use-case scenarios confirmed that all core features met initial requirements.

Each module was assigned to a team member for focused testing. Testing was performed after the development of each feature, with a full walkthrough and checklist validation near project completion. Although the process lacked a formal regression cycle, any new changes were followed by test reruns to ensure nothing broke. Overall, while some advanced aspects (like mobile testing and performance evaluation) were deferred, the implemented tests covered all critical user workflows.

11.2 Further Testing and Quality Assurance Recommendations

Although the application works reliably in a test setting, a fully production-ready system requires additional layers of testing and QA practices. These include:

- **Performance Testing:** Use tools like Apache JMeter or Locust to simulate high traffic and concurrent user interactions. This will help uncover performance bottlenecks in database operations and server response times.
- **Security Testing:** Conduct penetration testing to identify vulnerabilities such as SQL injection, XSS, or unauthorized access to protected routes. Verify proper JWT handling and session expiration policies.
- **Mobile and Cross-Browser Testing:** Manually test on various devices (phones, tablets) and browsers (Chrome, Firefox, Safari, Edge) to ensure consistent performance and appearance across platforms.
- **Formal Usability Testing:** Conduct task-based usability sessions with new users while observing pain points. For example, ask them to “find and compare two laptops” and evaluate their ease in navigating through the workflow.
- **Beta Launch with Feedback Loop:** Release the platform to a small group of students as a beta phase. Collect issue reports and UX feedback over a 2–3 week period. Incorporate findings into design and functionality improvements.
- **Automated Regression Suite:** Build comprehensive test coverage for high-priority paths such as registration, listing management, comparison workflows, and wishlist operations. This suite will serve as a safeguard for future code changes.

CampusTech Connect has undergone sufficient testing to confirm that core features are operational and stable. However, significant opportunities remain to improve system quality through extended testing. Key areas such as regression, performance, security, and mobile optimization are outlined for future implementation. The foundational work completed - manual testing, informal UX feedback, and a minimal automated test suite - sets the stage for scaling up our quality assurance process in future development cycles.

12.0 Challenges/Difficulties Encountered

The development of CampusTech Connect presented several significant challenges, both technical and organizational. As a team of relatively novice developers, integrating multiple components of the system while working with unfamiliar technologies demanded a steep learning curve. One of the foremost technical challenges was implementing a secure and functional authentication system using JWT (JSON Web Tokens). The team had to understand token generation, secure storage, and how to ensure that tokens were properly attached to user requests on the frontend and validated on the backend. Debugging this authentication flow, especially issues like token expiration resulting in unexpected logouts, took considerable time and deepened our understanding of web security concepts.

Another major technical hurdle was developing the laptop comparison feature, which required fetching real-time external pricing data to benchmark listings. The team chose to use the eBay API to retrieve market prices for specific laptop models. This task proved difficult due to several factors: understanding the API documentation, handling asynchronous HTTP calls, managing data parsing, and staying within API rate limits. Moreover, integrating this external data without slowing down the application required the team to implement basic caching mechanisms, which was a new concept for most of the team. Working through this helped the team better grasp not only external API usage but also performance optimization techniques.

On the organizational side, team collaboration posed its own set of challenges. With eight members on the roster, but only six actively contributing, coordinating tasks and aligning schedules was a persistent struggle. Early in the project, the team had to reach a consensus on the tech stack and system design, even though different team members had varying levels of familiarity with JavaScript, Java Spring, or SQL. The team mitigated this by assigning responsibilities based on individual strengths: for example, some members focused on the front-end interface while others handled the back-end server and database layers. However, merging front-end and back-end components proved complex. Differences in how data was structured or expected (e.g., JSON formats or object keys) led to multiple rounds of debugging, and the team had to adapt our API contracts iteratively to resolve inconsistencies.

Additionally, version control using Git introduced challenges, particularly for members unfamiliar with collaborative workflows. The team encountered merge conflicts, and in one

unfortunate case, code was accidentally overwritten due to improper branch handling. These incidents underscored the importance of clear communication and disciplined branching strategies, which the team gradually adopted by having more consistent commit messages, stricter pull request protocols, and regular sync-ups before significant merges.

Time management and scope control were perhaps the most persistent challenges. The team began with ambitious goals, like a built-in messaging system for buyer-seller communication and an admin dashboard for listing verification. However, the team soon realized it couldn't implement everything within the semester. Scaling back was difficult; the team was passionate about our ideas, but it needed to prioritize core functionality. For example, instead of developing a full listing approval interface, the team decided to rely on university email verification as an initial trust mechanism. The team introduced internal deadlines and reviewed our progress weekly to manage scope creep. Features that lagged were either simplified or scheduled for future phases of development.

