**VIETNAM NATIONAL UNIVERSITY – HCMC**

**INTERNATIONAL UNIVERSITY**

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

# WEB APPLICATION DEVELOPMENT

IT093IU

TECHNICAL REPORT

Course by Assoc. Prof. Nguyen Van Sinh

Lab Instructed by MSc. Nguyen Trung Nghia

**TOPIC:**

**SAMPLE TEST ARCHIVER WEBSITE**

BY DATABRICKS

# Table Of Contents

# I. INTRODUCTION

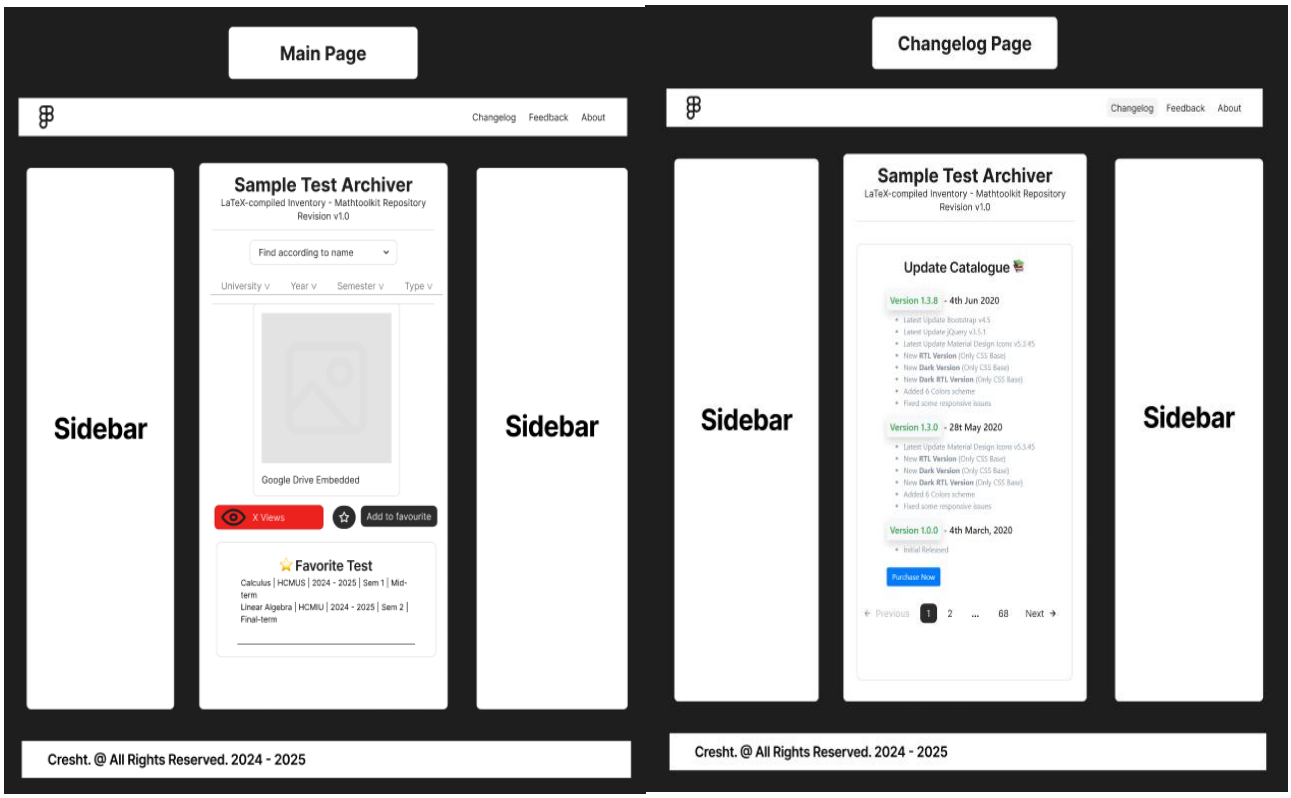
Sample Test Archiver originates from the Mathtoolkit repository with the aim of providing students from various academic backgrounds access to mathematical sample tests retrieved from diverse online forums and group communities. The project was motivated by the complexity of navigating GitHub repositories, especially for non-technical users whose primary objective is to locate, view and download relevant materials. In order to mitigate the challenges, a simplified and intuitive interface in the website allows users to search, filter, and access tests through familiar functionality such as Google Drive embedding and image viewers, thereby significantly improving accessibility.

# II. REQUIREMENT ANALYSIS & DESIGN

## 1) Inspiration

The inspiration behind Sample Test Archiver is the need to navigate university-sample materials from the solid lightweight platform that prioritizes ease of access and quality visibility. Rather than customary features such as user authentication, this system concentrates on seeking useful academic resources through ratings and feedback, which enables students to identify high-quality tests. This vision also aligns with the prominent website – IUExamHub, serving core functionality on large-scale systems.

**Initial Layout Design**




## 2) Functional Requirements

The system provides subject-based autocomplete search mechanics and supports multi-level filtering by university, year, semester and type. Users are able to browse a structured list of sample tests stored under descriptive metadata and reveal the content by using an embedded viewer. Each test view is tracked through session-based counters to measure frequency. Additionally, users can tag or untag favourite tests with preferences stored client-side for permanent use. The platform displays average ratings for each test derived from submitted feedback records, particularly users are allowed to evaluate with specific star rating from nil to

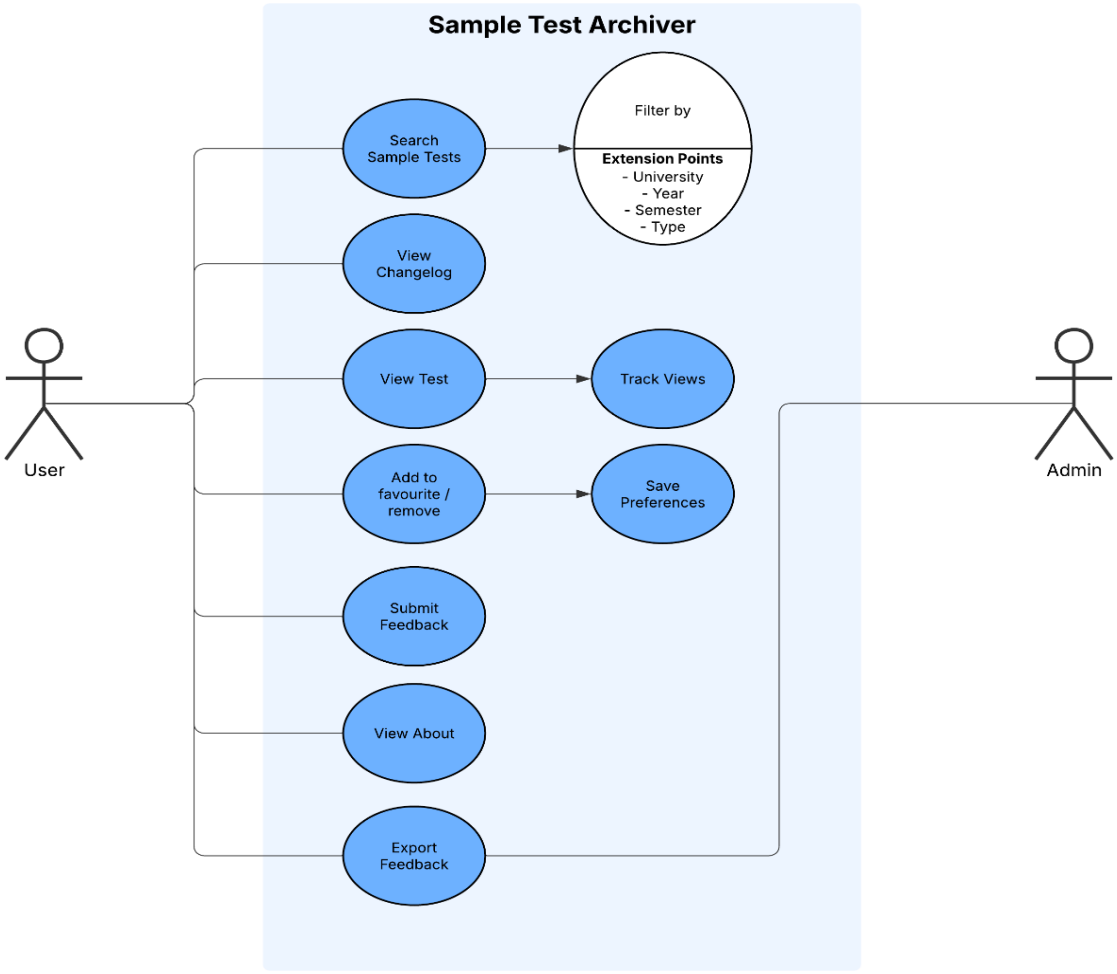
five. A changelog section is included to preserve historical updates, alongside static pages such as Home, Abouts.

