Title

Subtitle that is longer than the title

Summer Term 2024

September 22, 2025

Contents

1 Introduction

The importance of secure software systems has grown significantly in recent years, as vulnerabilities in applications continue to be a leading cause of data breaches, financial loss, and erosion of user trust. Despite this, many applications are still developed with insufficient attention to security principles, leading to exploitable flaws that could have been avoided through awareness and secure coding practices.

The primary aim of this project is to design and implement a deliberately insecure web-based system, and subsequently use it as an educational tool to demonstrate common security flaws and their impact. By first exposing students to insecure implementations, the project emphasizes the importance of security in the software development life cycle and builds awareness of how seemingly minor oversights can result in severe vulnerabilities.

To introduce the concept of insecure systems in a relatable manner, the well-known "Bobby Tables" cartoon was presented as a starting point. This example highlights the risks of improper input validation and serves as a memorable entry point to the broader field of software security.

For the literature review and hands-on exploration, the project made use of the *OWASP Juice Shop* platform, which is widely recognized as one of the most comprehensive and intentionally vulnerable web applications available for educational purposes. Students were instructed to attempt exploitation of its vulnerabilities by relying only on the hints provided, rather than directly consulting the solutions. This approach encouraged independent problem-solving, critical thinking, and a deeper understanding of how security weaknesses can be identified and exploited in practice.

Through this foundation, the project lays the groundwork for developing a school management system that demonstrates both insecure and secure implementations. The insecure version serves as a practical learning environment, while the secure version highlights industry-recommended best practices, thereby bridging the gap between theory and application in secure software engineering.

2 Related Work

The foundation of this project draws heavily from the **OWASP Juice Shop**, an intentionally vulnerable web application developed and maintained by the Open Web Application Security Project (OWASP). Juice Shop is widely recognized as one of the most comprehensive training grounds for web application security, as it incorporates vulnerabilities that map directly to the **OWASP Top 10** categories as well as numerous other security flaws encountered in real-world systems.

The OWASP Juice Shop was not only used as a reference, but also as a hands-on learning platform. Students were instructed to attempt exploitation of its vulnerabilities by relying exclusively on the integrated *hint system*, rather than consulting the provided solutions. This approach was intended to encourage independent exploration, critical thinking, and the development of problem-solving strategies commonly used by penetration testers and security researchers.

Through this practical engagement, students encountered a wide range of vulnerabilities including *injection attacks*, *broken authentication*, *insecure direct object references (IDOR)*, and *security misconfigurations*. These exercises provided both a theoretical grounding in web security concepts and a practical appreciation of how attackers exploit insecure code in real-world contexts.

By incorporating the OWASP Juice Shop into the project's foundation, the work is situated within a well-established security education framework. It also ensures that the deliberately insecure web application developed in this project, named the **Bobby School of Cyber Security**, reflects realistic vulnerabilities, thereby strengthening its effectiveness as a teaching tool for secure software engineering practices.

3 Project Specific Topic

Implementation

In this chapter, the implementation of the Django-based School Management System is briefly summarized and explained. Detailed annotations to the code are provided as comments directly in the source files and are therefore not repeated here. Instead, the general concepts, design choices, and module-level implementations are highlighted and discussed.

4 Third-Party Libraries

The School Management System leverages several third-party libraries and Python packages to implement its web-based functionalities efficiently and securely. This section lists all major external dependencies, their purpose, and the modules that utilize them.

- **Django [djangoproject]**: Django is the core web framework used throughout the application. It handles routing, request/response cycles, session management, ORM-based database interactions, authentication, and template rendering. All modules, including student registration, timetable management, and announcements, rely on Django for backend operations.
- **django-simple-captcha [djangocaptcha]**: This library is used for CAPTCHA generation and validation. Modules such as `teacher_registration` and `validate_captcha_manual` utilize it to prevent automated bot submissions. Manual validation is implemented in `validate_captcha_manual.py` with customized logic for response checking.
- **jQuery / AJAX**: Used primarily in the `vote_announcement` module to submit votes asynchronously via JSON without reloading the page. This allows students to upvote or downvote announcements interactively.
- **Bootstrap**: Provides responsive styling and UI components for HTML templates across the system. Although optional, it ensures consistent and professional front-end design for all web pages.
- **Pillow**: Used in modules that handle file uploads, such as `teacher_registration`. It provides image processing capabilities, including resizing and saving uploaded files.
- **Python Standard Libraries**:
  - `os` – file system and path handling.

- `time` – timestamps for CAPTCHA validation and form submissions.
- `json` – parsing and formatting JSON payloads (e.g., in `vote_announcement`).
- `random` – selecting teachers, rooms, and timeslots in the timetable module.
- `logging` – centralized logging and error tracking across modules.
- `functools` – used for decorators like `session_required`.
- `datetime` – date validation and formatting in forms and timetable entries.