Moreover, remote collaboration added its own complications. When team members couldn't meet in person, the team occasionally faced miscommunications. One such example was how the wishlist feature was interpreted differently by different members, leading to redundant work and misaligned expectations. To resolve this, the team improved our project log documentation, introduced shared diagrams, and conducted quick stand-up meetings to clarify misunderstandings early.

13.0 Lessons Learned

The development of CampusTech Connect has been an incredibly enriching learning experience for our team of undergraduate students, encompassing both technical and interpersonal growth. One of the most important lessons the team learned was the critical value of thorough planning and clear requirements before beginning development. Early in the project, the team dedicated time to understanding user needs and formally documenting requirements. This effort paid off significantly by providing the team with a strong foundation and roadmap. However, in areas where requirements were vague or rushed, the team encountered misinterpretations, rework, and delays. This acted as a reminder that rigorous requirement elicitation and upfront design are essential for smooth downstream implementation.

The team also gained a deep appreciation for user-centered design. By actively considering the needs and behaviors of our target users, students, through informal interviews and surveys, the team discovered that usability often depends on simplicity. For example, the team initially planned a complex, multi-step filtering system for laptop searches. However, feedback revealed students preferred a straightforward search bar with lightweight filters, prompting the team to simplify the workflow. This experience taught team members that building intuitive interfaces

requires stepping into the user's shoes, a principle team members will carry forward into future development work.

From a technical perspective, the team strengthened its understanding of full-stack web development. Concepts learned in coursework such as front-end/back-end separation, request/response life cycles, form validation, API integration, and security became real and tangible. The team experienced the entire software development lifecycle: from requirement gathering, design, and implementation to testing and consideration of deployment. This hands-on exposure revealed how tightly coupled these phases are and underscored why each phase is essential to building reliable software.

Equally valuable were the lessons learned from collaborative teamwork. Working in a team environment taught the importance of using tools like Git for version control, breaking down large tasks into smaller modules, and maintaining consistent communication. Initially, coordinating across different schedules and skill levels led to some confusion and delays. However, once the team instituted regular team meetings and weekly check-ins, it achieved better synchronization. The team also learned that clarity in communication is paramount. When everyone shares the same understanding of a feature, development flows efficiently. But when assumptions differ, problems arise. To avoid this, the team started maintaining a shared Project Log that documented all tasks, feature definitions, and individual responsibilities, which became central to our workflow.

The team also discovered the importance of being flexible and supportive within the team. On many occasions, a team member struggling with a bug or logic issue would receive help from another, leading to faster resolution and improved morale. This reinforced the idea that software development is a collaborative process, not a solitary one. Through this collaboration, the team became more comfortable with asking for help, reviewing each other's work, and working across different parts of the system.

Another key takeaway was time management. Balancing other academic workloads with the demands of this project was challenging. The team learned to prioritize core features, defer non-essential functionalities, and set internal milestones to keep the project on track. Knowing when to scale back and focus on what's critical, such as abandoning our initial plan for a full messaging system in favor of simpler contact methods, was a practical and valuable skill in managing project scope.

Lastly, this project taught the team resilience and structured problem-solving. Rather than being discouraged by bugs, integration issues, or late-stage surprises, it learned to approach problems methodically, through debugging, research, and peer collaboration. Whether through online documentation, community forums, or team brainstorming, the team found ways to overcome

hurdles. This increased the team's confidence in tackling unfamiliar problems and reinforced that with persistence, most issues are solvable.

In conclusion, through CampusTech Connect, the team not only built a working web application it is proud of, but also developed critical technical skills, learned how to collaborate effectively, and grew in its understanding of real-world software engineering practices. These lessons have prepared the team to take on more ambitious projects in the future with a more structured, empathetic, and confident approach.

References

[1] EDUCAUSE, “Student Experiences with Connectivity and Technology in the Pandemic,” 2021. [Online]. Available: https://www.educause.edu/ecar/research-publications/2021/student-experiences-with-connectivity-and-technology-in-the-pandemic/introduction-and-key-findings

[2] Inside Higher Ed, “Undergraduates’ Technology Problems and Needs,” Oct. 4, 2022. [Online]. Available: https://www.insidehighered.com/news/2022/10/04/undergraduates-technology-problems-and-needs

[3] ProductPlan, “Requirements Gathering Techniques for Agile Product Teams,” *ProductPlan Blog*. [Online]. Available: https://www.productplan.com/learn/agile-product-management/

