4. Design

4.2 Design Exploration

4.21 Design Decisions

1. Page Layout
    a. Since this project is a website, the user will only be interacting with the website. It is critical they can navigate through features and access the information they are looking for quickly. The layout of the page includes what information is shown on which page, what elements need to be accessible from the navbar, what needs to be displayed on each page, and many other visual elements. Without an easy-to-use interface, the software will not stand a chance of being used by anyone.
2. Design Flow
    a. The users need to be able to accomplish tasks quickly and easily. Avoiding overly complicated processes to create an account should be avoided. Creating a task decomposition allows for minimizing unnecessary steps that get between the user and their goal.
3. Data to store for users
    a. The amount of data needed for each user should be kept to a minimum. At least a username and password should be stored to allow users to sign in and use their accounts. Beyond that, any extra information asked for is another barrier to obtaining new users. The goal should be to reduce unnecessary data to give users ease of mind as to what personal data they are giving out.
    b. Data stored for students:
        i. Name
        ii. Email address
        iii. School name
    c. Data stored for professors:
        i. Name
        ii. Email address
        iii. School name
        iv. Area of study
4. Connection methods
    a. This project requires passing some amount of data between the frontend and the backend. The methods chosen determine what protocols are available. For personal information, higher levels of security should be used when passing user data back and forth between the server and the website.
    b. This project uses http, https, and websockets for connections. These protocols allow for information to be passed between front and back ends securely and in real-time.
5. Frameworks to use
    a. The frameworks used dictate the methods and functions available for programming the website. Certain frameworks provide libraries that make certain functions quicker and easier to create.
    b. React is the framework used for the frontend. This provides the ability to use JavaScript to give the website interactivity.
    c. Springboot is the framework used for the backend. This makes creating and connecting to a database easy.
6. Feature Selection

a. We have strategically incorporated the best elements and features from the existing LMS platforms while addressing where they fall short and struggle.

   We have decided to incorporate:

b. Anonymous questioning capabilities from Piazza
c. Interactive classroom activities from Top Hat
d. Student analytic features from Blackboard but focusing on a better user interface.
e. Overall centralization and organization implementing a better user experience.

4.2.2 Ideation

Page layout was the design decision with the most variation and choice for how to implement. Other decisions were based on the skill and previous experience of the team developing the project while deciding what to put on a page was much broader. When designing the page layout, many factors must be considered, such as readability of text, useful links in the toolbar, an order of information that makes sense to the user, and many more visual aspects.

Market research revealed several options for page layouts. Seeing what competitors decided was important for their layout provided a base upon which discussions were held to see what should be included in this project. Discussions about why these things are included and how they could be improved defined many of the design decisions made.

Adding a navigation bar at the top of the page was decided upon for quick access to all the significant features of the website. There are links for jumping to the Home, Live, Courses, Chat, and Profile pages in the navigation bar. These will be the most frequently used pages, and providing a way to always jump to those pages was deemed necessary.

Another decision had to be made on the general content layout of the pages. This had many options for different circumstances. The webpage could have a two column design where one is used for text and the other for pictures or videos. There could be “blocks” of information where formatting the text to highlight headers and section titles similar to chapters breaking up a book. Multiple choices for colors can create different moods in users when using the website. Bright colors are more cheerful and playful, while darker black and white pages make the text the center point of the screen.

The final decision was made to use layers in designing the webpages. This is how many similar websites are laid out where there are rows of similar information, and each row is a section focused on one topic. There can be any number of columns within each section and varying this was a good way to make the webpage feel more lively. Some sections have a picture on the left and text on the right, others are swapped. No matter what the configuration of the layer is, all information within it is related.

4.2.3 Decision-Making and Trade-Off

The decision-making process was kept quite simple to allow for quick and agile development. Anytime a question arose for a design aspect of the project, the team would vote for what they deemed the best option. A list of pros and cons was discussed, allowing all perspectives to be heard. Team members were allowed to give their opinions and suggest alternatives. Having a team that is open to criticism allows this strategy to be effective.