- **Database Drivers**:
  - SQLite / PostgreSQL – Persistent storage of all entities such as `Student`, `TeacherReg`, `TimetableEntry`, and `AnnouncementVote`. Managed through Django's ORM to ensure data integrity and security.

Notes on Custom Modifications:

- The manual CAPTCHA validation module includes a lowercasing feature for the user response to match stored values.
- The voting module currently allows duplicate votes by the same student on a single announcement; this vulnerability is included intentionally for educational purposes.

5 Assignment Management Module

The `assignment.py` module implements the core functionality for handling assignments within the school application. It supports both teacher and student workflows: teachers can create questions, review submissions, and grade them, while students can view available questions, submit assignments, and track their submission history. The main code module is located at */login/assignment.py*.

In order to deliberately demonstrate insecure software practices, this module introduces an exploitable XML parsing routine. The function `parse_xml()` employs the `lxml` parser with `load_dtd` and `resolve_entities` enabled, which makes the system vulnerable to XML External Entity (XXE) attacks such as the "Billion Laughs" denial-of-service vector. This vulnerability highlights how insecure handling of user-uploaded files can compromise system integrity.

5.1 Dependencies

The module depends on multiple Django components and third-party libraries:

- `django.shortcuts` for rendering templates and managing redirects.
- `django.contrib.messages` for user-facing notifications.
- `django.db.models.Q` for complex query filtering.
- `django.utils.timezone` for timestamp management.
- `lxml.etree` for XML parsing (insecurely configured).
- Custom models from *login.models*: `Student`, `TeacherAvailability`, `ClassSection`, `Subject`, `AssignmentQuestion`, `AssignmentSubmission`.
- Custom forms from *login.forms*: `AssignmentQuestionForm`, `AssignmentSubmissionForm`, `GradeSubmissionForm`, `SubmissionFilterForm`.

5.2 Session Decorator

The decorator `session_required` enforces role-based access control by checking session keys. It ensures that only authenticated users with the correct role (teacher or student) can access assignment functionality. Invalid sessions are flushed, and the user is redirected to the application index.

5.3 Main Functions

The module contains several core functions, which can be grouped by teacher and student responsibilities:

Teacher Workflows

- **Create Question** (`teacher_create_question`): Allows teachers to post questions restricted to their assigned subjects and class sections.
- **List Questions** (`teacher_questions_list`): Displays all questions authored by the teacher.
- **Review Submissions** (`teacher_review_submissions`): Provides filtering by subject, class section, and status. Implements ordering by submission time.
- **Mark Seen** (`teacher_mark_seen`): Updates submission status to `seen`.
- **Grade Submission** (`teacher_grade_submission`): Assigns marks to a student submission and updates its status to `graded`.

Student Workflows

- **List Questions** (`student_questions_list`): Shows available questions for a student's class and subjects.
- **Submit Assignment** (`student_submit_for_question`): Enables students to upload XML files or typed content as assignment answers. Contains the XXE vulnerability in the XML parsing routine.
- **Track Submissions** (`student_my_submissions`): Lists a student's past submissions with their current status and marks.

5.4 Security Weakness

This module intentionally demonstrates insecure coding practices. The XML parsing routine is configured with both `load_dtd` and `resolve_entities` enabled, making the application susceptible to XML External Entity (XXE) attacks. An attacker could upload a malicious XML file to perform denial-of-service or exfiltrate server-side files. This highlights the importance of secure file-handling practices in web applications.

6 Announcement Management Module

The `announcement.py` module manages announcements in the school application. It enables teachers to post announcements for their assigned subjects and class sections, while students can view announcements relevant to their enrolled classes. The main code module is located at */login/announcement.py*.

This module also demonstrates insecure coding practices by introducing a Cross-Site Scripting (XSS) vulnerability. User-submitted announcement content is not properly sanitized, allowing malicious users to inject scripts into announcements, which can then be executed in the browser of other users.

6.1 Dependencies

The module depends on several Django components and custom models/forms:

- `django.shortcuts` for rendering templates and managing redirects.
- `django.contrib.messages` for user-facing notifications.
- `django.utils.timezone` for handling announcement timestamps.
- Custom models from *login.models*: `Announcement`, `TeacherAvailability`, `Student`, `ClassSection`, `Subject`.
- Custom forms from *login.forms*: `AnnouncementForm`, `AnnouncementFilterForm`.

6.2 Session Decorator

The decorator `session_required` enforces role-based access control. It ensures that only authenticated teachers can create announcements and only authenticated students can view them. If the session is invalid, it is flushed, and the user is redirected to the index page.