Appendices

Appendix A
Requirements Gathering Process for CampusTech Connect

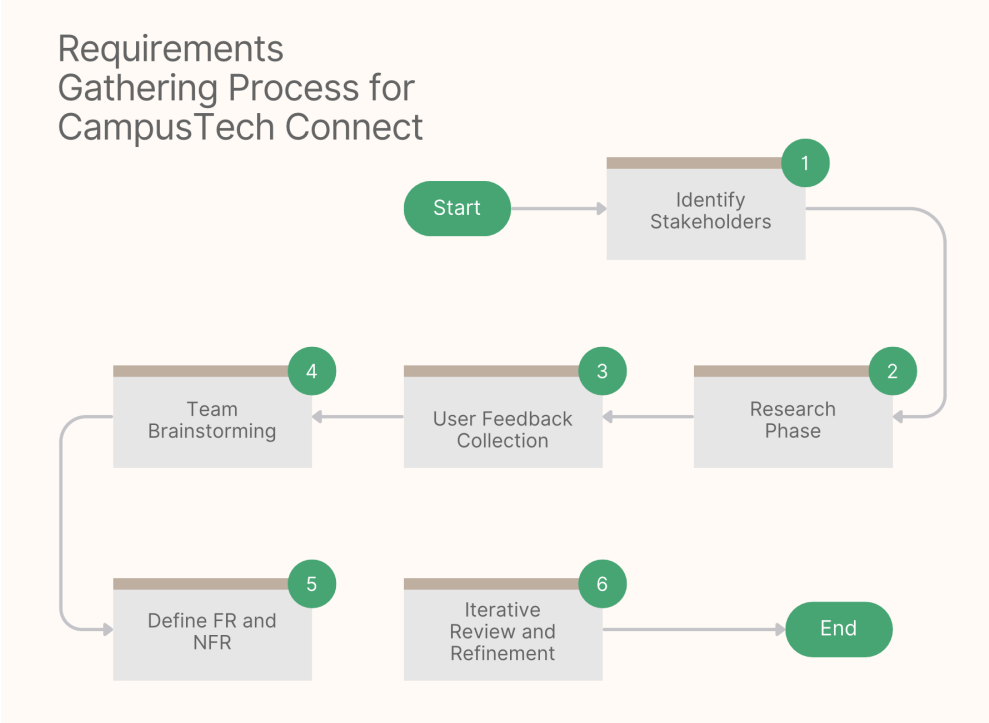

Figure A.1 Graphical View of Requirements Gathering Process.

The first step in our requirements gathering process was to **identify stakeholders**, specifically focusing on our primary target audience: university students who act as both buyers and sellers of laptops. The team recognized that this demographic had unique needs compared to the general public, academic performance, budget constraints, and campus-based interactions were central to their device-related decisions.

Next, the team entered the **research phase**, which involved reviewing existing literature and survey data on student laptop usage. The team analyzed statistics related to device malfunctions, software compatibility issues, and financial barriers. This data provided quantitative backing for the observed pain points and served as a foundation for identifying initial system goals.

Following our research, the team moved into **user feedback collection**, where the team gathered qualitative insights through informal interviews (see Appendix B) and a short cross-faculty survey (see Appendix C). The interviews helped the team understand personal frustrations and trust concerns with platforms like Kijiji, while the survey revealed desired features and trust-building elements for a student marketplace.

With both data and user feedback in hand, the team conducted a **team brainstorming session** (see Appendix D) to map these needs to actionable features. Each team member contributed from their own academic background to prioritize requirements. This collaborative session allowed the team to converge on key functionalities like email verification, wishlist tracking, and the recommendation engine, tailored specifically to academic use cases.

The team then defined the **functional requirements (FRs)** of the system. These included critical user services such as account registration, secure login, listing creation, filtering options, wishlist and target price features, and the laptop comparison tool. These FRs captured what the system must do to meet student needs.

Parallelly, the team established **non-functional requirements (NFRs)** that dictated how the system must behave. These focused on maintaining high security standards (e.g., HTTPS, hashed passwords), usability (clean UI, mobile support), and performance (fast response times, indexed database queries). Compatibility with multiple browsers and long-term maintainability were also emphasized.

The final step was an **iterative review and refinement phase**, where the team evaluated the feasibility of each proposed feature against the available timeline and resources. During this phase, the team made scope adjustments, prioritized core functions, and ensured alignment between technical implementation and user expectations. This cyclical refinement process helped the team develop a realistic, coherent set of system specifications.

Appendix B
Informal Interview Transcripts

Interview 1 – Computer Science Student (UofT)
Date: February 20th, 2024
Role: 3rd-Year Undergraduate
Faculty: Computer Science

Q1: Can you describe any challenges you've faced with your laptop during coursework?
My laptop's been lagging a lot, especially when I'm using VSCode with extensions or running Docker. It's okay for writing essays, but when it comes to technical work, it just can't keep up. It gets frustrating during assignment deadlines.

Q2: Have you used any platforms to try and upgrade or buy a new device?
Yeah, I checked out Kijiji and Facebook Marketplace. But honestly, I had a bad experience. I reached out to a seller and they ghosted me after I asked for specs. There's no guarantee they're students or even legit.

Q3: What would make you more confident in using a campus-specific laptop marketplace?
Definitely knowing the seller is a student. If you required a UofT email to sign up or list items, that'd help a lot. Also, filters for laptops that run programming software or handle VM workloads would be a game-changer.

