CS - 3810 - Design Practices in CS
Project Descriptions

Course Projects: Collaborative Text Editor, Image Editor, Public Issues Tracker

Instructions

Given below are the descriptions for your second project, which has already been assigned to your team. You can see the project allotment here. Additionally, you are required to submit code only on GitHub Classroom (and assignments on Google Classroom). Please ensure you have signed up for the same.

The second project is cloud based, i.e., you will deploy your Dockerized application on a cloud platform like AWS/Azure/GCP. Using serverless architectures like Vercel is not permitted. There are again 3 options in these projects, which have been randomly assigned to you.

This will be divided into the following four assignments:

Assignment # & Task	**Assign Date**	**Submit Date**
A4: Detailed Requirements Specification, Client-Server Architecture and High-level Design	*25-Oct-25*	*23:55, Oct 31, 2025*
A5: Low-level Design (classes, functions, protocols, etc.)	*31-Oct-25*	*23:55, Nov 07, 2025*
A6: Testplan and Test Suite	*08-Nov-25*	*23:55, Nov 17, 2025*
A7: Implementation and Test Deployment and Hosting	*18-Nov-25*	*23:55, Nov 25, 2025*

Various requirements (wish list) for the software are given below. Use these as guideline to prepare your details requirements specification that would drive your design, implementation, test, and deployment.

Some of the requirements are marked as *optional*. In your detailed specification, mark if you are including / excluding these. Inclusion of optional features successfully would be eligible for bonus marks.

Of course, you are free to add more requirements (without conflicting with the given ones). These will be appreciated, but would not carry any bonus.

1 Requirements for the Collaborative Text Editor

1.1 Functional Requirements

- *Real-time Collaborative Editing*:

  The software must allow a minimum of three users to edit a text document simultaneously.

  Changes made by one user should be reflected in real-time across all users' sessions.

- *User Interface (UI)*:

  The UI should be simple and responsive, supporting modern web browsers.

  Real-time user cursors: Each user's editing position (cursor) must be visible to all other collaborators with distinct identifiers (for example, colored cursors or labels).

- *Text Editing Features*:

  The editor should include basic text formatting features such as bold, italics, underline, font size adjustment, and bullet points (*optional*).

- *Document Management Features*:

  The editor should allow to open an existing document and create new ones.

  The user may download an online document as a Microsoft Word document.

  Note: No explicit option to save a document is needed, as collaborative editing needs autosave and sync.

- *Autosave and Document Syncing*:

  The document should be autosaved regularly to avoid data loss.

  Conflict resolution mechanism for handling simultaneous edits to the same part of a document.

- *Collaboration Features*:

  Users should be able to comment on specific portions of the text. (*optional*)

  There must be a notification system (or visual indicators) to alert users when someone else is editing the document.

- *Permissions and Access Control*: Document owners should be able to grant or restrict access to collaborators.

  Permission levels: Read, Comment, and Edit.

- *Version Control*:

  A document history feature should track changes over time, allowing users to view or revert to previous versions.

- *Accessibility*:

  The project must account for how users with different abilities can use them. Some things to focus on are screenreader friendliness, appropriate colour contrast, font spacing, font size and type. The above-mentioned are just a few of the visual factors, spend some time to think about how other parts of the application can be made accessible.

- *Scalability*: (*optional*)

  The software should support multiple users across various geographic locations, ensuring minimal latency during collaboration.

- *Backend*:

  The backend must run on a Unix-based system.

  It should handle user authentication, document storage, and synchronization across multiple users.

  A version control mechanism must be integrated to track changes.

1.2 Non-Functional Requirements

- *Performance*:

  The application should maintain low latency even when multiple users are editing the document.

- *Security*:

  Data must be encrypted during transmission (use of HTTPS protocol).

  Users should authenticate securely with username/password or OAuth.

- *Usability*:

  The interface should be intuitive, with clear indicators for collaboration and editing actions.

- *Compatibility*:

  The software must run on modern web browsers like Chrome, Firefox, Opera, Edge, and Safari.

- *Scalability*:

  The backend architecture should allow for horizontal scaling to support a growing number of users. (*optional*)

2 Photo Editing Service

2.1 Functional Requirements

- *User Accounts and Authentication*:

  Users should be able to register and log in securely using email/password or OAuth (e.g., Google, Facebook).