6.3 Main Functions

Teacher Workflows

- **Create Announcement** (`teacher_create_announcement`): Allows teachers to post announcements for their assigned subjects and class sections. The function validates that teachers cannot post outside of their scope.
- **List Announcements** (`teacher_announcements_list`): Displays announcements created by the currently logged-in teacher, ordered by creation time.

Student Workflows

- **View Announcements** (`student_announcements_list`): Shows announcements relevant to a student's enrolled class section and subjects. Announcements are retrieved from the database, optionally filtered by subject, and displayed in reverse chronological order.

6.4 Security Weakness

This module introduces an intentional security weakness: improper input sanitization leading to Cross-Site Scripting (XSS). Malicious users could craft announcements containing embedded JavaScript code, which would then execute in the browsers of other users viewing the announcement. This highlights the risks of failing to escape or sanitize user-generated content in web applications.

7 Dashboard Management Module

The `dashboard.py` module implements the role-based dashboards of the school application. It defines separate dashboards for administrators, teachers, and students, with each view presenting functionalities specific to the logged-in role. The main code module is located at */login/dashboard.py*.

This module demonstrates secure session validation practices by ensuring that each dashboard can only be accessed by an authenticated user of the correct role. Although no deliberate vulnerabilities are embedded in this file,

misconfiguration of role checks or session handling could potentially expose sensitive functionalities.

7.1 Dependencies

The module depends on the following Django components:

- `django.shortcuts` for rendering templates and handling redirects.
- `django.views.decorators.cache.never_cache` to prevent cached dashboard responses.
- `functools.wraps` to preserve metadata when applying decorators.

7.2 Session Decorator

The decorator `session_required` validates that the user session contains the correct role key (`Admin_login`, `Teacher_login`, or `Student_login`). If the session is invalid, it is flushed and the user is redirected to the application index. This enforces strict role-based access control to dashboards.

7.3 Main Functions

Administrator Dashboard

- **Admin Dashboard** (`admin_dashboard`): Provides access to administrative features such as managing timetables, approving users, viewing logs, managing pinboards, and profile management.

Teacher Dashboard

- **Teacher Dashboard** (`teacher_dashboard`): Offers functionality for teachers including posting questions, reviewing submissions, managing marks, viewing timetables, posting announcements, managing pinboards, and accessing their profile.

Student Dashboard

- **Student Dashboard** (`student_dashboard`): Allows students to view marks, access questions and submissions, manage profiles, interact in Q&A forums, view announcements and timetables, and use pinboards.

7.4 Security Considerations

The module itself does not introduce explicit vulnerabilities. However, because dashboards aggregate access to critical features, any weakness in the

session validation mechanism could expose privileged functionality to unauthorized users. Ensuring secure role checks and proper session handling is essential.

8 Error Management Module

The `errormanagement.py` module centralizes error handling and session validation across the school application. It provides utilities for rendering templates safely and enforcing robust role-based session management. The main code module is located at */login/errormanagement.py*.

By introducing structured error handling, this module reduces the likelihood of uncaught exceptions propagating to end users, and improves resilience against faulty templates, database errors, or session misuse.

8.1 Dependencies

The module depends on both Django core components and Python's standard logging framework:

- `django.shortcuts` for rendering templates and managing redirects.
- `django.http.HttpResponseServerError` for returning server error responses.
- `django.core.exceptions.SuspiciousOperation` for handling invalid requests.
- `django.db.DatabaseError` for handling database-related failures.
- `django.contrib.messages` for user-facing notifications during session failures.
- `django.views.decorators.cache.never_cache` to prevent caching of session-sensitive views.
- `django.template.TemplateDoesNotExist` for catching template resolution errors.
- Python's `logging` module for structured error logging.

8.2 Safe Rendering

The function `safe_render()` wraps Django's `render()` to provide fault-tolerant template rendering. It catches specific exceptions:

- **TemplateDoesNotExist**: Logs the error and returns a generic server error response.

- **Unhandled Exceptions**: Attempts to render a fallback `errors_500.html` template with a user-friendly message.

If rendering the fallback template also fails, a plain `HttpResponseServerError` is returned.

8.3 Session Decorator

The decorator `session_required` extends the role-based access control mechanism by including error handling for:

- **Expired or Invalid Sessions**: Flushes the session, notifies the user, and redirects to the index page.
- **SuspiciousOperation**: Logs the incident as a warning, resets the session, and forces re-authentication.
- **DatabaseError**: Logs the exception and renders an error page indicating a database failure.
- **Other Exceptions**: Logs the error and displays a generic fallback message.

This approach ensures that critical issues are caught gracefully while preserving security and user awareness.

