SQR Voting System - Project Report

Project Overview

The SQR Voting System is a web-based platform designed for creating, managing, and participating in surveys and polls. The system allows users to register, log in, create polls with single or multiple-choice answers, vote in polls, and view survey results. Authorized users can also close polls at any time or set a specific closing date for a poll.

Team Members

- **Egor Nisckikh** - Team Leader, Backend Developer
- **Almaz Gayazov** - Backend Developer, Tester
- **Renata Latypova** - Backend Developer, Tester
- **Artemij Volkonitin** - Tester
- **Julia Martynova** - Frontend Developer

Project Setup and Installation

Prerequisites

- Python 3.11+
- Poetry
- SQLite

Installation

1. Clone the repository:

```
git clone https://github.com/Fridorovich/voting.git
```

2. Install dependencies:

```
poetry install
```

3. Create a `.env` file with the following content:

```
DATABASE_URL=sqlite:///./sqr_voting.db
SECRET_KEY=your-secret-key
ALGORITHM=HS256
ACCESS_TOKEN_EXPIRE_MINUTES=30
REFRESH_TOKEN_EXPIRE_DAYS=7
```

4. Apply database migrations:

```
alembic upgrade head
```

5. Run the server:

```
uvicorn app.main:app --reload
```

6. Access documentation at:

```
http://127.0.0.1:8000/docs
```

Technical Stack

- **Python 3.11**
- **FastAPI** - Web framework
- **SQLite** - Database
- **Poetry** - Dependency management
- **Alembic** - Database migrations
- **JWT** - Authentication
- **Pytest** - Testing framework
- **Coverage.py** - Test coverage analysis
- **Flake8, Bandit** - Code quality and security analysis
- **Streamlit** - Frontend
- **Docker, Docker Compose** - Containerization and orchestration

Implementation Details

Modules

- **Authentication Module:** User registration, login, token generation, and refresh mechanisms.
- **Voting Module:** Poll creation, voting, and result viewing.
- **Administration Module:** Poll management, closing polls, and setting poll expiration dates.

Key Features

- User registration and authentication using JWT tokens.
- Poll creation with single/multiple choice options.
- Voting functionality with single-use voting and the ability to change votes.
- Poll closing by the creator or automatically based on set dates.
- Poll result viewing with real-time updates.
- Robust error handling and logging for all actions.

Continuous Integration (CI)

The CI pipeline is implemented using GitHub Actions. The pipeline includes the following stages:

- **Setup:** The environment is configured to use Python 3.11.
- **Dependency Installation:** Poetry is installed and dependencies are managed in an isolated environment.
- **Code Quality Analysis:**
  - `flake8` is run to check code style compliance with PEP8.
  - `bandit` is used for security analysis to identify potential vulnerabilities.
- **Testing:**
  - All tests are executed using `pytest` with detailed output enabled.
  - Test coverage is assessed with `coverage.py`. The coverage report is generated in HTML format and currently shows **89%** coverage.
- **Dockerization:**
  - If the branch is `main`, the Docker image is built and pushed to Docker Hub using `docker/build-push-action`.

Docker Containerization

- The application is containerized using Docker and orchestrated with Docker Compose.
- The `backend` service is built from the Dockerfile and exposed on port **8000**.
- The application applies database migrations on startup using Alembic.
- Data persistence is handled through volume mounts for the SQLite database.

Docker Commands:

- Build the image:

```
docker-compose build
```

- Run the container:

```
docker-compose up
```

- Stop the container:

```
docker-compose down
```

Poetry Configuration

- The project is managed using Poetry, a robust tool for dependency management and packaging.
- The `pyproject.toml` file defines dependencies in two groups:
  - `main`: Production dependencies such as `fastapi`, `sqlalchemy`, `uvicorn`.
  - `dev`: Development tools like `flake8`, `bandit`, `pytest-cov`.
- To install dependencies:

```
poetry install --with dev
```

Logging Implementation

The logging system is implemented using Python's built-in `logging` module. The logging configuration is initialized in the `setup_logging()` function, located in the `app/shared/logging.py` file.

Logging Configuration:

- **Log Directory:** Logs are stored in the `logs` directory. If the directory does not exist, it is automatically created.
- **Log File:** The main log file is `sqr_voting_system.log`.
- **Log Rotation:**
  - Maximum file size: 10 MB
  - Backup count: 5 log files are retained before the oldest logs are removed.
- **Log Format:** The log format includes the timestamp, log level, module name, and the log message. Example:
  `[2025-05-08 14:23:56] [INFO] [app.modules.auth.services] User authenticated: id=1`
- **Log Levels:** The system logs events at the following levels: `INFO`, `WARNING`, `ERROR`, `CRITICAL`.

Testing

- The testing strategy includes unit testing, integration testing, and security testing.
- All tests are located in the `app/tests/` directory and are structured by module (e.g., `auth`, `voting`).

Test Coverage Analysis

- Total test coverage: **89%**
- Key coverage results:
  - `app/modules/auth/routes.py`: 100%

- `app/modules/auth/services.py`: 100%
- `app/modules/voting/routes.py`: 67%
- `app/modules/voting/services.py`: 70%
- `app/modules/admin/routes.py`: 54%
- `app/modules/admin/routes.py`: 98%

Performance and Reliability Analysis

- **Maintainability Index:** A (calculated using `radon mi`). All files in the project received a rating of `A`, indicating a high level of maintainability.
- **Recovery Time (MTTR):** 0.296 seconds (measured using docker-compose up -d), significantly below the target of 15 minutes.
- **Performance Analysis:**
  - **Poll creation:** Average response time - **2 ms**, Min - **1 ms**, Max - **4 ms** (target ≤ 500 ms)
  - **Voting:** Average response time - **2 ms**, Min - **0 ms**, Max - **4 ms** (target ≤ 300 ms)
  - **Result retrieval:** Average response time - **2 ms**, Min - **0 ms**, Max - **9 ms** (target ≤ 1 second)

Security

- Passwords are hashed using JWT and bcrypt.
- Protection against SQL Injection and XSS.
- All actions are logged for accountability.

Lessons Learned

- Implementing structured testing and coverage analysis significantly reduces potential bugs.
- Automated CI pipelines streamline deployment and testing.
- Comprehensive Dockerization ensures consistency across environments.

Future Improvements

- Increase test coverage for the `voting` and `admin` modules to align with the 100% goal achieved in the `auth` module.
- Implement additional security checks with Bandit for more comprehensive analysis.
- Expand the frontend to provide a more interactive user experience.

Accessibility

- **Public access**: https://voting-inno.ru or http://79.174.93.194:8501/

References

- FastAPI Documentation
- Poetry Documentation
- pytest Documentation
- Docker Documentation

- GitHub Actions
- Coverage.py
- Bandit