**MILESTONE 1 Report**

**CS396 Software Design Principles and Practices**

**People**: Al Nahiyan and Saeed AlKaabi

## Option

Option 1 - Design Your Own Product

## Link to Project GitHub Repository

https://github.com/nu-cs-sw-design/project-20252601-396fp

# Project Planning

Week 1: Requirements and Initial Design

- Define complete use cases and user stories
- Create initial architectural design considerations
- Identify quality attributes and priorities
- Submit Milestone 1 report

Week 2: Implementation of Messy Prototype

- Implement core functionality (user authentication, item listing, rental requests)
- Create basic UI/UX for primary workflows
- Focus on getting features working (prioritize functionality over design)
- Submit Milestone 2 progress report

Week 3: Design Analysis and Refinement

- Analyze messy code for design principle violations
- Create comprehensive class diagrams
- Document architectural decisions
- Identify refactoring opportunities
- Prepare final deliverables and submit

# Initial Analysis

## Purpose

The Campus Rental Platform enables students to participate in a peer-to-peer rental marketplace exclusively within their academic community. The platform addresses the financial constraints students face by providing an affordable alternative to traditional rental services through direct student-to-student transactions.

**Key Value Propositions:**

- **For Rentees (Borrowers):** Access items at significantly reduced costs compared to commercial rental services, with convenient campus-based pickup and return
- **For Renters (Lenders):** Generate supplementary income from underutilized possessions while helping fellow students

- **For the Community:** Foster a sustainable, collaborative economy that builds trust and connections within the student body

## Product Description

The Campus Rental Platform is a mobile-first web application that facilitates secure, time-bound rentals between verified students. The system manages the complete rental lifecycle from item discovery to payment processing and dispute resolution.

## Target Users

**Primary Users:**

1. **Full-Time Campus Students** - Students enrolled in on-campus programs with regular access to campus facilities
2. **Part-Time Campus Students** - Students with limited campus presence who need occasional access to items

**Secondary Users:**

3. **Visiting/Exchange Students** - Temporary students who need short-term rentals without long-term commitments

4. **Campus Administrators** - (Future consideration) For oversight and policy enforcement

**User Constraints:**

- Must have valid .edu email address for verification
- Must be currently enrolled students
- Must have access to campus for item exchange

**Core Product Features**

**1. User Authentication and Verification System**

- Dual email registration (primary contact email + university .edu email)
- Student status verification through university email confirmation
- Secure login with password recovery

**2. Item Listing Management**

- Create detailed rental listings with photos, descriptions, and pricing
- Set availability calendars and rental durations (hourly, daily, weekly)
- Define pickup/return locations (campus locations, dorm buildings, etc.)
- Categorize items (electronics, textbooks, sports equipment, tools, etc.)
- Mark items as unavailable or remove listings

**3. Discovery and Search System**

- Browse available items by category
- Filter by: price range, location, availability dates, item condition, renter ratings
- AI-powered recommendation engine matching rentee searches to relevant listings
- Interactive campus map showing item locations and proximity
- Save favorite items and create wish lists

**4. Rental Request and Booking Flow**

- Request to rent with proposed dates and times
- Calendar integration showing availability
- Rental agreement generation with terms and conditions
- Renter approval/denial with optional messaging
- Automated booking confirmations (Maybe use third party calendar tools like Calendly or Google Calendar)

**5. Payment Processing**

- Integrated payment gateway supporting multiple methods (credit/debit, digital wallets, digital services like Adyen)
- Security deposit handling (held during rental period)
- Automatic payment release after successful return
- Transaction history and receipt generation
- Refund processing for cancellations

**6. Communication System**

- In-app messaging between renters and rentees
- Automated notifications for:
    - New rental requests
    - Booking confirmations/denials
    - Payment confirmations
    - Pickup reminders

    - Return reminders
    - Late return warnings
- Real-time chat for coordination
- Message history and archiving

**7. Reputation and Trust System**

- Two-way rating system (renters rate rentees, rentees rate renters)
- Written reviews with timestamps
- Reputation scores affecting search rankings
- Badge system for reliable users (verified, top renter, responsible rentee)
- Report functionality for inappropriate behavior

**8. Scheduling and Logistics**

- Pickup location coordination (with map pinning)
- Proposed pickup times with confirmation workflow
- Return scheduling with automated reminders
- Extension requests during active rentals