4.3 Design Problems

4.3.1 Overview

SmartClass is a web-based platform designed to help students and instructors interact more easily in large lecture halls. In larger classes, it is often more challenging for students to ask questions or for instructors to know who is having trouble understanding the lecture. SmartClass fixes that by giving everyone simple tools to participate in real-time.

Students can:

- Answer live quizzes or polls during class
- Ask questions anytime (even anonymously) using text, images, audio, or video
- Reply to each other and instructors in an open discussion space

Instructors and TAs can:

- Post quizzes or discussion prompts to check understanding
- Respond to student questions
- See participation statistics to help with grading and course planning

All conversations are saved and organized by topic (like homework or lectures), so students can go back and review them anytime. In the future, SmartClass can also be turned into a mobile app, so it is easy to use on the go.

The goal is to make learning more interactive and inclusive, giving every student a voice, no matter where they are sitting.

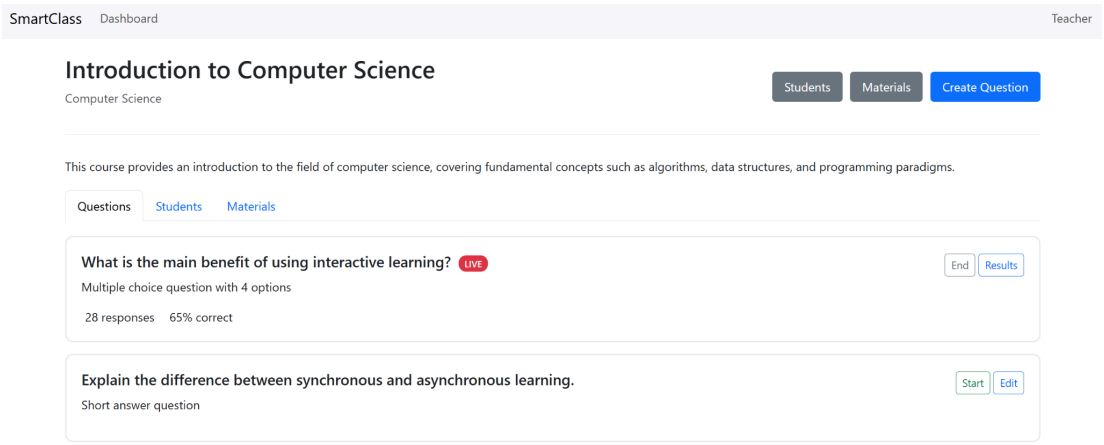
4.3.2 Detailed Designs and Visuals

SmartClass is a full-stack, role-based interactive classroom web platform designed to facilitate student engagement, especially in large lecture environments. It enables real-time Q&A, live quizzes, anonymous participation, media-rich discussions, and statistical tracking for performance and engagement.

The system consists of the following major components:

- Frontend (React.js + Websockets)
    - Role-based navigation for Student, Teacher, and TA accounts
    - Pages: Home, Class Dashboard, Forum, Activities, Archive
    - Dynamic updates for polls/questions using WebSockets
    - Potential Mobile-friendly accessible design in the future
- Backend (Spring Boot + Rest APIs)
    - Serves as middleware for database access and logic control
    - Manages authentication, quiz logic, question handling, and user roles
    - Handles media file uploads and secure data storage
- Database (MySQL)

    - Stores user credentials, class rosters, Q&A threads, quiz results, participation logs, and uploaded files
    - Indexed for filtering/search performance
- WebSocket Server
    - Enables real-time communication for:
        - Live poll updates
        - Forum interaction
        - Class-wide announcements



4.3.3 Functionality

SmartClass is designed to enhance the classroom experience for the students and the teachers. Here is how the system would function during a typical school day:

Before Class:

1. The teacher logs into SmartClass and creates their course (or if created access it again)
2. They prepare their lecture materials, including quizzes, slideshows, and other interactive elements.
3. Students then access the course page to review previous materials and prepare for class

During Class:

1. The teacher starts the lecture session and activates attendance tracking
2. Students log in and a displayed code is given to them in class and they input the code to be marked present.
3. As the lecture progresses the teacher can:
    a. Present slides and annotate them in real-time
    b. Launch quick polls to guage understanding
    c. Post quiz questions to get immediate feedback and understanding
4. Students can:
    a. View Lecture materials on their devices
    b. Respond to the polls and quizzes in class
    c. Submit questions to group/class discussion or to the teacher themselves.
    d. Take notes and comment on lectures directly on the application

After Class:

1. All lecture materials, this includes annotations and interactions are automatically saved.
2. Students can review materials and continue discussions.
3. Teachers can analyze participation and quiz data to look for areas of improvement.
4. More resources can be posted based on identified knowledge gaps.

4.3.4 Areas of Concern and Development

The current designs envisioned for SmartClass are strongly aligned with both user needs and system requirements. The proposed features directly target the core problems laced in large classroom settings, particularly the difficulty of inclusive participation, real-time feedback, and accessible learning tools.

Key areas where the design meets user needs:

- Inclusive Interaction: The anonymous Q&A and live chat tools address the needs of shy or hesitant students who want to engage without speaking up
- Real-Time Feedback: Live polls and quizzes allow instructors to quickly assess class comprehension, supporting their need to adapt teaching on the fly.
- Centralized Resources: The design includes archived questions, lecture materials, and discussion threads, helping students revisit material even if they miss class.
- Role-Based Structure: Custom views and permissions for teachers, students, and TAs help streamline workflows and responsibilities.
- Future Scalability and Accessibility: The team is considering mobile-friendly design and accessibility features such as TTS and minimal hardware requirements, making the platform usable by a wide range of students.

Primary Concerns Moving Forward:

1. Scope Creep vs Core Focus
    a. The team has many ideas but needs to prioritize what's the most essential.
    b. Concern: Trying to implement too much at once may slow progress or dilute core values.
2. User Adoption and Usability
    a. Since no prototype has been tested yet, it is unclear how intuitive the UI will be.
    b. Concern: Will students and instructors understand and embrace the platform during live lectures without training or confusion?
3. Real-Time System Complexity
    a. Real-time features like WebSocket-based quizzes and chats are conceptually solid, but complex to implement.
    b. Concern: Achieving low latency and stable performance in a large class environment will require careful design and testing.

Immediate Plans to Address These Concerns

- Develop Wireframes and Mock UIs: Begin with static designs for all major views (login, forum, quiz view, etc.) to guide frontend development and gather feedback before coding.

- Focus on Important Core: Finalize which 2–3 features are essential for initial testing (e.g., Q&A forum, anonymous questions, and quizzes), and limit scope to just those for now.
- Build a Clickable Frontend Prototype: Use tools like Figma or early-stage React components to simulate workflows and run usability tests with students and instructors.
- Backend Planning: Continue building REST API endpoints and defining the database schema to support core data types like users, classes, questions, and responses.
- Early Testing Strategy: Plan small usability and feedback tests even on unfinished components, to refine UX early. Use past examples like ColorWorks' diverse user testing as inspiration .

Open Questions for Clients, TAs, and Faculty Advisors

- For Instructors (Clients):
    - Which 2–3 features would you most like to test live in class?
    - Would you prefer to moderate questions live, after class, or not at all?
    - Are any institutional integrations (Canvas, grade export) important now or only later?
- For TAs:
    - What tools would help you manage student questions more easily?
    - Would you use the system if it replaced current email or office hour Q&A?
- For Faculty Advisors:
    - Do you foresee any concerns with our role-based permissions or anonymous posting?
    - Should accessibility testing (e.g., screen reader support) be prioritized earlier?

4.4 Technology Considerations

The SmartClass system design leverages a modern full-stack web development approach. Each chosen technology reflects a trade-off between scalability, ease of use, performance, and team familiarity. Below is an overview of the technologies being used, why they were selected, and what alternatives were considered.