Interview 2 – Design Student (OCAD)
Date: February 20th, 2024
Role: 2nd-Year Undergraduate
Faculty: Design

Q1: What kinds of performance issues have you faced with your device?
Adobe Illustrator and Photoshop just kill my battery, and rendering stuff in After Effects is basically impossible unless I leave my laptop plugged in all day. I bought this machine during first year thinking it'd be fine, but clearly not.

Q2: Did you consider buying a used device online?
I tried Facebook Marketplace, but the posts were super vague. I didn't want to waste time messaging people only to find out their laptop didn't have a good enough GPU. It just felt like too much of a hassle.

Q3: What features would you like to see in a student-specific laptop marketplace?

I'd love to see listings grouped by design suitability, like "good for Adobe Suite" or "4K color-accurate display." Plus, if you can show who's a real student, it would build trust right away.

Interview 3 – Engineering Science Student (UofT)
Date: February 21st, 2024
Role: 1st-Year Undergraduate
Faculty: Engineering Science

Q1: Have you struggled with your device's capability since entering university?
It's okay now, but I've been warned by upper-years that MATLAB and AutoCAD run really slow on older laptops. So I've started looking for a used one with better specs before things get intense.

Q2: Would you prefer a university-verified marketplace for this?
100%. I'd rather meet someone from my campus than a random seller. Also, if I can compare specs or even see benchmark scores, that'd be helpful. Trust and performance matter most for me.

Appendix C
Short Cross-Faculty Student Survey Summary

Purpose: To collect quick insights from students in different academic programs about their expectations for a campus-based laptop marketplace.

Date Distributed: February 24th, 2024
Platform: Google Forms
Respondents: 22 students
Faculties: Engineering (9), Commerce (4), Computer Science (5), Design (4)

Survey Questions and Results:

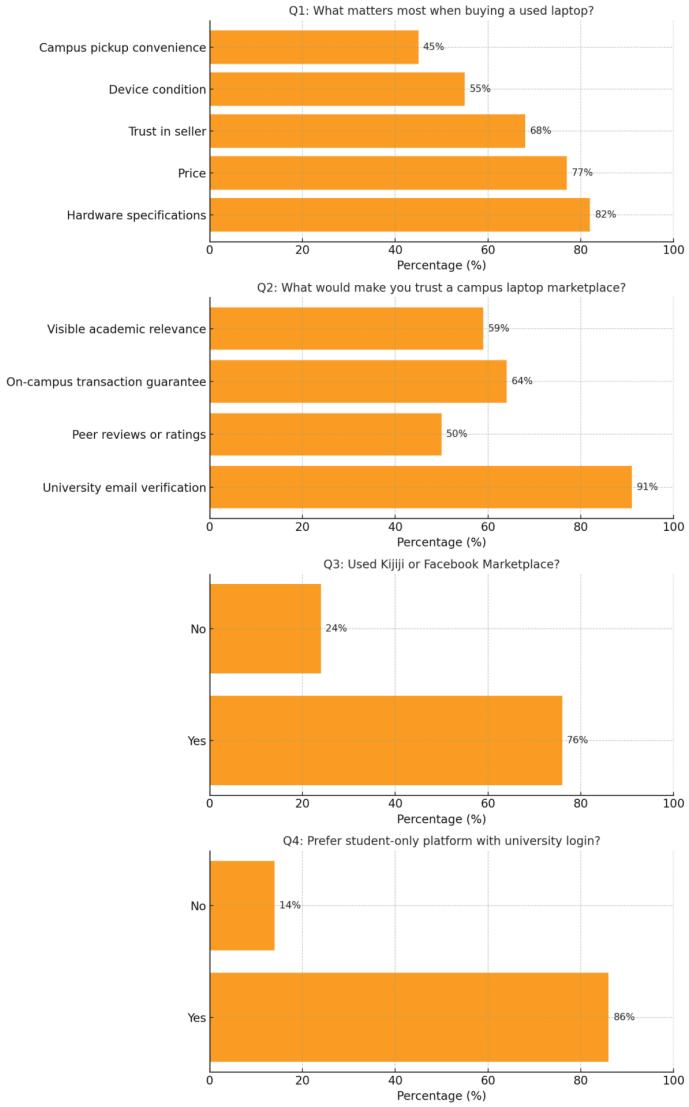

Figure C.1 Survey Results

Summary Insight:
Students overwhelmingly value specs, price, and seller credibility. A majority want university email verification and student-only access. Academic filtering and performance guidance were strong recurring themes.

Appendix D
Team Brainstorming Session Notes

Date: February 28, 2024

Participants:
- Samrath Singh (Industrial Engineering, 3rd Year)
- Kuval Brar (Industrial Engineering, 3rd Year)
- Ali Ahmad (Industrial Engineering, 3rd Year)
- Michael Furlano (Industrial Engineering, 3rd Year)

Key Discussion Points:

- Samrath: Suggested robust authentication using verified university emails, plus JWT for secure session management. Emphasized trust as the foundation of the platform.