### 3) Non-functional Requirements

From a non-functional perspective, this system is dedicated to being responsive across devices using available frameworks and optimized to deliver fast API responses. Database interactions are secured through parameterized queries to alleviate the risk of cyberattack. Maintainability is ensured in route modules and separated controller, concurrently portability is achieved using environment-based configuration files. Observability is centralized via console logging for debugging and monitoring purposes.

### 4) Use Case Diagram

In the general workflow, a student searches and filters subjects or tests, views selected tests, marks them as favourites, observes average ratings, submits feedback, reviews the changelog for updates and biography. An implicit administrative role exists within the system, which is responsible for seeding and maintaining subject data, sample tests, and changelog entries in the database, although this role does not directly interact with the user interface.



- Actors

| *Actor* | *Description* |
|---|---|
| *User* | A student or general user who searches, views and interacts with sample tests |
| *Admin* | Administrator who manages sample test collection and exports feedback |

- Sample Use Case Table

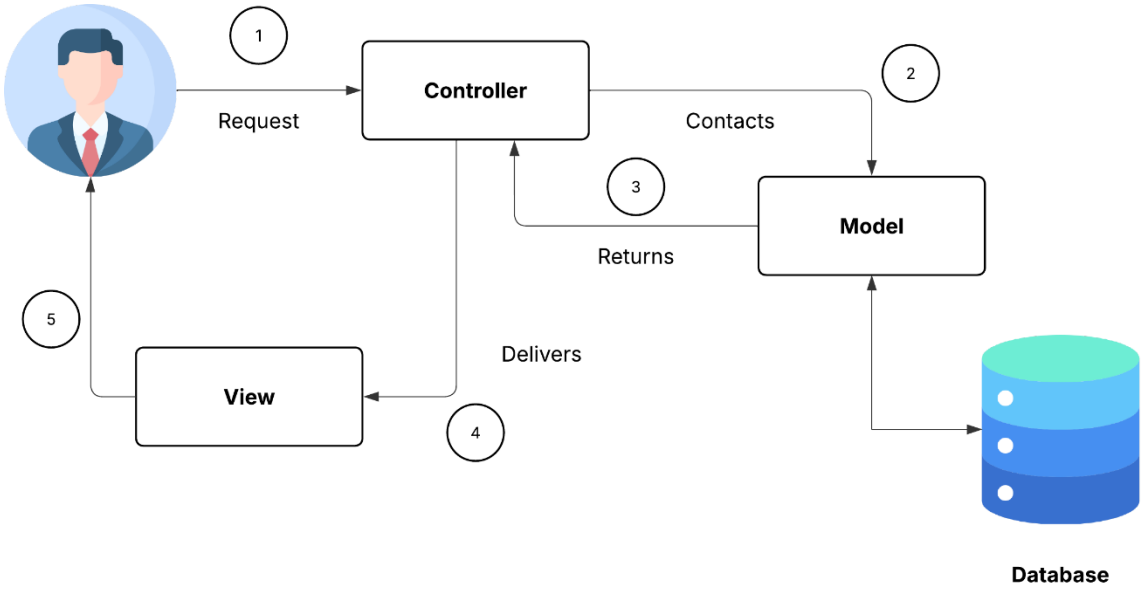
| *UC No.* | *Name* | *Actor* | *Input* | *Output* |
|---|---|---|---|---|
| *UC1* | Search Sample Tests | User | Search keywords | List of matching sample tests |
| *UC2* | Filter Sample Tests | User | University, Year, Semester, Type | Filtered test list |
| *UC3* | View Test | User | Selected test | Test content displayed |
| *UC4* | View Changelog | User | Changelog request | Displayed changelog |
| *UC5* | Track Views | Admin | Test view activity | View statistics |
| *UC6* | Add to Favourite / Remove | User | Favourite action | Updated favourite list |
| *UC7* | Submit Feedback | User | Feedback text | Feedback stored |
| *UC8* | View About | User | About request | About page displayed |
| *UC9* | Export Feedback | Admin | Export request | Downloaded feedback export |

- Pre-condition
  - The Sample Test Archiver system is online and accessible
  - Users have access to the system interface
  - Admin is authenticated (by interacting with deployment) with administrative privileges.
- Post-condition
  - Search and filter results are displayed to the user
  - Selected tests are successfully viewed
  - User favourites are updated and saved
  - Feedback is stored securely in the system
  - Admin can view manual analytics and export feedback data
  - View tracking data is saved into the database.

- User Stories
    - As a user, I want to search for sample tests so that I can quickly find relevant materials.
    - As a user, I want to filter tests by university, year, and semester so that results match my course.
    - As a user, I want to view a test so that I can prepare for exams.
    - As a user, I want to add tests to my favourites so that I can easily access them later.
    - As a user, I want to submit feedback so that I can help improve the system.
    - As an admin, I want to track test views so that I can analyze usage.
    - As an admin, I want to export feedback so that I can review user suggestions offline.

## 5) MVC Diagram

The system follows a Model – View – Controller architecture. The Model layer consists of metadata tables such as subjects, tests, feedback, test ratings, and changelog entries, accessed through a database utility module. The View layer is implemented using static files located in the client directory, providing the user interface and client-side logic. The Controller layer is built using routes that handle application logic for tests, subjects, feedback and changelog functionality, with overall application initialization managed by the main server entry point.



# III. TOOLS & TECHNIQUES

## 1) Front-end

The front-end of the application is developed using standard web technologies including HTML and CSS, with Bootstrap used for responsive layout and styling. Font Awesome is utilized for icons, while JavaScript handles client-side logic such as search interactions, filtering, favourites, and API communication.

## 2) Back-end

The back-end is implemented using Node.js with the Express.js framework to manage routing and API endpoints. Additional middleware is used to handle cross-origin requests, JSON parsing, and session-based functionality required for view tracking and state management.

## 3) Database

A MySQL relational database is used to store structured data such as subjects, sample tests, feedback records, ratings and changelog entries. This database design supports efficient querying and analytics while maintaining data integrity.

## 4) Deployment

The Express server is configured to serve both static client files and RESTful APIs from a single deployment. Application settings such as database credentials and server port are managed using environment variables for flexibility across environments. The project is deployed using Railway, enabling straight-forward cloud hosting and availability.