**9. User Profile and Dashboard**

- View rental history (as renter and rentee)
- Track active rentals
- Manage listings
- Update payment methods
- Adjust notification preferences
- View earnings and spending analytics

**Future considerations:**

**Dispute Resolution Framework**

- Report damaged or non-returned items
- Photo evidence upload
- Admin review system (future phase)
- Insurance claim integration (future phase)

## Use Cases

### UC-1: User Registration and Verification

**Actor:** New Student User
**Preconditions:** Valid .edu email

**Main Flow:**

1. User opens registration page
2. User enters info and .edu email
3. System sends verification email
4. User clicks verification link
5. System activates account

**Postconditions:** Verified account created

**Alternatives:**

- If the email is not received, the user can resend it
- If the link expires, the user requests a new link
- If the email is not a .edu email, the system asks for a valid university email

### UC-2: Create Rental Listing

**Actor:** Student Renter
**Preconditions:** Logged in and verified

**Main Flow:**

1. User opens Create Listing
2. User uploads photos and enters item details
3. User sets availability and pickup location
4. System validates listing
5. Listing is posted

**Postconditions:** Listing becomes visible

**Alternatives:**

- If required fields are missing, the system highlights them
- If an image is too large, the system compresses or asks for a smaller file

**UC-3: Search and Request Item Rental**

**Actor:** Student Rentee
**Preconditions:** Logged in and verified

**Main Flow:**

1. User searches or browses
2. System shows results and filters
3. User views item
4. User selects rental dates
5. System shows total cost
6. User submits request

**Postconditions:** Rental request is created

**Alternatives:**

- If dates are unavailable, the system suggests alternatives
- If no items match, the system suggests broader search options

**UC-4: Approve or Deny Rental Request**

**Actor:** Student Renter

**Main Flow:**

1. Renter views new request
2. Renter reviews details
3. Renter approves or denies
4. If approved, rentee submits payment
5. System confirms booking

**Postconditions:** Rental is approved and paid

**Alternatives:**

- If denied, the system notifies the rentee
- If payment fails, the user updates their payment method

**UC-5: Pickup Item**

**Actor:** Student Rentee

**Main Flow:**

1. System sends pickup reminder
2. Users meet at location
3. Rentee inspects item
4. Rentee confirms pickup
5. System starts rental period

**Postconditions:** Item handed to rentee

**Alternatives:**

- If item condition is wrong, the rentee cancels
- If someone does not show up, the system logs it
- If the rentee forgets to confirm, the system auto-confirms after 24 hours

**UC-6: Return Item**

**Actor:** Student Rentee

**Main Flow:**

1. System sends return reminder
2. Users meet
3. Renter checks item
4. Renter confirms return

5. System releases deposit and closes rental

**Postconditions:** Rental finished

**Alternatives:**

- If damage is found, the system creates a dispute
- If the return is late, the system applies fees

**UC-7: Request Extension**

**Actor:** Rentee

**Main Flow:**

1. User opens active rental
2. User selects new return date
3. System calculates added cost
4. Renter approves
5. System processes payment and updates rental

**Postconditions:** Rental extended

**Alternatives:**

- If dates are unavailable, the system suggests another date
- If the renter denies the request, the original return date remains
- If payment fails, the extension is cancelled

**UC-8: Report Dispute**

**Actor:** Renter or Rentee

**Main Flow:**

1. User opens Report Issue
2. User selects dispute type and uploads evidence

3. System notifies the other party
4. Other party responds

5. System reviews the case or sends it to an admin
6. System issues a final decision

**Postconditions:** Dispute resolved

**Alternatives:**

- If the other party does not respond, the system uses existing evidence
- If policies are violated, the system applies penalty immediately

**UC-9: Manage Listing Availability**

**Actor:** Renter

**Main Flow:**

1. User opens My Listings
2. User edits availability or marks item unavailable
3. System saves changes

**Postconditions:** Listing updated

**Alternatives:**

- If pending requests are affected, the user contacts the rentee
- If listing is deleted, pending requests are cancelled

**UC-10: AI Recommendations**

**Actor:** Rentee

**Main Flow:**

1. User opens home or search
2. System analyzes activity

3. System shows recommended items
4. User selects a recommendation

**Postconditions:** User receives personalized suggestions

**Alternatives:**

- New users see trending items
- If a user dismisses items, the system adjusts future recommendations

