Pickify

Alli Garcia

Nabou Diouf

Abstract

- **Pickify** is a web-based collaborative platform for real-time polling, task assignment, and decision analytics.
- Users create polls with multiple choice, word cloud or Q&A, share via unique links, and vote in real-time.
- Instant result viewing and live updates foster interactive decision-making.
- User authentication ensures privacy; only registered participants can create polls.
- With an intuitive interface, Pickify is ideal for team projects, social gatherings, and organizational decisions.

User Stories

- **Poll Creation**

As a user, I want to create polls with customizable options so that I can adapt them to specific needs.

- **Voting and Engagement:**

As a participant, I want to vote on polls and view live results so that I can see group preferences in real time.

As a participant, I want to provide comments on a poll so that I can discuss options and clarify choices.

- **Analytics and Reporting:**

As a poll creator, I want to generate detailed reports of poll results and participation metrics so that I can share findings with my team.

As a user, I want to view visual analytics (e.g., bar charts, pie charts) so that I can easily interpret poll results.

Functional Requirements and Modules

- **Authentication Module**

Allow users to register securely with encrypted credentials. Ensure session management to maintain user login states.

- **Poll Creation and Management Module**

Enable users to create polls with titles, descriptions, and options.

- **Voting and Real-Time Updates Module**

Allow participants to cast votes securely and view live results. Ensure real-time updates using WebSocket for instant feedback.