- Kuval: Proposed laptop filters based on student use-cases: programming, design, etc. Also suggested tagging listings with academic relevance to help non-technical users.

-Ali: Highlighted the importance of a wishlist and target price feature, citing that students often plan large purchases in advance and need a tool to track changing prices.

-Michael: Focused on UI/UX simplicity. Advocated for intuitive search, minimal clicks, and clean navigation. Noted that too many features can overwhelm the user.

Final Consensus:

- Require university email to register and list items.
- Include listing filters by program or usage type.
- Smart comparison tool to evaluate specs and pricing.
- Minimalist, mobile-responsive interface for quick use between classes.
- Recommendation engine based on academic needs.

Appendix E
Mobile Optimization Strategy

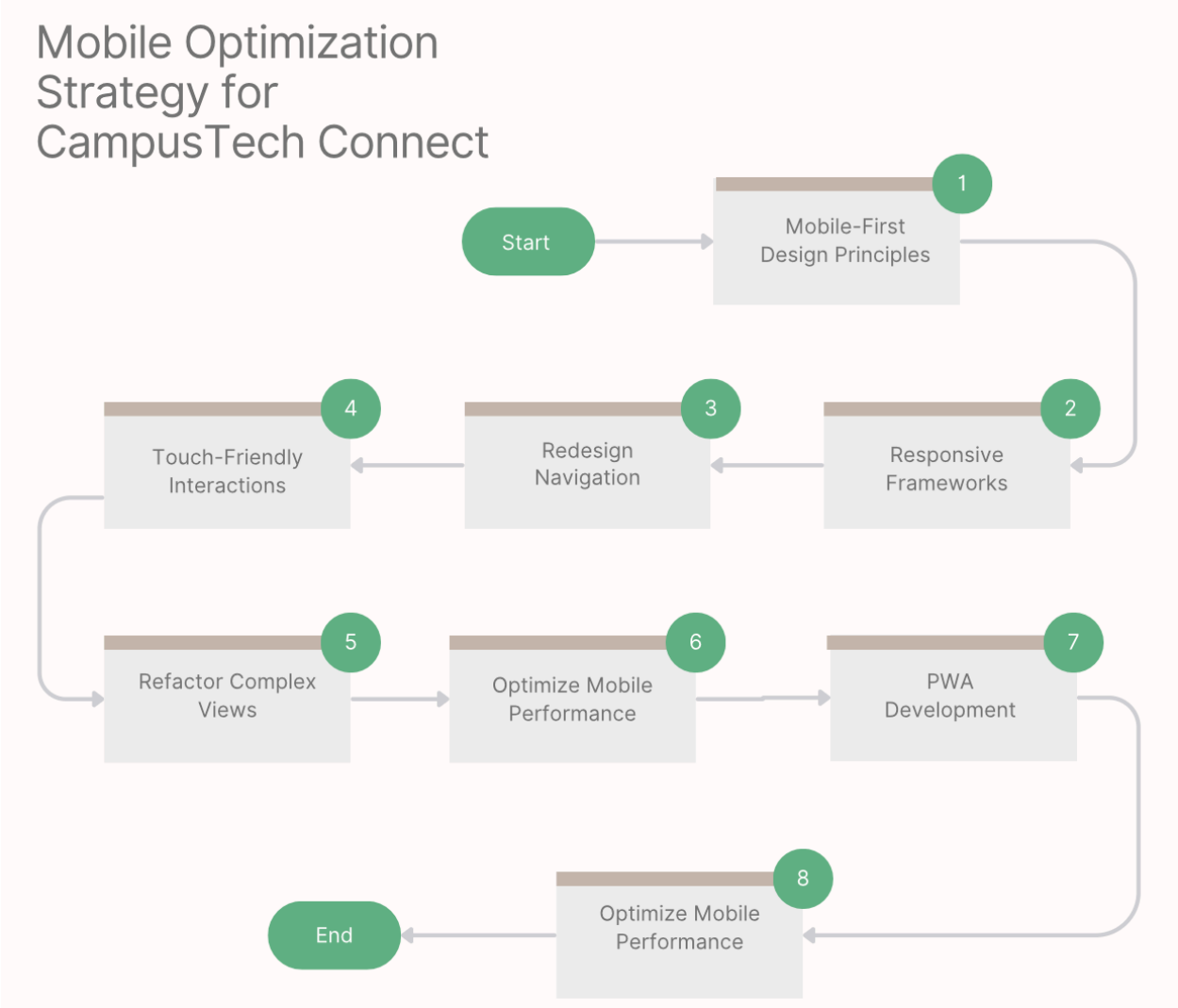

Figure E1. Graphical View of Mobile Optimization

The first step in our mobile optimization strategy will be the adoption of mobile-first design principles. Rather than scaling down desktop layouts, our team will prioritize smartphone and tablet interfaces as the default experience. This shift will ensure that essential interactions and user flows are designed with smaller screen constraints in mind, providing a streamlined experience from the outset.