## Functionalities

Below are the functionalities that our design would need:

**User Account & Verification**

F1: System allows students to register using a primary email and a university (.edu) email
F2: System verifies student status through .edu confirmation
F3: System allows secure login, logout, and password recovery

**Listing Management**

F4: Users can create rental listings with photos, descriptions, and prices
F5: Users can set availability calendars and rental durations
F6: Users can define pickup and return locations
F7: Users can edit, update, or remove listings
F8: Users can mark listings as unavailable

**Search & Discovery**

F9: Users can browse items by category
F10: System supports filtering by price, availability, location, condition, and ratings
F11: System provides an interactive campus map showing item locations
F12: System provides AI-powered recommendations based on user behavior
F13: Users can save favorites and create wish lists

**Rental Request & Booking**

F14: Users can request items with proposed dates
F15: System displays availability and calculates total rental cost
F16: Item owners can approve or deny rental requests
F17: System generates confirmations for both parties
F18: Users can reschedule or negotiate via in-app messaging

**Payments**

F19: System processes secure payments
F20: System handles security deposits
F21: System releases deposits after successful return
F22: System manages refunds when rentals are canceled or invalid

**Communication**

F23: Users can message each other in-app
F24: System sends notifications for all rental events (requests, approvals, reminders, late alerts)
F25: System keeps message history

**Pickup & Return Logistics**

F26: System sends pickup reminders
F27: Users confirm pickup in the app
F28: Users confirm return in the app
F29: System applies late fees if needed
F30: Users can request extensions

**Reputation & Trust**

F31: Users can rate and review each other
F32: System calculates reputation scores
F33: Users can report inappropriate behavior

**Dispute Handling**

F34: Users can report issues such as damage or no-shows
F35: System collects evidence from both parties
F36: System resolves disputes or escalates to admin

**Future / Advanced Functionalities**

F37: System integrates optional insurance for high-value items
F38: System provides fraud prevention and user verification upgrades
F39: System allows campus administrators to oversee activity (future)

## Software Quality Attributes

Below are the software quality attributes that we aim for.

**1. Availability**

**Client Need:** Users must access the system anytime, especially for pickup/return coordination.
**Scenario:** If the system experiences high user load or a subsystem fails, the platform stays online and responds normally.
**Response Measure:** 99.5% uptime or higher
**Importance:** Critical for time-sensitive rentals and coordinating campus meetups

**2. Scalability**

**Scenario:** If the number of listings or active users doubles during peak semester times, the system continues functioning without noticeable slowdown.
**Response Measure:** No more than 10% degradation in response time
**Importance:** The platform expands with student population and seasonal demand

**3. Security**

**Scenario:** When users log in, only authorized roles (renter, rentee, admin) access protected features. All personal information is encrypted.

**Response Measure:** Zero unauthorized access; 100% encryption
**Importance:** Protects student identity and financial transactions

**4. Usability**

**Scenario:** A new user should be able to find and request an item in under 3 steps without assistance.
**Response Measure:** ≥90% task completion during usability tests
**Importance:** Students need a simple, intuitive interface

**5. Maintainability**

**Scenario:** Developers can deploy a small update (bug fix, UI change) without taking the system offline.
**Response Measure:** Updates deploy in < 30 minutes with zero downtime
**Importance:** Ensures system stability throughout the academic year

**6. Data Accuracy**

**Scenario:** Availability calendars and rental status must always reflect the correct user actions without conflict or delay.
**Response Measure:** ≥95% accuracy in operations
**Importance:** Prevents double-booking and rental disputes

**Priority Ranking with Justification:**

1. **Security** (Highest Priority)
    - Justification: Without strong security, the platform cannot protect student data, financial information, or prevent fraud. A security breach would destroy user trust and could expose the university to liability. This is foundational—all other attributes depend on users feeling safe using the platform.
2. **Availability**
    - Justification: If the system is down during critical times (pickup coordination, exam week rentals), students cannot complete transactions. High availability ensures students can rely on the platform for time-sensitive needs, which is essential for a campus service.
3. **Data Accuracy**

    - Justification: Incorrect availability or double-booking undermines the entire rental model and leads to disputes between students. Accurate data is critical for maintaining trust and preventing conflicts that could harm the community.
4. **Usability**
    - Justification: Students will abandon a confusing platform in favor of alternatives. Good usability drives adoption and reduces support overhead, making the platform sustainable.
5. **Scalability**
    - Justification: While important for growth, initial user base is limited to one campus. Scalability becomes more critical as the platform expands, but in early phases, other attributes take precedence.
6. **Maintainability**
    - Justification: Important for long-term sustainability but less immediately critical than security or availability. A maintainable system reduces costs over time and enables feature evolution, but it can be improved iteratively.