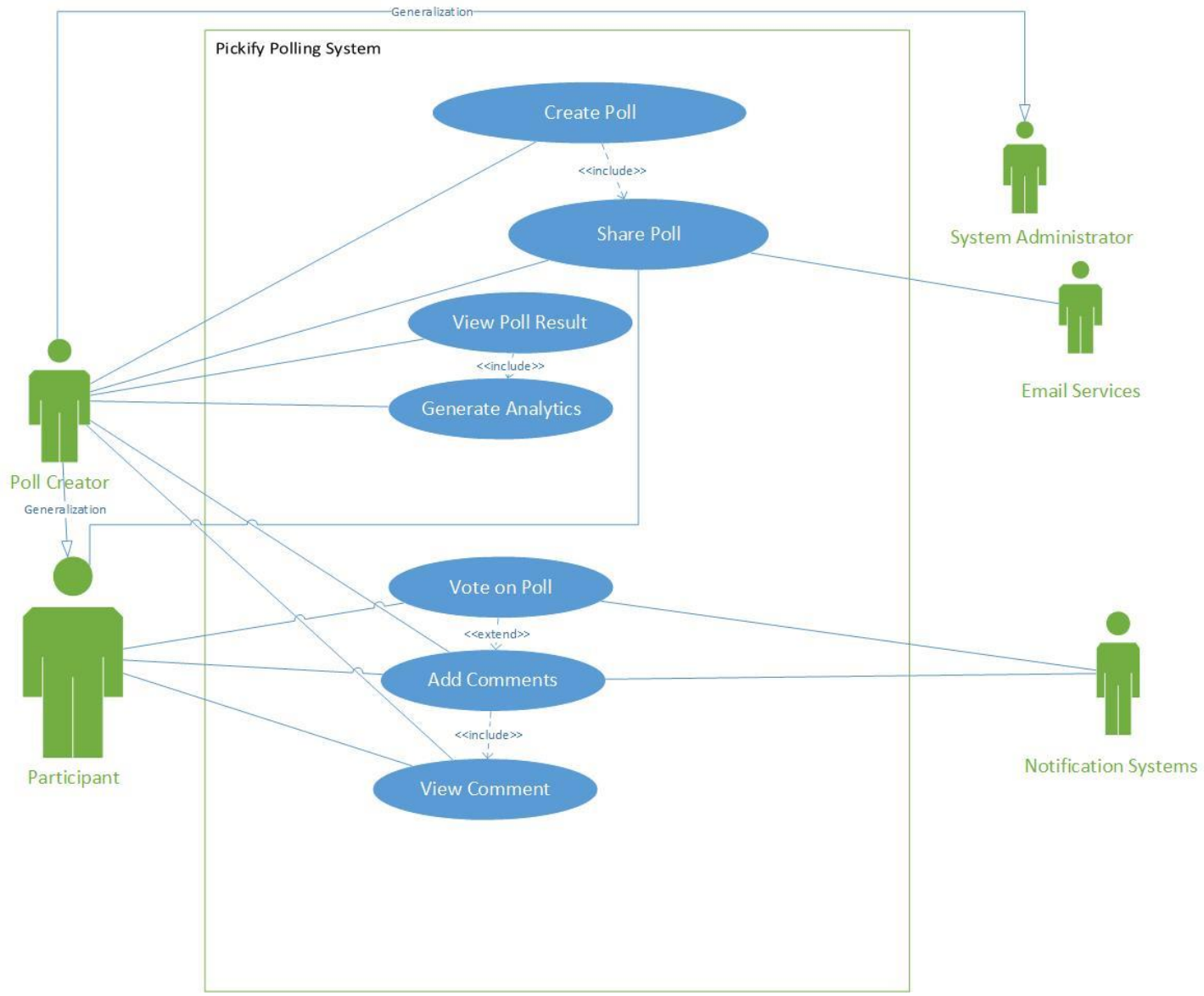
- **Comments and Feedback Module**

Allow participants to add comments to add comments to polls in real time.

- **Analytics and Results**