# IV. TASK ASSIGNMENTS

The project tasks are divided across front-end, back-end and database responsibilities. Front-end UI and styling are implemented through CSS files and HTML page layouts within the client directory. Core client-side logic for searching, filtering, favourites and ratings are handled in modular JavaScript files. The feedback feature is managed through a dedicated feedback page and script, while the changelog functionality is implemented using a separate page with pagination logic. Back-end APIs are developed using Express route handlers and initialized through the main server file. Database schema design, table creation, and seed data are managed through structured SQL scripts.

| *Member* | *Role* |
|---|---|
| *Trần Minh Đức* | Front-end & Back-end: Analysis, Project Structure (Website & Database), Search Bar Functionality, Sample Test Filters, Data Integration, Changelog & About Pages |
| *Đặng Hoàng Minh Long* | Front-end & Back-end: Feedback System, Add To Favourite, View Tracker |
| *All* | Deployment, Testing, Debugging |

# V. INSTALLATION & USE GUIDE

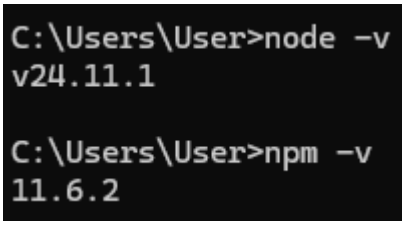
## 1) Prerequisites

To run the system locally, Node.js (LTS version) and npm must be installed, along with a running MySQL server.

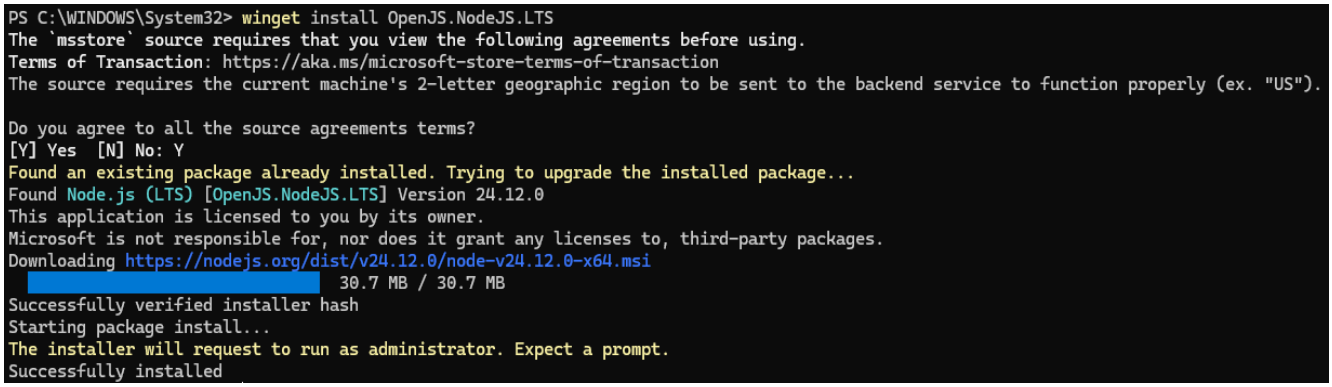
## 2) Installation Steps

The installation process begins with creating a MySQL database for the project and importing the finalized schema SQL file. Environment variables are then configured by copying the example environment file and updating database credentials and server settings as needed. After installing project dependencies using npm, the server can be started using Node.js or npm script if provided.

- Check if node.js and npm are installed using the following commands:




- If not installed, run the following commands in windows powershell:
  **winget install OpenJS.NodeJS.LTS**




(In this circumstance, node.js is already installed but it's outdated so after running the command, it'll automatically update to the latest version.)

- Open VSCode and locate the project folder manually (File → Open Folder → Select project location) or (Ctrl + O + K → select project location).
- Open the integrated terminal (Ctrl + ~) and paste the following commands: **npm install**
- **npm install express-session**

```
C:\Users\User\OneDrive\Documents\Study\Web Application Development project\sample-test-archiver-local\sample-test-archiver-local>npm install

added 81 packages, and audited 82 packages in 770ms

21 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

```
C:\Users\User\OneDrive\Documents\Study\Web Application Development project\sample-test-archiver-local\sample-test-archiver-local>npm install express-session

added 8 packages, and audited 90 packages in 1s

22 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

- Open MySQL then create a new connection and import the database script in the “db” folder -> create_tables_2.sql
- Click the thunder icon to establish the database:

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 1 | 02:25:14 | DROP DATABASE `sample_archiver` | 6 row(s) affected |
| 2 | 02:27:57 | CREATE DATABASE IF NOT EXISTS sample_archiver CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci | 1 row(s) affected |
| 3 | 02:27:57 | USE sample_archiver | 0 row(s) affected |
| 4 | 02:27:57 | CREATE TABLE subjects ( id INT AUTO_INCREMENT PRIMARY KEY, code VARCHAR(50) NOT NULL UNIQUE, name VARCHAR(255) NOT NULL... | 0 row(s) affected |
| 5 | 02:27:57 | CREATE TABLE tests ( id INT AUTO_INCREMENT PRIMARY KEY, -- FK → subject.id subject_id INT NOT NULL, university VARCHAR(100) NOT NUL... | 0 row(s) affected |
| 6 | 02:27:57 | CREATE TABLE IF NOT EXISTS changelog ( id INT AUTO_INCREMENT PRIMARY KEY, version VARCHAR(50) NOT NULL, date DATE, body T... | 0 row(s) affected |
| 7 | 02:27:57 | CREATE TABLE feedback ( id SERIAL PRIMARY KEY, user_fullname VARCHAR(120), email VARCHAR(120), dob DATE, problem_type VARCH... | 0 row(s) affected |
| 8 | 02:27:57 | CREATE TABLE test_ratings ( test_id INT PRIMARY KEY REFERENCES tests (id), avg_rating DECIMAL(3, 2) DEFAULT 0, total_reviews INT DEFAU... | 0 row(s) affected |
| 9 | 02:27:57 | CREATE TABLE feedback_export_queue ( id SERIAL PRIMARY KEY, feedback_id INT REFERENCES feedback (id), exported BOOLEAN DEFAULT... | 0 row(s) affected |
| 10 | 02:27:57 | CREATE INDEX idx_tests_subject ON tests (subject_id) | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 |
| 11 | 02:27:57 | CREATE INDEX idx_tests_filters ON tests(year, semester, type, university) | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 |
| 12 | 02:27:57 | CREATE INDEX idx_feedback_test_id ON feedback (test_id) | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 |
| 13 | 02:27:57 | CREATE INDEX idx_feedback_problem_type ON feedback (problem_type) | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 |
| 14 | 02:27:57 | CREATE INDEX idx_feedback_created_at ON feedback (created_at) | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 |
| 15 | 02:27:57 | CREATE INDEX idx_feedback_test_rating ON feedback (test_id, rating) | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 |
| 16 | 02:27:57 | CREATE INDEX idx_export_queue_exported ON feedback_export_queue (exported) | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 |
| 17 | 02:27:57 | CREATE INDEX idx_export_queue_feedback_id ON feedback_export_queue (feedback_id) | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 |
| 18 | 02:27:57 | CREATE INDEX idx_test_ratings_avg ON test_ratings (avg_rating) | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 |
| 19 | 02:27:57 | INSERT IGNORE INTO subjects (code, name) VALUES ('MATH101', 'Calculus I'), ('MATH102', 'Calculus II'), ('MATH201', 'Discrete Mathematics'), ('STAT201',... | 6 row(s) affected Records: 6 Duplicates: 0 Warnings: 0 |
| 20 | 02:27:57 | INSERT IGNORE INTO tests ( subject_id, university, year, semester, type, drive_embed_url ) VALUES (1, 'HCMIU', 2024, 'Final-term', 'Final', 'ht... | 22 row(s) affected Records: 22 Duplicates: 0 Warnings: 0 |

