**WEEK 1**

* **Project Title:** CampusConnect
* **Problem Statement :**
* Difficulty finding and joining study groups for specific courses or topics.
* Lack of a centralized platform to stay updated on campus events, deadlines, and club activities.
* Limited options for coursework discussions and resource sharing within a trusted student community.
* No secure, student-only space for buying and selling items like textbooks or dorm essentials.
* **Objectives:**
* **Facilitate Academic Collaboration**: Enable students to form study groups, share resources, and discuss coursework seamlessly.
* **Centralize Campus Information:** Provide a single platform for discovering and sharing university events, deadlines, and club activities.
* **Foster Community Engagement:** Create a space for students to connect with peers, join interest-based groups, and participate in campus life.
* **Ensure Security and Exclusivity:** Restrict access to verified university students to maintain a safe and relevant community.
* **Target Users:**
* **Primary users:** undergraduate students, university clubs
* **Key users Needs:**
* Easy access to study groups and academic resources.
* A reliable source for campus event information.
* A secure platform to connect with peers and buy/sell items.
* **User Characteristics**: Tech-savvy, active on mobile devices, values privacy, and seeks convenience.
* **Technologies and Frameworks:**
* **Frontend:** HTML, CSS, JavaScript
* **Backend:** Python with Django
* **Database:** SQLite
* **Expected Design Patterns:**

1. **Observer Pattern**
2. **Factory Pattern**
3. **Singleton Pattern**

* **Initial High-Level System Overview:**

1. **Frontend (Client-Side):**

**Tech:** HTML, CSS, JavaScript

**Responsibilities:**

* Display user interface: feed, profile, messages
* Handle UI interactions (like button clicks, post submissions)
* Fetch data via AJAX or Django views
* Dynamically update parts of the page

**2. Django Backend (Server-Side Logic):**

**Tech:** Python (Django framework)

**Responsibilities:** ****

* Handle HTTP requests (GET, POST, etc.)
* Route URLs to appropriate views
* Process logic for creating posts, comments, likes, etc.
* Serve HTML templates or JSON data
* Use Factory Pattern to manage post/notification creation
* Use Observer Pattern for internal event handling (e.g., signal when a post is liked)
* Use Singleton Pattern for config settings, DB connection handling

**3. Database (SQLite):**

**Tech:** SQLite

**Responsibilities:**

* **Store user data, posts, likes, comments, relationships**
* **Optimize query access via Django's querysets**
* **Can expand to use Redis for caching later**

**4. Media Storage:**

**Default:** Django's media storage + Filesystem or S3 (in production)  
**Responsibilities:**

* Upload and retrieve images/videos for posts and profiles
* Serve media files to users
* Integrate with CDN later for scalability

**5.Notification System (Observer-based):**

**Use:** Django Signals or custom event manager  
**Responsibilities:**

* + Trigger notifications when a user likes/comments
  + Can be extended to WebSockets using Django Channels (for real-time)