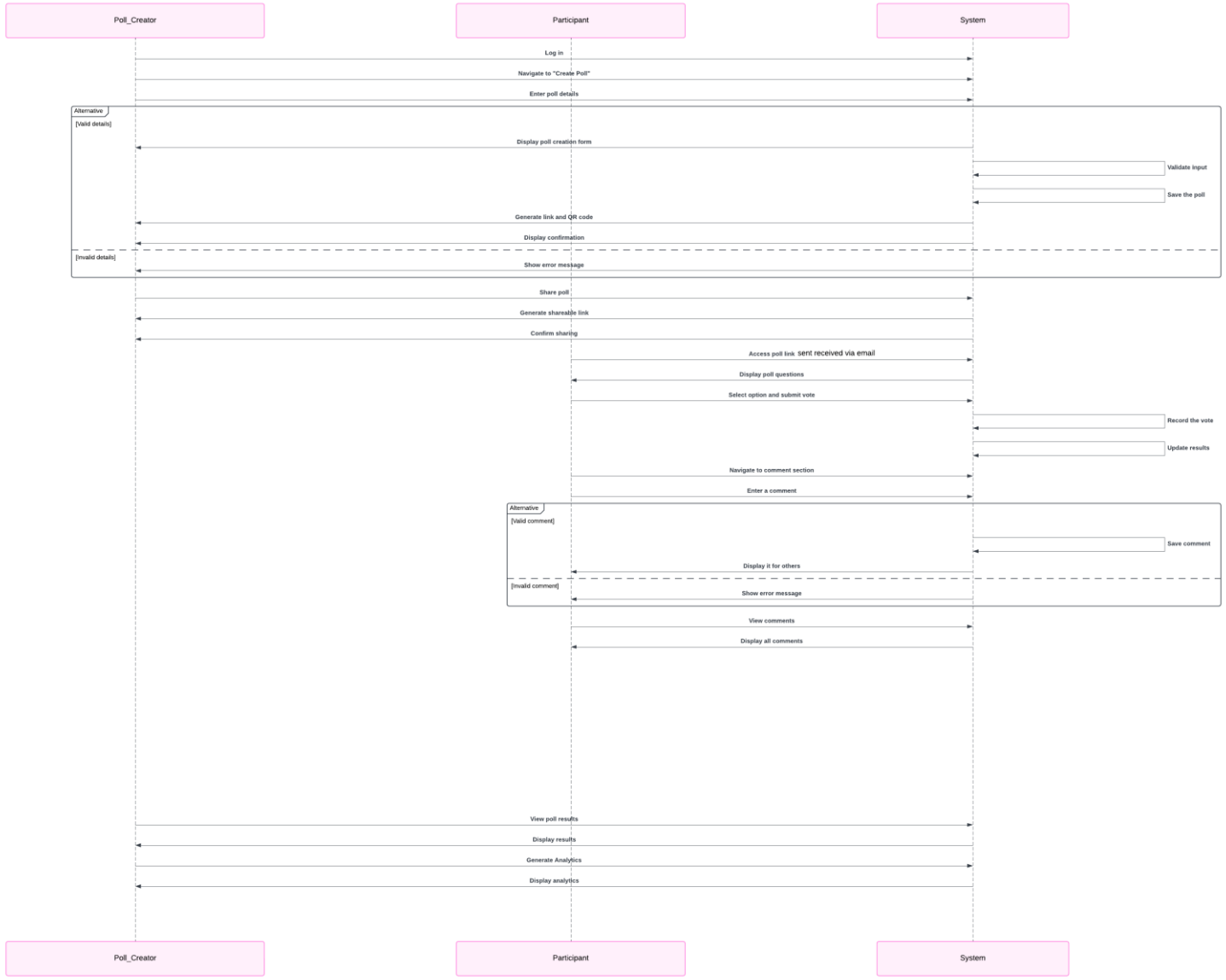
Create visual analytics for better interpretation of results. Log and track user participation metrics. Provide unique links for sharing polls with participants.