- Open VSCode, locate to server/db.js and change the username and password to your MySQL username and password

```
user: process.env.DB_USER || 'root',
password: process.env.DB_PASS || 'password'
```

- Locate to client/instructions/frmMain.java and do the same to username and password

```
101     String username = "root"; // Use your MySQL username
102     String password = "password"; // Use your MySQL password
```

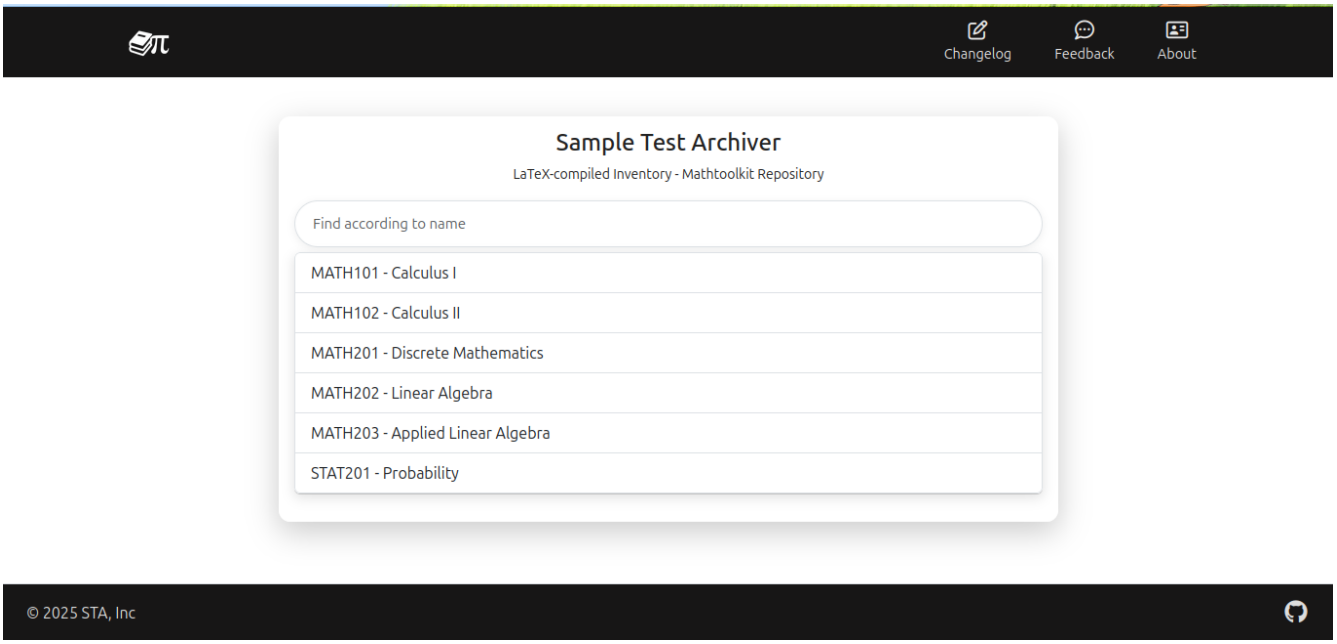
- After open the project in VSCode, open the integrated terminal and paste in the following command: **node server/server.js**
- Ctrl + left click on http://localhost:3000 to open the site

```
C:\Users\User\OneDrive\Documents\Study\Web Application Development project\sample-test-archiver-local\sample-test-archiver-local>node server/server.js
[dotenv@17.2.3] injecting env (0) from ..\.env -- tip: add observability to secrets: https://dotenvx.com/ops
[dotenv@17.2.3] injecting env (0) from .env -- tip: load multiple .env files with { path: ['.env.local', '.env'] }
Server listening on http://localhost:3000
MySQL Connected Successfully
```

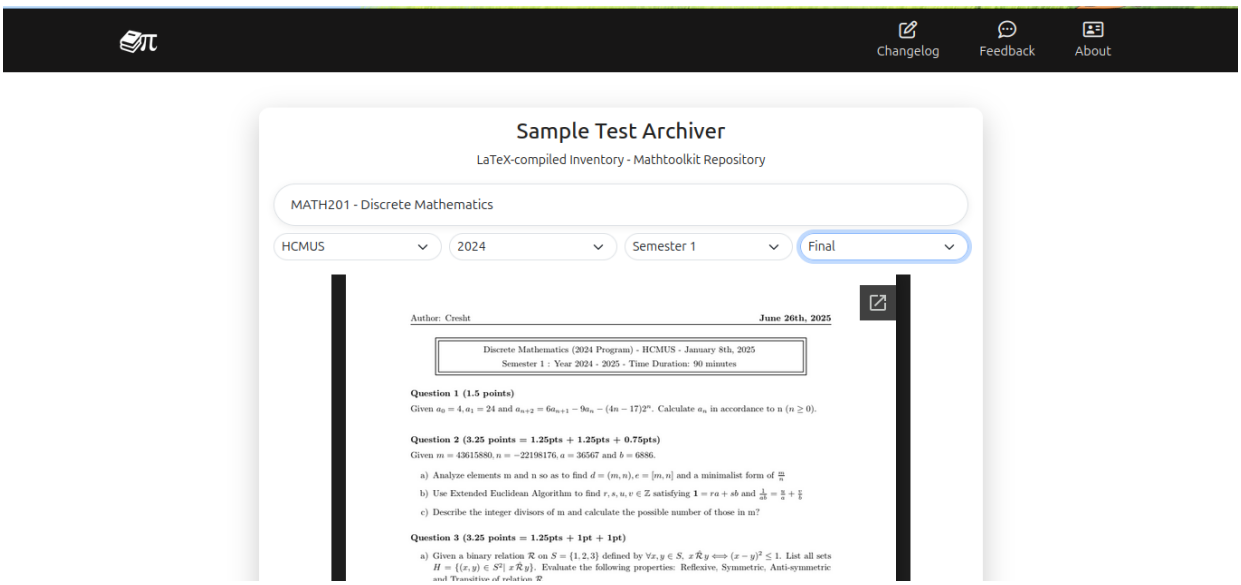
3) Usage

Once the server is running, users can access the local host through a web browser and use the search bar with autocomplete and filtering options to identify sample tests. Selected tests can be seen in embedded viewers, marked as favorites, and evaluated using average ratings from the number of user suggestions. Feedback can be submitted through the feedback page, changelog entries can be browsed with pagination, and additional project information is available on the About page.