Next, the team will implement responsive frameworks such as Tailwind CSS and Bootstrap. These tools will allow us to build adaptive components with greater efficiency and consistency.

By leveraging pre-built utility classes and responsive breakpoints,the team will reduce manual styling and improve cross-device compatibility.

With a responsive foundation in place, the team will then turn our attention to navigation. Traditional desktop menus will be replaced by collapsible “hamburger” menus for mobile users. Tap targets will be resized and spaced appropriately to accommodate users with varying hand sizes and to ensure accessibility compliance.

Touch-friendliness will be a major focus of the next phase. Features that currently rely on hover interactions, such as tooltips and dropdowns, will be redesigned to function with taps and long-presses. All interactive elements will be reviewed to meet the minimum 48x48 pixel tap area guideline, enhancing usability for mobile users.

The team will also tackle complex views such as the comparison feature. Recognizing that multi-column layouts do not translate well to mobile, the team will restructure these components using vertical stacking and swipeable carousels. This will preserve the interactive nature of the tool while maintaining readability on smaller devices.

To enhance performance on mobile networks, the team will introduce several optimizations. These will include responsive images, lazy loading of non-critical assets, and content prioritization. Together, these measures will ensure faster load times and a smoother experience, especially for users with limited bandwidth.

Recognizing the value of a more native-like mobile experience, the team will explore the development of a Progressive Web App (PWA). This will involve enabling offline access, push notifications, and home screen installation. The goal will be to deliver app-like convenience without the overhead of traditional app store deployment.

Finally, the team will conduct extensive device-specific testing and quality assurance. The system will be evaluated across various screen sizes, operating systems, and browsers. Usability testing sessions will help uncover areas for improvement, which will then be addressed through an iterative refinement cycle to ensure optimal performance and user satisfaction on all mobile platforms.

Appendix F
Automated Test Documentation

This appendix outlines the integration and backend tests executed using mockMvc in Spring Boot. All tests passed successfully and confirmed the expected behavior of critical backend functions.

Test Suite 1: RegistrationTests.java

Purpose: Verify correct student registration, handle duplicate accounts, and enforce university email constraints.

Test ID	Test Description	Input	Expected Output	Result

RT-01	Register new user with valid university email	{
		email: "john@utoronto.ca", password: "secure123"
		}
			201 Created, user saved in DB	Pass

RT-02	Register duplicate email	{
		email: "john@utoronto.ca", password: "secure123"
		}
		Same as RT-01
			400 Bad Request, error message "User already exists"	Pass

RT-03	Register with non-university email	{
		email: "john@gmail.com", password: "secure123"
		}
			400 Bad Request, error message "Invalid university email"	Pass

Test Suite 2: LoginTests.java

Purpose: Ensure users can log in with correct credentials and are rejected for incorrect credentials.

Test ID	Test Description	Input	Expected Output	Result

LT-01	Login with correct credentials	{
		email: "john@utoronto.ca",
		password: "secure123"
		}
			200 OK, JWT token returned	Pass

LT-02	Login with incorrect password	{
		email: "john@utoronto.ca",
		password: "wrong"
		}
			401 Unauthorized	Pass

LT-03	Login with non-existent user	{
		email: "nonuser@utoronto.ca",
		password: "123"
		}
			404 Not Found	Pass

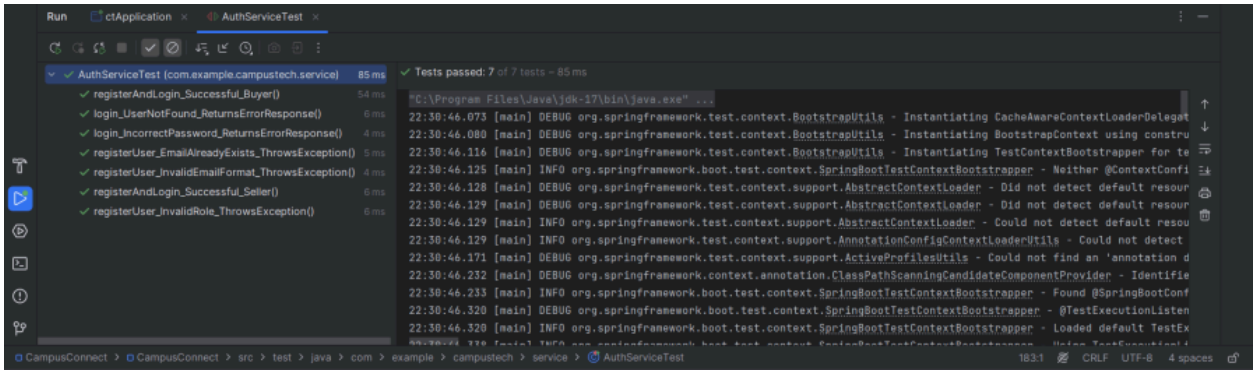

Figure F1. Registration + Login Tests cases

Test Suite 3: ListingTests.java

Purpose: Test seller's ability to create, retrieve, and delete laptop listings.