Use Case Diagram

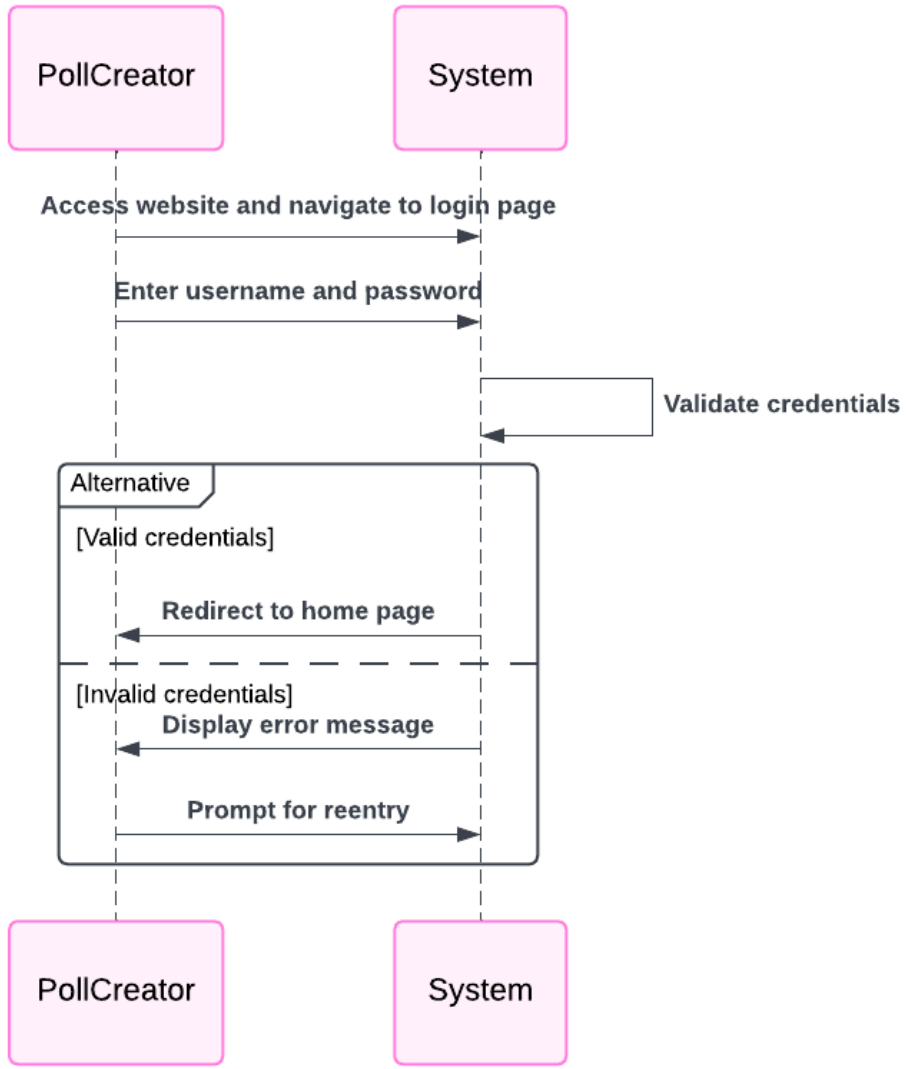


Activity Diagram

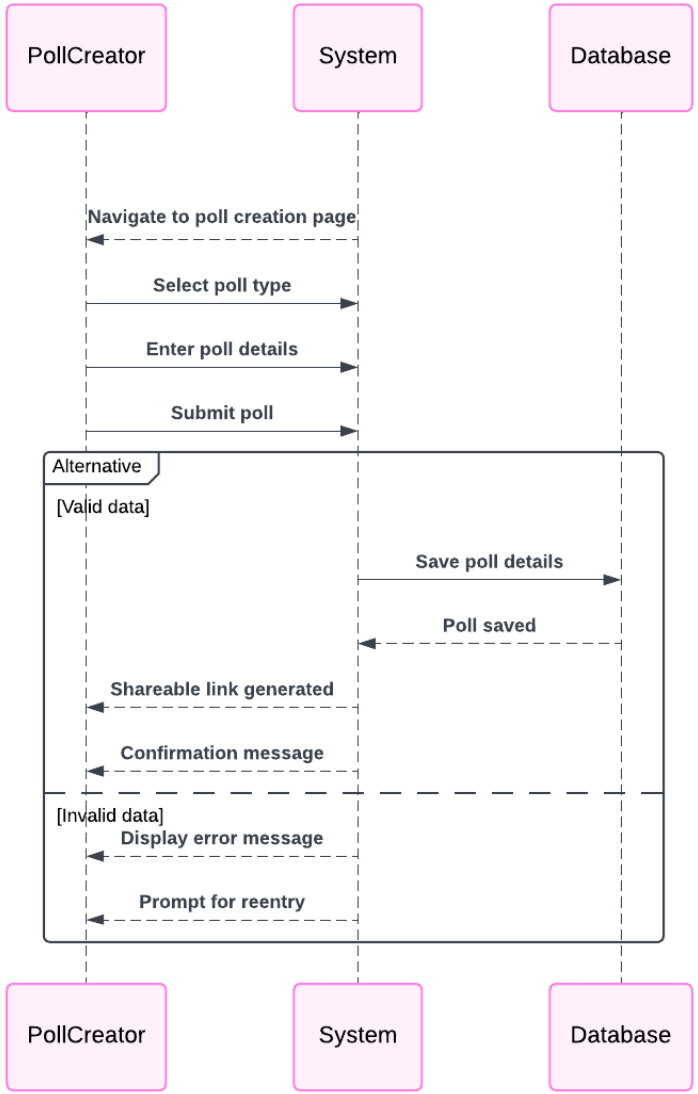