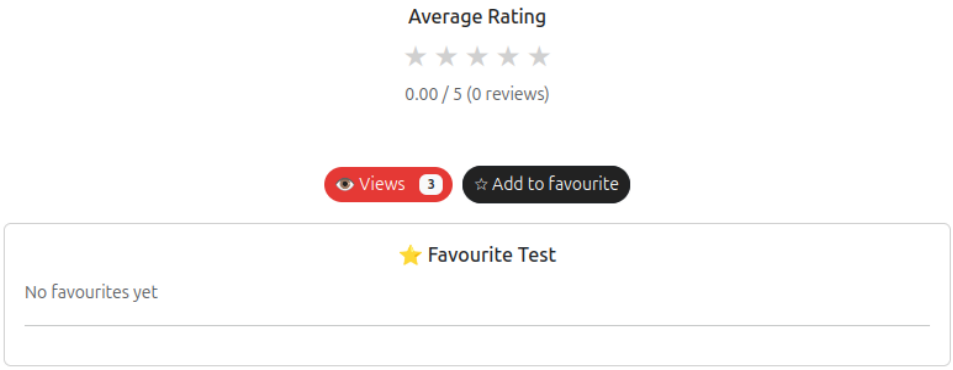
Mainpage of the website



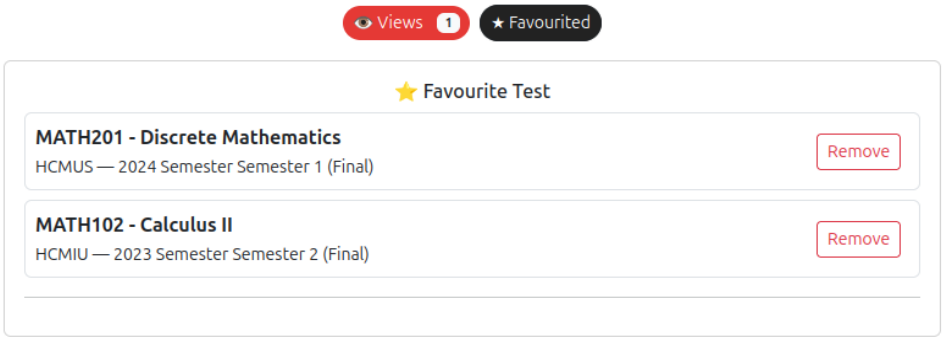

Type in any keywords and apply filters to identify sample tests



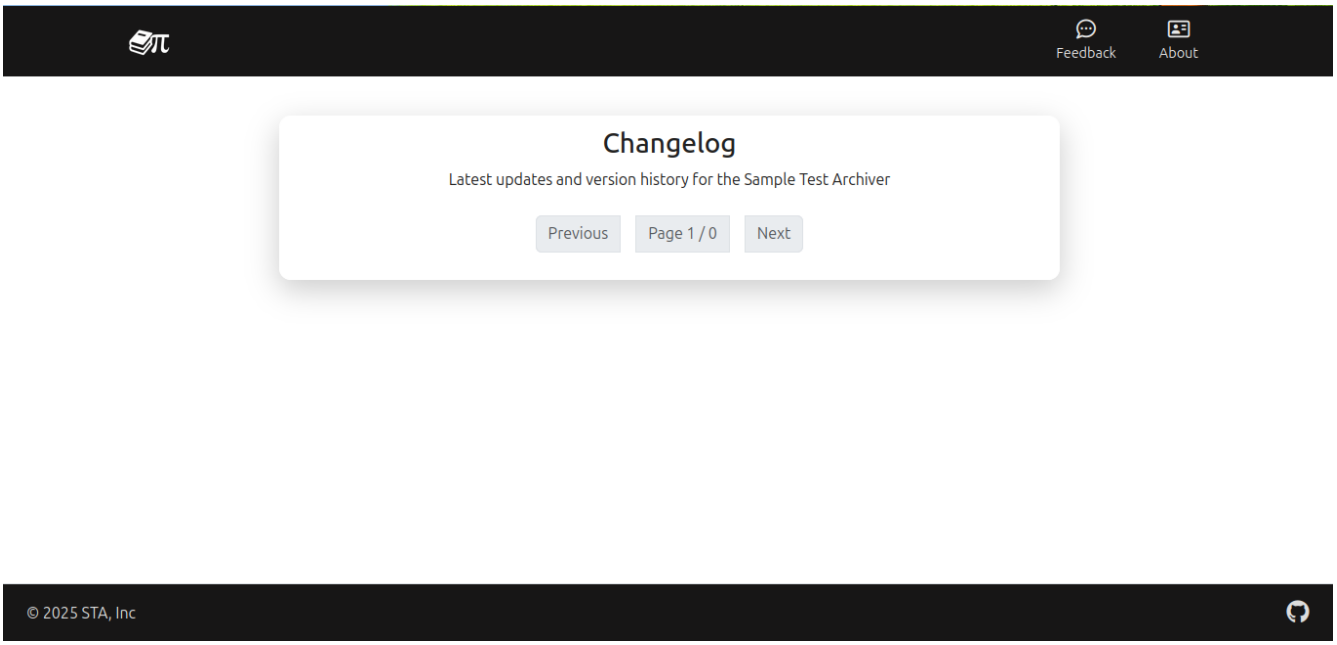

For each sample test being called, an average star rating appears below the viewer besides "X Views" and "Add To Favourite" sections. All actual values are called directly from the database (via APIs).



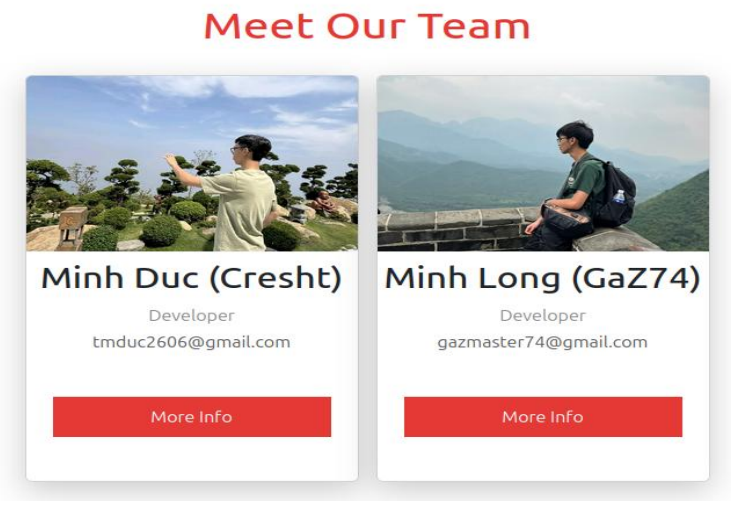

Regarding the favourite list, each test card is filled with specific filters and subject names. To save your desired tests for later revision, press "Add to favourite". Next time, it will be automatically saved on personal web browsers (localStorage).

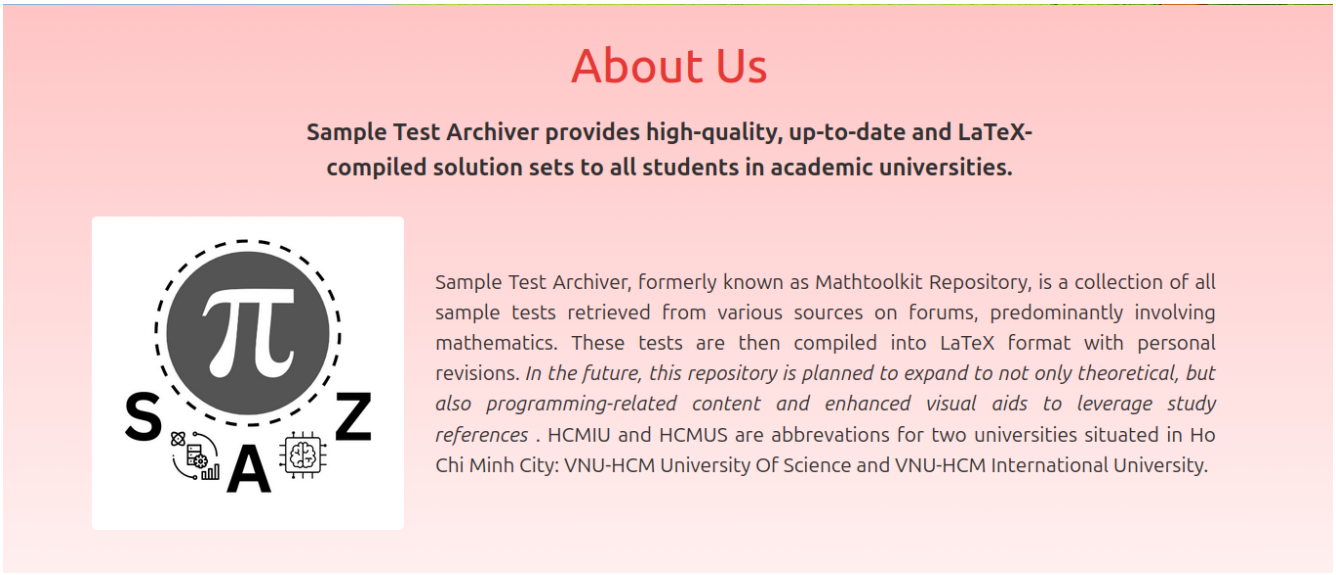


Changelog Page:




About Us Page:

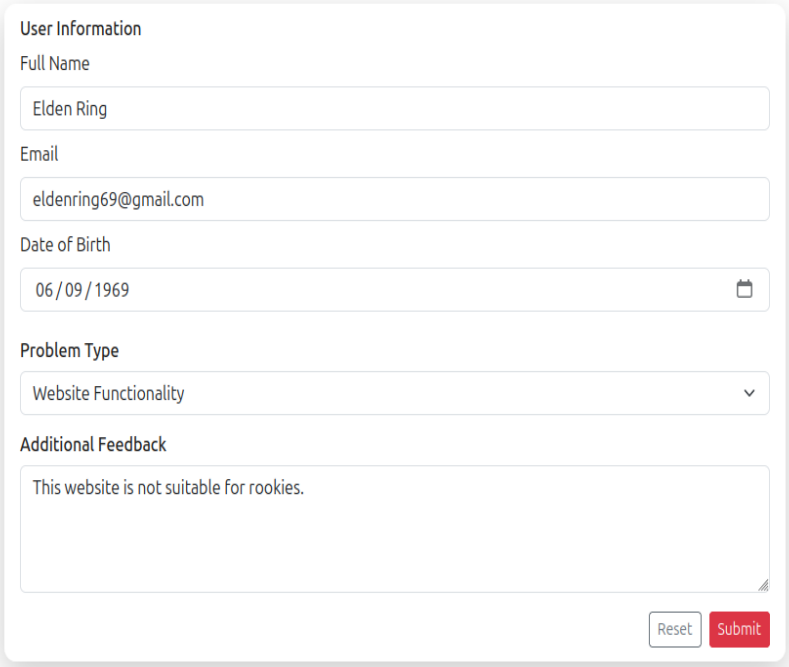




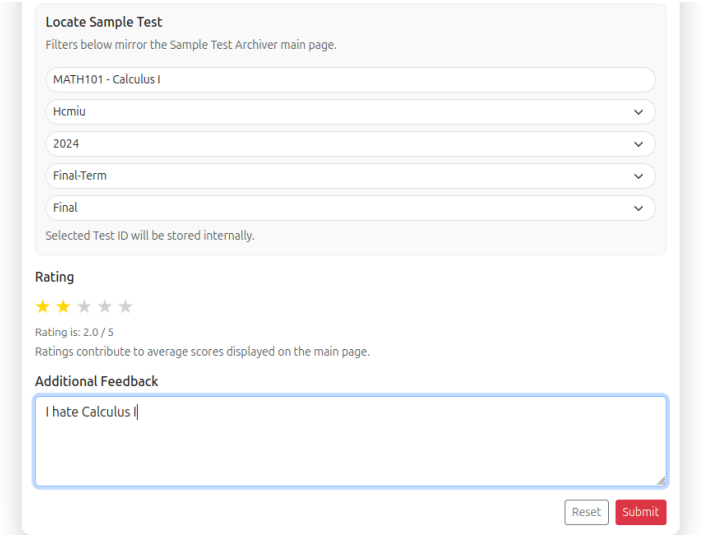

Feedback Mechanics - There are two types that users could make suggestions for upcoming improvements: Website Functionality & Sample Tests. In the Sample Tests section, the panel will pop up additional filters to address the desired one to give a particular score rate alongside additional description (acts similar to Website Functionality).

Fill in respective fields (Full Name, Email, D.O.B, Problem Type and Additional Feedback).

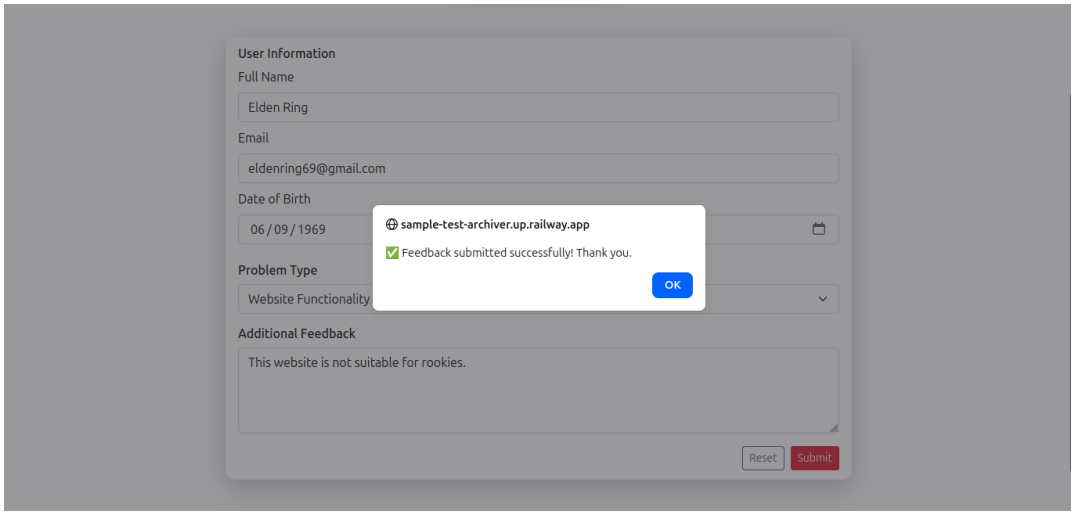
Website Functionality:



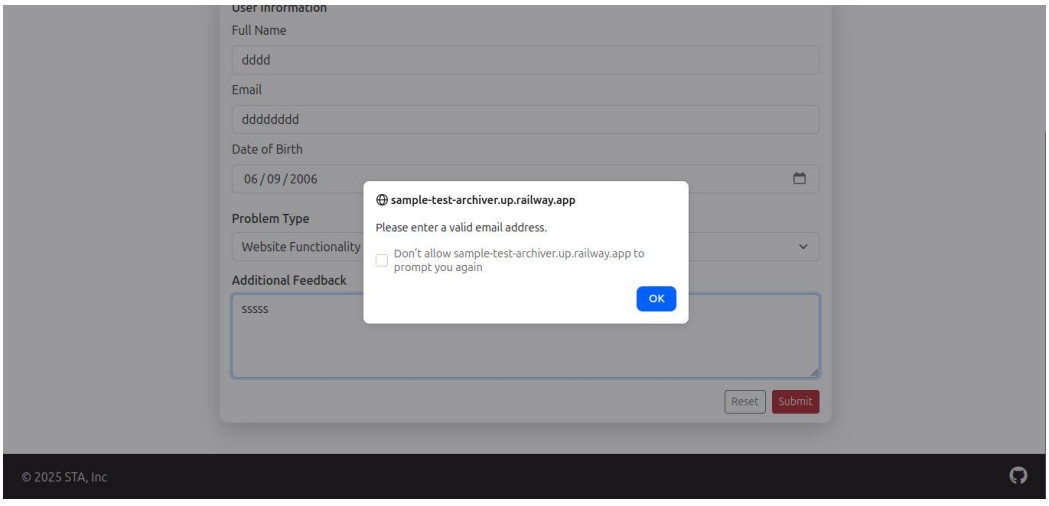

Sample Tests:




Successful Message:




Exception Handling (Validate Expression):