Test Id	Test Description	Input	Expected Output	Result

LTG-01	Create new laptop listing	{
		title: "MacBook Pro",
		specs: "...",
		price: 800
		}
			201 Created, entry saved	Pass

LTG-02	Retrieve seller's listings	Authenticated GET request	200 OK, array of listings returned	Pass
LTG-03	Delete own listing	Listing ID + valid token	200 OK, entry removed	Pass

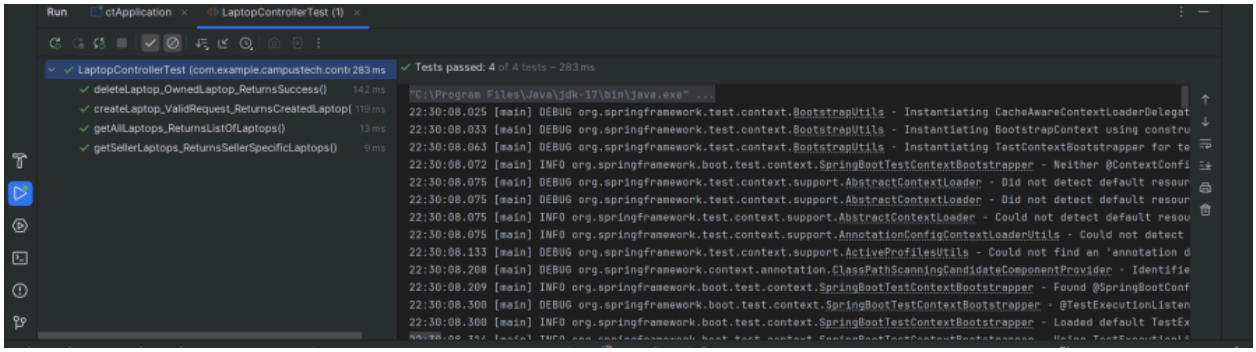

Figure F2. Laptop Listing and Viewing Test Cases

Test Suite 4: WishlistTests.java

Purpose: Test buyer's ability to add, update, view, and remove laptops from wishlist.

Test Id	Test Description	Input	Expected Output	Result
WT-01	Add laptop to wishlist	{ laptopId: 101, targetPrice: 600 }	201 Created	Pass
WT-02	Add duplicate laptop	Same as WT-01	400 Bad Request, "Already in wishlist"	Pass
WT-03	View wishlist	GET with token	200 OK, list of items	Pass
WT-04	Update target price	{ laptopId: 101, newPrice: 550 }	200 OK, wishlist item updated	Pass
WT-05	Remove wishlist item	Laptop ID	200 OK, item removed	Pass