Swimlane Diagram



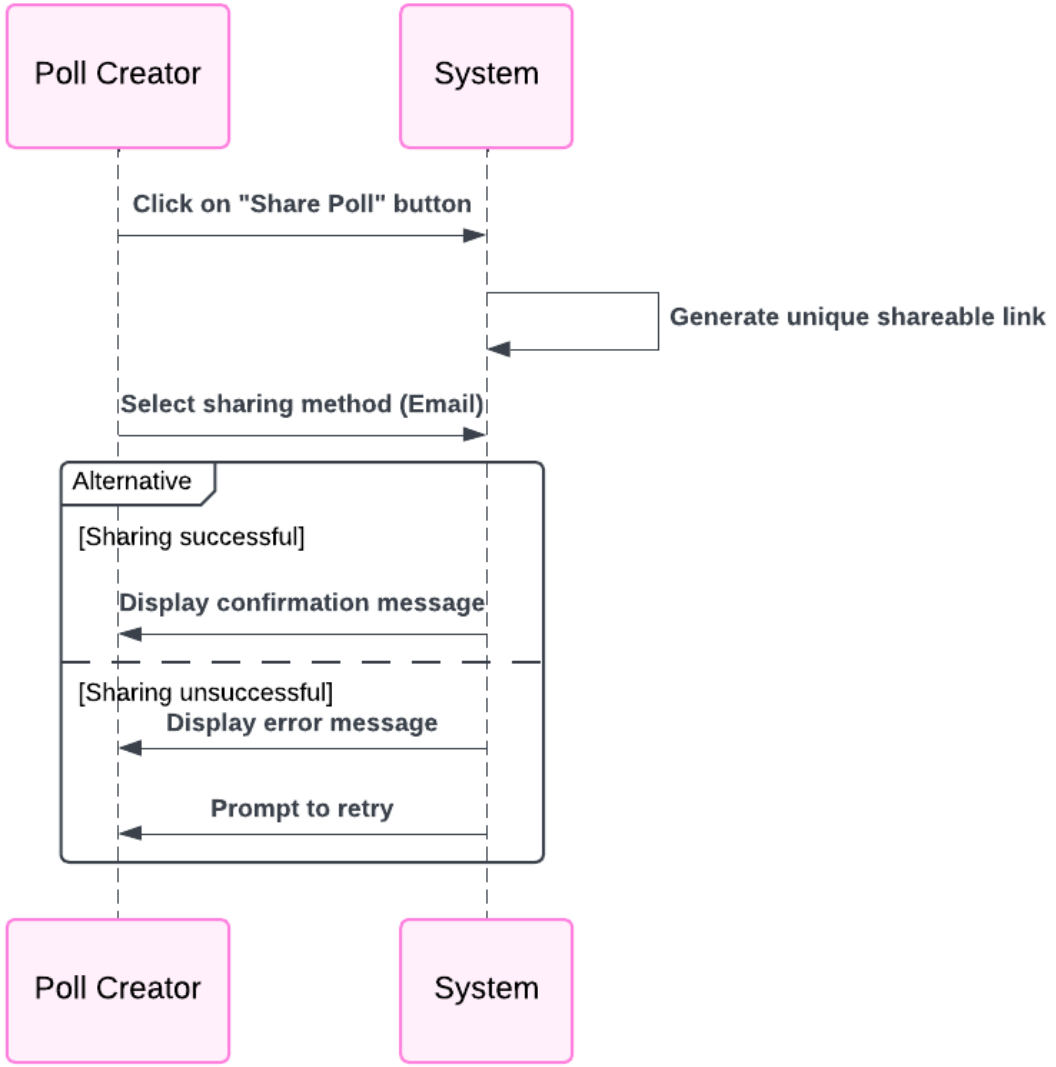
Sequence Diagrams (Login)