# VI. IMPLEMENTATION

## 1) Project File Structure

The project is organized in a clear separation between client-side, server-side, and database resources to support further scalability. The client directory contains all static assets, including HTML pages, CSS and JavaScript files responsible for layout and interaction logic. The server directory encapsulates the Express.js application, routing logic, and database utilities, while the database directory stores SQL scripts for schema creation, seeding, and queries. This modular structure enables independent development and testing of each layer while maintaining a rational architecture.

```
sample-test-archiver-local/
├── package.json                    # Node dependencies and scripts
├── client/                         # Front-end static assets
│   ├── index.html                  # Main search/browse interface
│   ├── feedback.html               # Feedback submission page
│   ├── changelog.html              # Version history display
│   ├── about.html                  # Project information
│   ├── gaz_cv.html                 # Developer CV page
│   ├── 404.html                    # Error page
│   ├── assets/
│   │   ├── js/
│   │   │   ├── main.js             # Core app logic (search, filters, favorites)
│   │   │   ├── api.js              # Centralized API fetch helpers
│   │   │   ├── utils.js            # Utilities (localStorage, session, debounce)
│   │   │   ├── feedback.js         # Feedback form logic and star ratings
│   │   │   └── changelog.js        # Changelog pagination
│   │   ├── styles/
│   │   │   ├── main.css            # Global styles and layout
│   │   │   ├── feedback.css        # Feedback page styling
│   │   │   ├── changelog.css       # Changelog table styling
│   │   │   └── about.css           # About page styling
│   │   └── images/                 # Static images
│   ├── backups/                    # Local backup storage
│   └── instructions/
│       ├── file structure.txt      # Project structure documentation
│       ├── Structure.txt           # Additional notes
│       └── frmMain.java            # Java Swing DB query tool
├── server/
│   ├── server.js                   # Express app entry point
│   ├── routes/
│   │   ├── tests.js                # Test retrieval, view tracking, ratings
│   │   ├── subjects.js             # Subject search and listing
│   │   ├── feedback.js             # Feedback submission and retrieval
│   │   └── changelog.js            # Changelog API
│   └── utils/
│       └── db.js                   # MySQL connection pool configuration
└── db/
    ├── create_table_final.sql      # Primary schema with seed data
    ├── create_tables.sql           # Alternative schema version
    ├── create_tables_2.sql         # Another schema iteration
    └── View total View query.sql   # Analytics query example
```

## 2) HTML – CSS Components

The HTML layer defines the structural layout and content of the application's pages. The index.html page serves as the primary interface, featuring a subject search bar with autocomplete, cascading filter controls for university, year, semester, and test type, a result grid displaying test cards with metadata, and an embedded viewer for content, alongside favourite and rating indicators. The feedback.html page provides a structured feedback form that includes a test

selection dropdown, a five-star rating widget, a comment input area, and submission confirmation. The changelog.html page presents system updates in a paginated table format with version, date, and description fields, while about.html displays project background information and contact details. A custom 404.html page is used to handle invalid routes.

The CSS layer customizes the visual presentation of the application. The main.css file defines global styles, Bootstrap overrides, navigation bar, header - footer layout, card designs, buttons and so on. The feedback.css file focuses on styling the star rating widget, form validation states, interactive animations, and feedback listings. The changelog.css file enhances table readability with alternating row colors, responsive wrappers, pagination controls, and version badges. The about.css file styles profile sections, informational layouts, and responsive elements for different screen sizes. Bootstrap 5 provides the foundational UI framework, offering responsive grids, components, and form controls, while Font Awesome supplies iconography. Custom CSS variables are used to ensure consistent theming and spacing across the application.

### 3) Client Components

The client-side implementation is apparent in index.html, which acts as the entry point for the main user experience by rendering the subject search interface, filter controls, result cards, and embedded viewer modal. This page loads core JavaScript modules, including api.js, utils.js, and main.js, along with global CSS assets. The main.js file controls the primary application logic by handling subject autocomplete, applying cascading filters, fetching test data, rendering results, opening embedded viewers, and managing favorite toggling. Client-side persistence of favorites is implemented using localStorage utilities provided by utils.js. All communication with the backend is abstracted through api.js, which standardizes REST API calls and error handling.

Shared utilities in utils.js manage localStorage and session-related helpers, as well as reusable UI functions such as debouncing and formatting. The feedback page logic is implemented through feedback.js, populating test selection options, capturing star ratings, submitting feedback to the backend, and optionally displaying feedback entries filtered by test. Similarly, the changelog.js script retrieves paginated changelog data from the server, renders it into a table, and connects pagination controls. Static informational pages such as about.html, gaz_cv.html, and 404.html present non-interactive content while sharing common styling rules to ensure visual consistency across the application.

### 4) Database Establishment

The database layer is implemented by using MySQL, with the primary schemas and seed data defined in the create_table_final.sql script. This schema includes core tables for subjects, tests, feedback, aggregated test ratings, changelog entries, and feedback export queue, with appropriate foreign keys and indexes to maintain referential integrity and query optimization. Earlier schema iterations are preserved in separate SQL files to document the evolution of the data model. An

additional query file provides an example of aggregating total view counts across tests. Database connectivity is managed through a centralized utility module that configures a connection pool and sources credentials from environment variables, ensuring consistent and secure access across all server routes.

## 5) Server Components

The server-side implementation is built on Express.js and is initialized in the main server entry file, which configures middleware, serves static client assets, mounts API routes, and starts the HTTP server. A shared database utility module establishes and manages the MySQL connection pool for use across all route handlers. The subjects API provides endpoints for listing and searching subjects to support autocomplete and filtering functionality. The tests API retrieves test data based on selected filters, keeps track of view counts on a per-session basis, and exposes rating aggregates for display. The feedback API accepts feedback submissions and retrieves feedback entries, optionally filtered by test. The changelog API delivers paginated changelog entries along with metadata needed for client-side pagination controls.

## 6) Deployment

The deployment process relies on environment-based configuration for flexibility during development. On startup, the server loads environment variables, determines the listening port, and confirms availability through console output. Middleware such as CORS handling, JSON and URL-encoded body parsing, cookie parsing, and session management is applied to support cross-origin requests, state management, and basic security. The Express server statically serves the client directory and routes unmatched paths to a custom 404 page. All API endpoints are exposed under formatted /api namespace and rely on the shared database pool. Deployment involves installing dependencies, configuring environment variables, ensuring database connectivity with the correct schema, and starting the server, with Railway used as the hosting platform.