## Software Architectural Design

The Campus Rental Platform follows a layered architectural approach that separates the user interface, business rules, and data storage into three principal layers. This ensures maintainability, reliability, and clarity in the overall system structure. Inside the Presentation Layer, the MVC pattern is used to organize user interactions across views, controllers, and data models. Two design patterns are incorporated in the Domain Layer: the Strategy Pattern and the Observer Pattern.

**Three Principal Layers**

1. Presentation Layer (MVC)
   This layer provides all user-facing interfaces and handles user interactions.
   Responsibilities include:
   • Rendering pages for registration, verification, listing creation, search, item details, rental requests, messaging, and user dashboards
   • Displaying notifications, reminders, and updates
   • Performing basic input validation before handing requests to the domain layer
   • Using controllers to process actions such as submitting rental requests, approving bookings, confirming pickups, leaving reviews, or updating listings
   • Using models or view models to pass data from domain services to UI screens
   • Managing the map view for pickup locations and item proximity

MVC organizes this layer by separating:
• Views (screens and pages)
• Controllers that receive user actions and coordinate with domain logic
• Models that structure data passed between layers

2. Domain Layer (Business Logic)
   This layer contains all the business rules and the core functionality of the platform.
   Responsibilities include:
   • Verifying eligibility requirements such as student status
   • Managing the rental lifecycle from request, approval, activation, return, and completion
   • Validating availability and enforcing rental constraints
   • Calculating rental fees, extension fees, deposits, and late fees
   • Managing search ranking logic and recommendation strategies
   • Triggering appropriate notifications for approvals, changes, reminders, and deadlines
   • Updating user trust metrics based on reviews and completed rentals
   • Handling dispute creation, evidence submission, and resolution logic
   • Coordinating payment actions using the data source layer

This layer defines the main domain entities (User, Listing, Rental, Review, Dispute) and service classes (RentalService, ListingService, PaymentService, RecommendationService, NotificationService).

3. Data Source Layer (Persistence and Integrations)
   This layer manages all data storage and external system communication.

Responsibilities include:

• Storing user accounts, verification status, listings, availability, messages, rentals, reviews, and disputes
• Managing rental histories and payment records
• Communicating with external services such as payment processors, email verification tools, or notification systems
• Applying encryption and secure access rules to sensitive data
• Handling temporary failures or retries when external services are unavailable
• Providing repository classes that abstract away database schemas

This separation ensures the Domain Layer stays unaffected by database changes or external service upgrades.

**Design Patterns Used**

1. Strategy Pattern

   The Strategy Pattern is used in places where the system needs to support multiple interchangeable algorithms.

   In this platform, strategies are used for:

   • Fee calculation (different ways to compute rental cost or late fees)
   • Pricing logic (standard pricing vs. dynamic pricing for high-demand items)
   • Recommendation logic (content-based recommendations vs. popularity-based suggestions)

These strategies can be swapped or extended later without modifying the core business workflow, which supports modifiability and maintainability.

2. Observer Pattern

   The Observer Pattern is used for event-driven actions throughout the system.

   Examples include:

   • When a rental request is approved, the notification service receives the event and sends alerts
   • When a return deadline is approaching, the notification system receives the event and creates reminders
   • When a rental is completed, the reputation service updates the renter and rentee

scores

Using Observer keeps the rental workflow independent from notification logic or analytics, making the system easier to extend.

**Rationale**

A layered architecture, supported by MVC, provides clarity and separation of concerns. The Strategy Pattern and Observer Pattern directly address two major needs in the platform: the need for flexible algorithms (pricing, recommendations) and the need for asynchronous event handling (notifications and reminders). This combination allows the system to remain clean, maintainable, and ready for future expansion without unnecessary complexity.