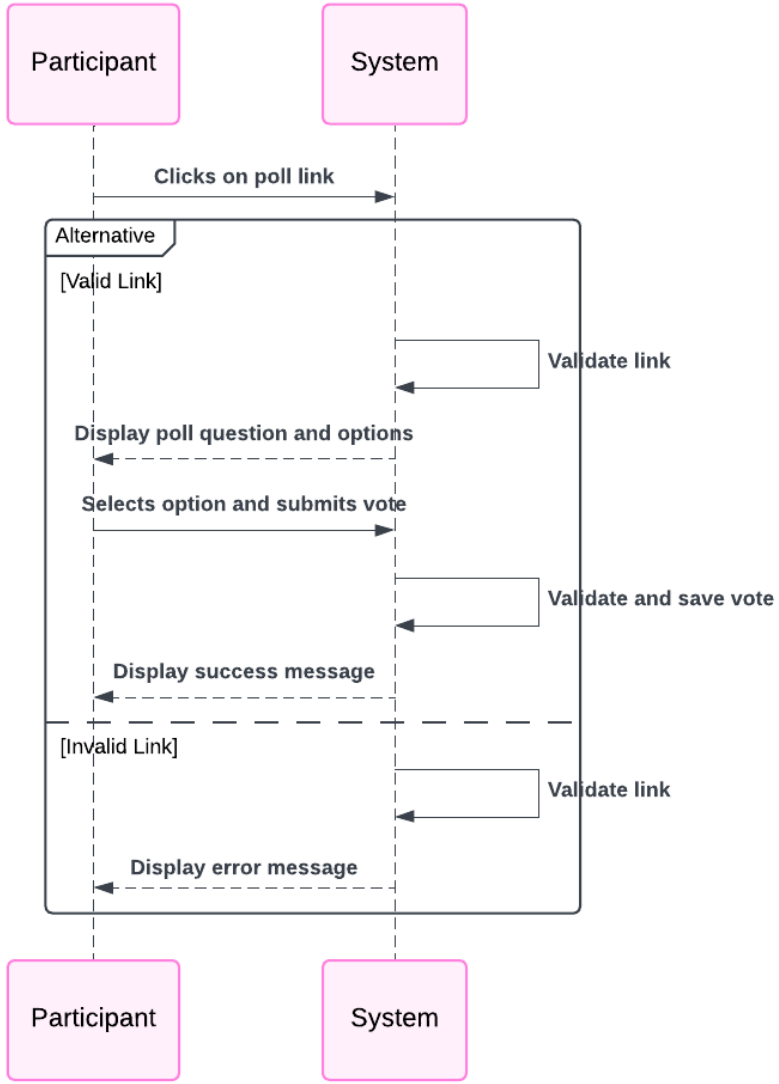
Sequence Diagram (Poll Creation)



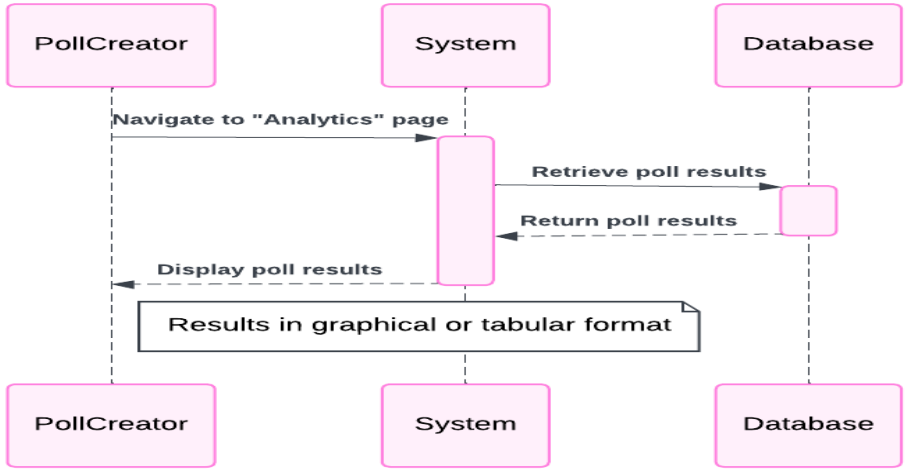
Sequence Diagram (Poll Sharing)