  Each user's photos and albums must be private by default, with optional sharing permissions.

- *Photo Upload and Management*:

  Users must be able to upload multiple photos at once (supporting JPEG, PNG, WEBP formats).

  Allow album creation, renaming, deletion, and photo rearrangement.

  Uploaded photos should be stored in a cloud bucket (e.g., AWS S3).

- *Image Editing Tools*:

  The editor must include essential features: crop, rotate, resize, brightness, contrast, and saturation adjustment.

  Advanced features such as filters, background blur, and AI-based enhancements are *optional*.

- *Search and Organization*:

  Users should be able to search photos by name, tags, or metadata (date, location, etc.).

  Automatic photo categorization (by people, places, or events) is *optional*.

- *Sharing and Collaboration*:

  Allow users to share photos or albums with others using view-only or editable links.

  Shared albums should display comments and likes in real time (*optional*).

- *Backup and Sync*:

  Changes (uploads, edits, deletions) should be synced automatically with the cloud backend.

  Periodic backups must ensure photo safety even in case of accidental deletion.

- *Notifications*:

  Users should receive notifications when someone comments on or shares an album with them. (optional)

  Email notifications for significant events (optional).

- *Backend Requirements*:

  The backend must handle user data, media storage, and authentication.

  Use of REST or GraphQL APIs to serve frontend requests.

  Must run on a Unix-based cloud instance (e.g., AWS EC2) with persistent cloud storage (S3 or RDS).

2.2 Non-Functional Requirements

- *Performance*:

  Uploads and edits should complete within acceptable latency limits (preferably under 3 seconds for basic edits).

- *Security*:

  All photos and metadata must be stored securely with access control and HTTPS encryption.

  Authentication tokens (JWT or OAuth) should protect API endpoints.

- *Usability*:

  Interface should be mobile-friendly and intuitive, with easy access to albums, editing tools, and sharing features.

- *Reliability*:

  The system must handle large uploads without data loss, using transactional or resumable uploads.

- *Scalability*:

  The backend and storage systems should scale to handle millions of images and users concurrently (*optional*).

3 Crowdsourced Map-Based Public Issues Tracker

3.1 Functional Requirements

- *User Accounts and Authentication*:

  Users should be able to register/login securely and submit issues under their profile.

  Guest submissions (with limited access) are *optional*.

- *Issue Reporting*:

  Users can report a public issue (e.g., pothole, garbage, broken light) with:

  - A short title and description
  - Location (via map pin or GPS)
  - Image upload if necessary

- *Map Interface*:

  Display all reported issues as pins on an interactive map.

  Each pin should be color-coded based on issue status (e.g., Open, In Progress, Resolved).

- *Issue Tracking and Updates*:

  Users should be able to view issue details, upvote if it affects them, and add comments.

  Authorities (admins) can update status or post official responses.

- *Filtering and Search*:

  The map must support filtering by issue type, location, or status.

  Keyword search should also be available for issue descriptions.

- *Notifications and Alerts*:

  Users should receive updates when an issue they reported or upvoted changes status.

  Nearby users may optionally receive alerts for new local issues.

- *Admin and Moderation Tools*:

  Admin users should be able to verify, categorize, or close issues.

  Automatic flagging or hiding of inappropriate content (*optional*).

- *Data and Analytics*:

  The system may provide a dashboard showing issue density, trends, or heatmaps (*optional*).

- *Backend Requirements*:

  The backend must handle authentication, issue submissions, map data storage, and status updates.

  Should integrate with a geolocation API (e.g., Google Maps, Leaflet, or OpenStreetMap).

  Backend hosted on a Unix-based server with persistent database (e.g., PostgreSQL + PostGIS).

3.2 Non-Functional Requirements

- *Performance*:

  Map rendering and updates must be smooth, even with hundreds of markers.

- *Security*:

  All data transmissions should be encrypted.

  Prevent unauthorized edits using role-based access control.

- *Usability*:

  The map interface must be simple and intuitive for both users and administrators.

  Desktop compatibility is essential.

- *Availability*:

  The application should maintain high uptime, with fallback for map data in case of external API failures.

- *Scalability*:

  The system should handle increasing numbers of issue reports and map interactions across cities (*optional*).