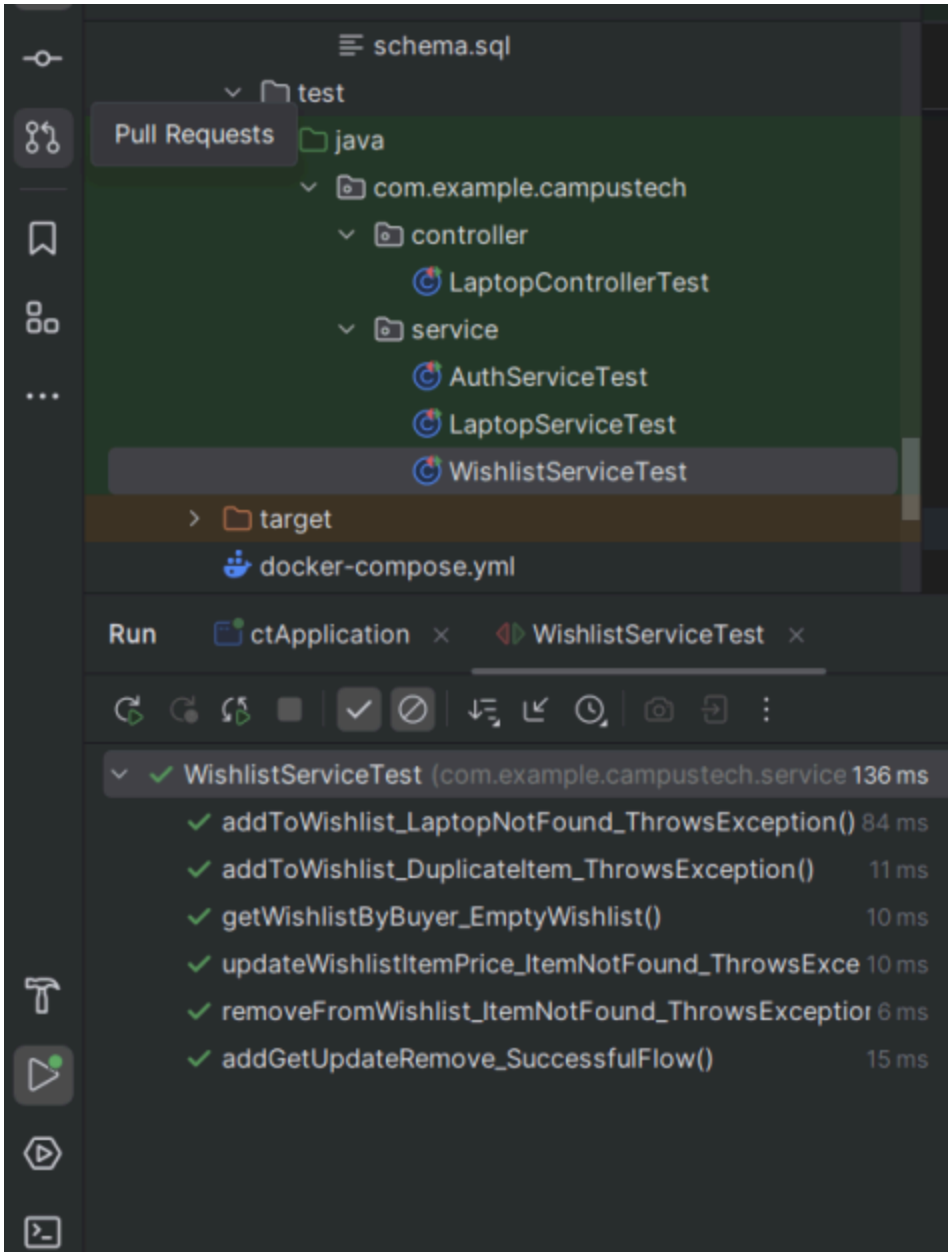

Figure F3. Wishlist Viewing and Workflow Test Cases

Appendix G
Informal Usability Testing Results

Objective: To evaluate the usability, layout clarity, and intuitiveness of CampusTech Connect by observing how non-developer users interact with key features. The goal was to identify friction points in user experience that could be improved before project finalization.

Method: The team invited 4 undergraduate students from the University of Toronto, who were not involved in the development to test the system in its beta stage. Each participant was asked to complete a series of realistic tasks using the application on their own laptops. Observations were noted during these sessions, and a short follow-up questionnaire was administered.

Participant Details:

Participant Id	Faculty	Year	Technical Background
P01	Computer Science	3	Moderate
P02	Engineering	1	Low
P03	Design	2	Low
P04	Rotman	4	Moderate

Figure G1. Table of Participant Info

Tasks Given:

1. Register using a university email.
2. Log in and browse available laptop listings.
3. Compare two laptops using the comparison tool.
4. Add one of the laptops to a wishlist.
5. Update the wishlist item with a new target price.

Findings & Observations:

Issue Id	Observation	Impact	Suggested Fix
UI-01	“Compare” button is not easily visible after	Moderate	Make the button more prominent; consider

	selecting laptops.		floating action.
UI-02	Participants hesitated when choosing between “Wishlist” and “Compare.”	Low	Add brief tooltips or a help icon next to each button.
UI-03	Dropdown options (like RAM or price filters) were not clearly labeled.	Moderate	Improve label descriptions and default selection text.
UI-04	Wishlist target price edit button was unclear to some users.	Moderate	Add an “Edit” icon or text next to each wishlist item.
UI-05	P02 tried hovering on a button expecting info, which didn’t work on mobile.	Low	Replace hover-tooltips with click/tap-based descriptions.

Figure G2. User Feedback Summary (From Post-Test Survey):

- P01 (CS): “The comparison view is great but could use a visual highlight to show differences.”
- P02 (Eng): “I wasn’t sure if the wishlist had a price drop alert. That’d be useful.”
- P03 (Design): “Loved the academic focus. Icons and layout are clean, but a bit small.”
- P04 (Commerce): “It’s fast and intuitive, I’d use this during back-to-school season.”

Conclusion: The informal usability testing surfaced several minor but meaningful user experience issues that were not previously caught during internal testing. Most of the problems

were related to visual clarity, button prominence, and instructional support. Several suggestions were implemented immediately (e.g., repositioning buttons, improving labels), while others (like tooltip redesign and accessibility enhancements) are scheduled for future iterations.

This testing exercise affirmed that CampusTech Connect's core workflows are intuitive, but UI refinements are essential to improve adoption and engagement, particularly for non-technical users.