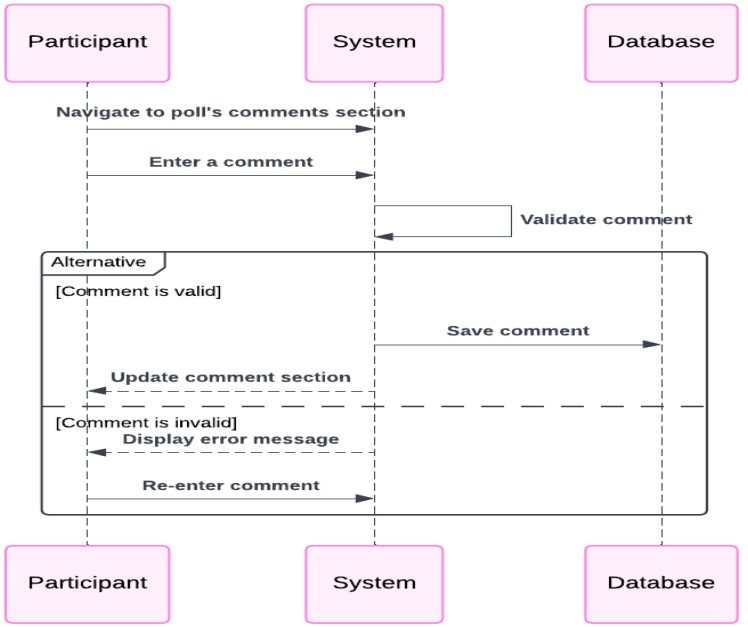
Sequence Diagram (Poll Participation)



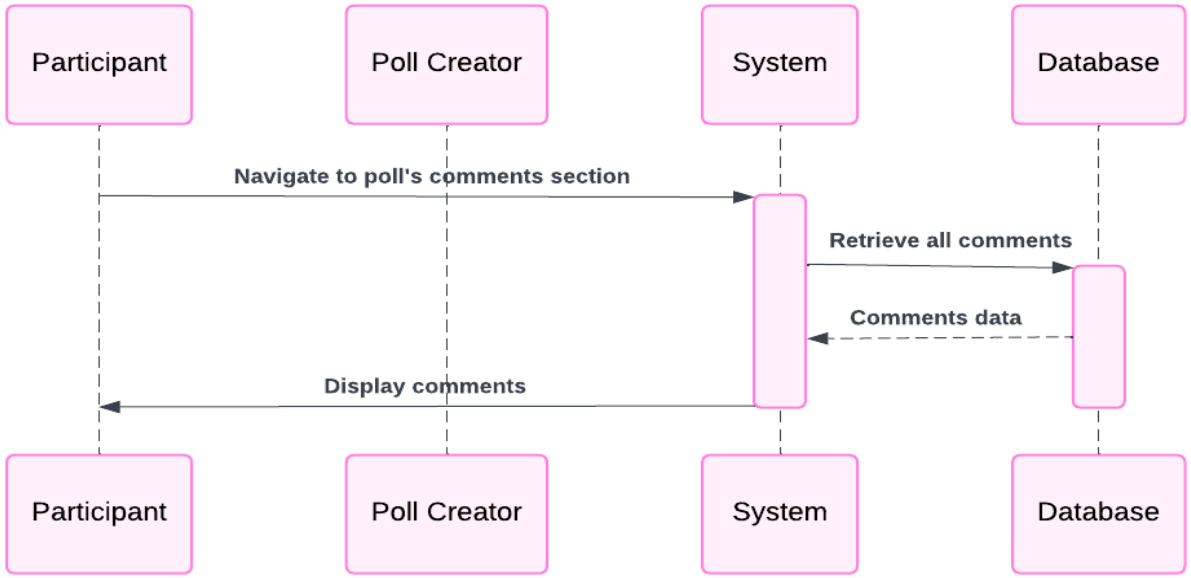
Sequence Diagram (View Results)



Sequence Diagram (Make a comment)



Sequence Diagram (View Comments)



Code Smells

1.Long Functions

- Functions like `view_dashboard` and `submit_question` were overloaded with multiple responsibilities, making them harder to read and maintain.

2. Duplicate Code

- Redundant logic was found in both `poll_controller.py` and `analytics_controller.py` for handling poll retrieval and authorization.

3. Inconsistent Authorization Handling

- Functions inconsistently handled user authentication, leading to repeated token-parsing logic.