# VII. DISCUSSION

### 1) Achievements

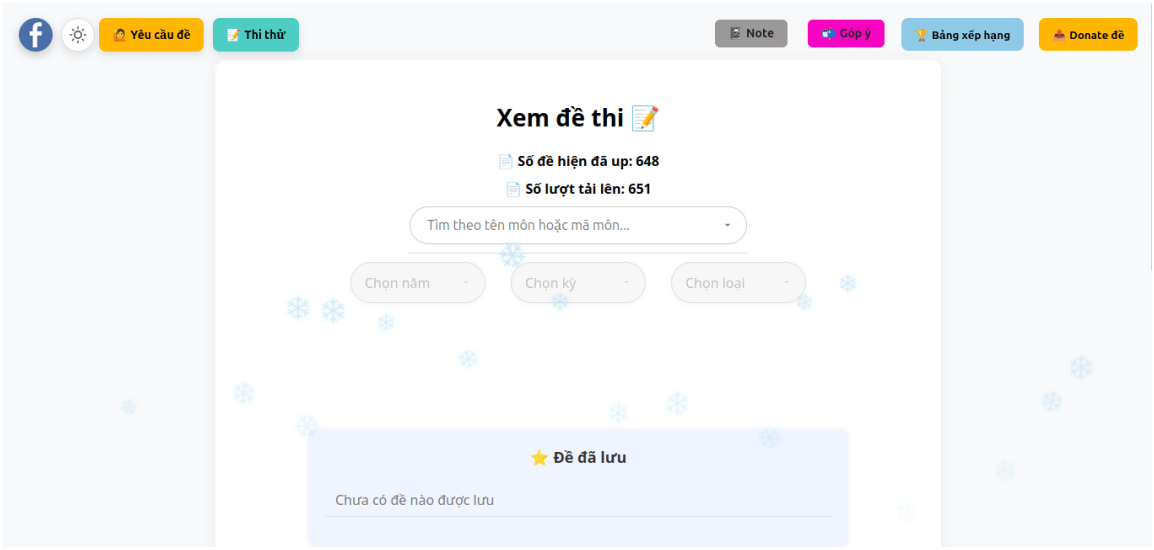
The project successfully delivers an end-to-end archive of academic resources, integrating Google Drive embedding, subject-based search, multi-level filtering, favorites management, average rating display, feedback submission, and update catalogue. The implementation of session-based view tracking and client-side persistence for favorites demonstrates effective state management without requiring user authentication. Furthermore, the modular routing structure and well-defined database schema with seed data contribute to the system's clarity, extensibility, and maintainability.

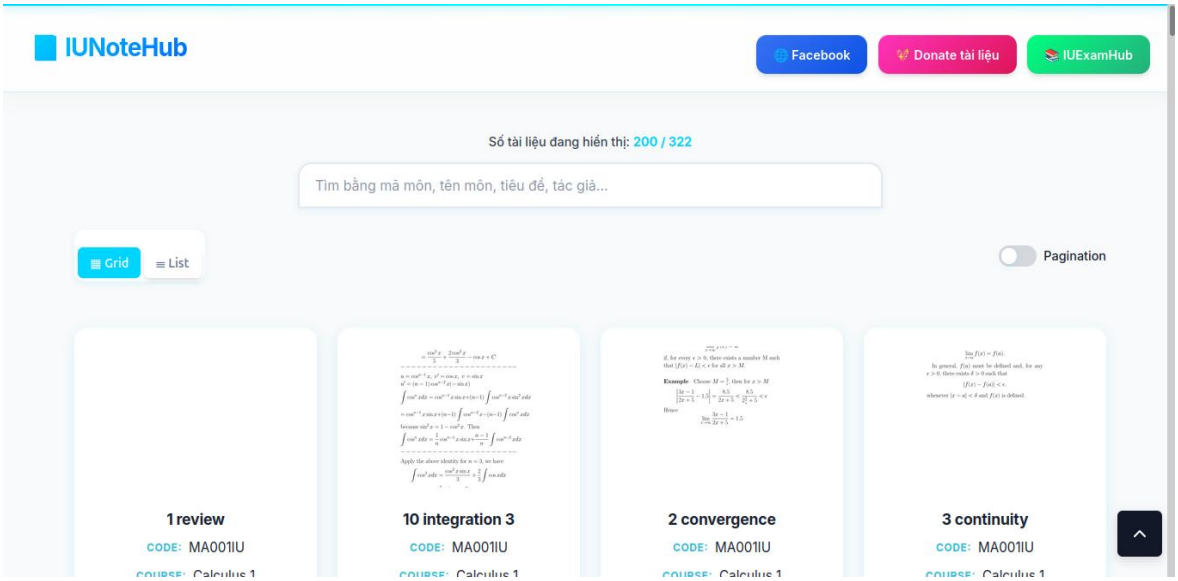
### 2) Limitations

Despite solid functionality during demonstration, the system still contains several limitations. The absence of user authentication means that favorites and preferences are stored only on the client side and are not portable across devices. Client-side validation and error messaging are relatively minimal, which may affect user experience in extreme cases. Ratings are directly tied to feedback submissions rather than managed through a standalone rating mechanism, and the project does not include automated testing or continuous integration pipelines, limiting robustness and long-term scalability.

### 3) Comparison with IUExamHub

When compared with platforms such as IUExamHub, Sample Test Archiver shares the common objective of providing centralized access to exam-related materials, underlying lightweight design choices, including embedded Google Drive viewing, session-based view tracking, and client-side favorites. Due to the concern of feature creeps in Software Engineering, this project only adds selective features such as star rating system and interactive feedback, rather than Google Form feedback type from IUExamHub. In contrast, such a comprehensive platform typically offers richer analytics (number of sample tests uploaded and contributions), separate lecture note hub, under-development mock test system and minor effects (snowfall, etc).


Yêu cầu đề
Thi thử
Note
Góp ý
Bảng xếp hạng
Donate đề
Xem đề thi
Số đề hiện đã up: 648
Số lượt tải lên: 651
Tìm theo tên môn hoặc mã môn...
Chọn năm
Chọn kỳ
Chọn loại
Đề đã lưu
Chưa có đề nào được lưu


IUNoteHub
Facebook
Donate tài liệu
IUExamHub
Số tài liệu đang hiển thị: 200 / 322
Tìm bằng mã môn, tên môn, tiêu đề, tác giả...
Grid
List
Pagination
1 review
CODE: MA001IU
10 integration 3
CODE: MA001IU
2 convergence
CODE: MA001IU
3 continuity
CODE: MA001IU


Top 5 Đề thi nổi bật
BA005IU - 2024 - Kỳ 1 - Cuối kỳ
20 lượt
BA003IU - 2022 - Kỳ 1 - Cuối kỳ
12 lượt
MA024IU - 2025 - Kỳ 2 - Giữa kỳ
6 lượt
Năng lượng nâng cấp web (mua tên miền & server):
Ủng hộ web

# VIII. CONCLUSION

## 1) Future Work

Further enhancements include the introduction of a light-dark mode toggle, user interface refinements, and the development of a recommendation system to suggest relevant sample tests based on statistical measurements. These improvements would enhance usability and personalization while preserving the project's lightweight philosophy.

## 2) Final Words

In conclusion, Sample Test Archiver achieves its goal of delivering a concise and user-friendly academic resource platform that emphasizes accessibility and feedback-driven quality indicators. By leveraging a straight-forward Node.js, Express.js, and MySQL technology stack, the project demonstrates a practical and maintainable approach to building an interactive archive to student needs.

# IX. REFERENCES


1. Thanh, N. (2025). *IU Exam Hub*. https://iuexamhub.netlify.app/
2. GeeksforGeeks. (2025). *Create an About us Page using HTML and CSS*. https://www.geeksforgeeks.org/css/design-an-about-us-page-using-html-and-css/
3. GeeksforGeeks. (2025). *Star Rating using HTML CSS and JavaScript*. https://www.geeksforgeeks.org/javascript/star-rating-using-html-css-and-javascript/
4. W3Schools. (n.d.). *HTML Tutorial*. https://www.w3schools.com/html/
5. W3Schools. (n.d.). *CSS Tutorial*. https://www.w3schools.com/css/default.asp
6. W3Schools. (n.d.). *JavaScript Tutorial*. https://www.w3schools.com/js/default.asp
7. NodeJS. (n.d.). *Node.js v25.2.1 Documentation*. https://nodejs.org/docs/latest/api/
8. Figma. (n.d.). *Figma: The Collaborative Interface Design Tool*. https://www.figma.com/
9. Railway. (n.d.). *Railway Documentation*. https://docs.railway.com/
10. Long, D. (2025). *BotCoffeeAndTea*. Github Repository. https://github.com/GaZ74/BotCoffeeAndTea.github.io
11. ThoughtSpot. (n.d.). *Performance Tuning SQL Queries.* https://www.thoughtspot.com/sql-tutorial/sql-performance-tuning