Frontend (React.js)

- Strengths:
    - Highly modular and component-based, making it easy to build reusable UI elements.
    - Strong environment with support for state management (e.g., React Context, Redux), form validation, routing, and testing libraries.
    - Community support and rapid development cycle.
- Weaknesses:
    - Steeper learning curve for new developers compared to traditional HTML/CSS.
    - Requires setup with bundlers (Webpack, Vite) and tooling to be production-ready.
    - Can be overkill for very simple views.
- Trade-offs:
    - Chosen for flexibility and scalability, even though setup and initial development require more overhead than static HTML/CSS.
- Alternatives Considered:
    - Static HTML/CSS with JS: Simpler but not scalable for real-time, dynamic interactions needed in SmartClass.

Backend (String Boot)

- Strengths:
    - Robust and well-documented framework for RESTful APIs.

    - Strong support for secure authentication, form validation, and database access via JPA.
    - Scalable for enterprise-grade applications.
- Weaknesses:
    - Verbose syntax compared to more modern frameworks like Node.js/Express or Python's FastAPI.
    - Slower startup time during development.
    - More complex deployment pipeline for beginners.
- Trade-offs:
    - Chosen for its maturity and built-in features for security, scalability, and structure, despite being more heavyweight.
- Alternatives Considered:
    - Node.js with Express: Lighter and faster to prototype, but less structured and potentially messier at scale.

Database (MySQL)

- Strengths:
    - A well-supported relational database that integrates smoothly with Spring Boot via JPA and Hibernate.
    - Ideal for structured data like user roles, quizzes, and posts.
- Weaknesses:
    - Schema changes require migrations, which can slow iteration during early prototyping.
    - Doesn't handle unstructured data (e.g. media metadata) as flexibly as NoSQL solutions.
- Trade-offs:
    - Chosen for structure and ACID compliance, which is important for handling user data, class records, and participation logs.
- Alternatives Considered:
    - MongoDB (NoSQL): More flexible, easier to work with media or JSON-like documents, but lacks relational integrity.
    - PostgreSQL: Considered, but MySQL was chosen due to easier hosting and broader familiarity among developers.

WebSockets

- Strengths:
    - Enables real-time push-based communication (ideal for polls, chats, and live Q&A).
    - Better performance than frequent REST polling.
- Weaknesses:
    - Harder to debug and test than traditional HTTP endpoints.
    - Introduces complexity in connection handling and user session tracking.
- Trade-offs:
    - Chosen for interactivity; however, fallback mechanisms like long-polling may be implemented if needed for poor network conditions.
- Alternatives Considered:
    - Polling via REST: Simpler but less efficient.
    - SignalR (Microsoft): Considered but not compatible with Java backend.

Testing Tools (Jest, React Testing Library, Mockito, Postman, CI/CD)

- Jest (Frontend Unit Testing)
    - Strengths: Fast and popular JavaScript testing framework; integrates well with React Testing Library

    - Weaknesses: Limited to frontend/unit scope (doesn't cover integration or user flows)
    - Use Case: Testing UI logic, component rendering, props behavior, and simulated user actions
- React Testing Library Strengths:
    - Encourages testing from the user's perspective
    - Weaknesses: More verbose than traditional shallow testing
    - Use Case: Testing component accessibility and realistic interactions

- Mockito (Backend Unit Testing)
    - Strengths: Allows mocking of service layers and testing business logic in isolation
    - Weaknesses: Requires setup of mock data and test scaffolding
- Postman
    - Strengths: Easy to manually test REST endpoints, debug headers, and payloads
    - Weaknesses: Manual testing unless combined with Newman (for automation)
- CI/CD Pipeline
    - Tools Considered: GitHub Actions or GitLab CI
    - Purpose: Automate linting, unit testing, and deployment processes
    - Benefit: Prevent regressions and ensure production-readiness by catching errors earlier

4.5 Design Analysis

As of now, the SmartClass team has made solid progress on both the frontend and backend, laying the foundation for key system functionality. While the full product isn't yet operational, several important components are partially implemented and aligned with the original design goals outlined in section 4.3.

What Has Been Built So Far:

- A website built with React which has a home, about us, and sign-up for account pages.
- A database for storing user login and account information