4. Lack of Separation of Concerns

- Business logic (e.g., vote counting) was tightly coupled with controller logic.

5. Hardcoded Strings

- Repeated hardcoded error messages and URL paths reduced flexibility.

6. Complex Conditionals

- Nested `if` conditions in endpoints like `vote` and `submit_question` made it difficult to follow the logic.

7. Database Over-Reliance

- Direct database operations (e.g., MongoDB queries) spread throughout the code without an abstraction layer.

Refactorings

1. **Function Extraction**

- - Broke down long functions like `view_dashboard` into smaller helper functions to handle tasks such as `calculate_analytics`, `fetch_feedback`, and `authorize_user`.

- - Example: `calculate_analytics` now processes Q&A and multiple-choice analytics separately.

2. Centralized Authentication

- - Standardized the `get_current_user` logic into a reusable helper function, reducing redundancy across multiple files.

3. Encapsulation of Business Logic

- - Moved vote and question-related logic into dedicated functions (e.g., `process_vote`, `add_question`).

- - Example: `submit_question` now offloads question-creation logic to a helper function.

4. Introduction of Constants/Config

- - Replaced hardcoded strings like `"multiple_choice"` and error messages with constants for better maintainability.

5. Simplification of Conditionals

- - Replaced complex `if-else` blocks with dictionary-based mappings where appropriate (e.g., poll type handling in `view_dashboard`).

6. Redirection Logic Standardization

- - Unified the redirection pattern for dashboard views after voting, answering, and submitting questions.

Design Patterns

- **Factory Pattern**

- - Applied for handling different poll types (e.g., `multiple_choice`, `q_and_a`).

- - Example: A factory method determines the analytics logic to use based on poll type, avoiding hardcoded checks in multiple places.

- **Template Method**

- - Used for the `view_dashboard` function.

- - A base template now handles common elements of the dashboard (e.g., fetching the poll, authorization), while specific analytics (e.g., vote data, Q&A data) are handled by overridden methods.

- **Decorator Pattern**

- - Used to wrap endpoints with authorization and authentication checks via `Depends(get_current_user)`.

- - Example: All endpoints requiring user context leverage a consistent authorization layer.

- **Repository Pattern (partially implemented)**

- - Started abstracting MongoDB operations (e.g., `polls_collection.find_one`) into repository functions to decouple data access from business logic.

Softwares

- Backend Framework and Web Server
- - FastAPI: For building the backend API and web application.
- - Hypercorn: ASGI server to run the FastAPI application.
- Security and Authentication
- - bcrypt: For password hashing.
- - python-jose: For handling JSON Web Tokens (JWT).
- - passlib[bcrypt]: A comprehensive password hashing library.
- Database and ORM
- - MongoDB Atlas: A cloud-based database service used to host your MongoDB database.
- - MongoDB Compass: A GUI tool to manage and explore the MongoDB database.
- - motor: An async driver for MongoDB, used to interact programmatically with the database.
- - pymongo: A MongoDB driver for Python, providing additional database management capabilities.
- Templating and File Handling
- - Jinja2: For rendering HTML templates.
- - aiofiles: For asynchronous file handling.

Softwares 2

- Testing
- - pytest: For writing and running test cases.
- - pytest-asyncio: For testing asynchronous code.
- Utilities
- - python-dotenv: For loading environment variables from `.env` files.
- - python-multipart**: For handling multipart form data (e.g., file uploads).
- - logging: For logging errors, warnings, and system activities to debug and monitor the application effectively.
- Networking
- - httpx: For making HTTP requests.
- - websockets: For handling WebSocket communication.
- Data Validation
- - pydantic[email]: For data validation and handling email types.
- Data Visualization
- - matplotlib: For creating visualizations such as charts and graphs.
- Email Services
- - Custom Email Service: Integrated functionality for sending emails to participants, including invitations and notifications.
- Deployment
- - Render: A cloud platform used to deploy the application and host your website.

WEBSITE DEMO

THANK YOU!