8.4 Security Considerations

This module strengthens the overall security posture of the system by:

- Preventing sensitive exception traces from being exposed to end users.
- Handling suspicious sessions proactively to reduce session fixation and tampering risks.
- Logging all unexpected errors for post-incident investigation.

Although no deliberate vulnerability is introduced here, improper configuration of error messages or logging could still leak sensitive information if not handled correctly.

9 Error Handler Module

The `error_view.py` module defines custom error handlers that provide user-friendly responses for common HTTP error conditions. It replaces default Django error pages with tailored templates that display meaningful messages to end users. The main code module is located at */login/error_ view.py.*

This improves the user experience by avoiding exposure of raw error traces and ensures that system errors are logged for developers.

9.1 Dependencies

The module relies on:

- `django.shortcuts.render` for rendering error templates.
- `django.http.HttpResponseServerError` for returning server error responses when rendering fails.
- `django.conf.settings` for toggling debug-mode messages.
- Python's `logging` module for capturing error details during rendering failures.

9.2 Safe Error Rendering

The helper function `_safe_error_render()` ensures resilience during error page rendering:

- Attempts to render the requested error template with a user-friendly message.
- Falls back to the `login/errors_500.html` template if rendering fails.
- As a last resort, returns a minimal `HttpResponseServerError`.

This layered approach guarantees that a response is always provided, even if templates are missing or misconfigured.

9.3 Custom Error Views

The module defines four handlers for common HTTP error codes:

- **400 Bad Request** (`custom_bad_request_view`): Displays a message when the client request is malformed.
- **403 Forbidden** (`custom_permission_denied_view`): Alerts the user that they lack permission to access the requested resource.
- **404 Not Found** (`custom_page_not_found_view`): Returns a user-friendly page when the requested resource does not exist.
- **500 Internal Server Error** (`custom_server_error_view`): Provides a fallback error page for unhandled server-side exceptions.

9.4 Security Considerations

This module enhances security by:

- Preventing internal server details and stack traces from being exposed to end users.
- Ensuring all errors are logged for developers without leaking sensitive system data.
- Providing consistent and user-friendly error messages that minimize confusion for non-technical users.

Improper configuration of templates could still degrade user experience, but the layered fallback approach minimizes risk of blank or insecure error responses.

10 Password Reset Module

The `password_reset.py` module implements the multi-step password recovery process for users of the school application. It validates usernames, enforces security question checks, and allows verified users to securely reset their password. The main code module is located at */login/password_ reset.py.*

This module demonstrates how user identity verification is enforced before allowing a password change, but also contains an intentional workflow weakness to illustrate poor security practices.

10.1 Dependencies

The module depends on:

- `django.shortcuts` for rendering templates and redirects.
- `django.contrib.messages` for providing user feedback.
- `django.views.decorators.cache.never_cache` to prevent caching of sensitive pages.
- Models `Login` and `Login2` from *login.models.*
- Forms `ForgotPasswordForm`, `SecurityAnswerForm`, and `ResetPasswordForm` from *login.forms.*
- Utility function `simple_hash` from *login.views* for password hashing.

10.2 Session Decorator

The `session_required` decorator is reused in this module to enforce that intermediate steps (username verification and security question validation) are completed before accessing subsequent steps in the password reset flow. If the required session key is missing, the user is redirected back to the application index.

10.3 Main Functions

Step 1: Enter Username

- Function: `forgot_password_step1`
- Clears any previous session state, accepts a username, and checks if it exists.
- If valid, stores the username in the session and redirects to the security question step.

Step 2: Verify Security Question

- Function: `forgot_password_step2`
- Fetches the stored username and prompts the user for the correct security question answer.
- On successful validation, sets a session flag (`forgot_verified`) and allows access to the reset page.
- If no security question is configured, the user is redirected to the index with a warning.

Step 3: Reset Password

- Function: `forgot_password_step3`
- Ensures that the username is valid and security verification is complete.
- Accepts new password and confirmation, validates them, and applies a hash before saving.
- Updates both `Login` and `Login2` model records for consistency.
- On success, clears the session flags and redirects the user to the index page.

10.4 Security Weakness

This module intentionally demonstrates a flawed password reset workflow:

- The system may allow users without configured security questions to bypass the verification step, reducing overall account protection.
- Passwords are saved both hashed (in `Login`) and in plaintext (in `Login2`), which is highly insecure and could lead to data exposure.
- Custom hashing via `simple_hash` is weaker than Django's built-in password hashing framework, making it easier for attackers to crack stolen credentials.

This highlights the importance of enforcing consistent verification, securely storing passwords, and using proven hashing algorithms.

11 Login Module

The `login.py` module implements the authentication system for the application. It provides user login, logout, and activity logging functionality with role-based redirection. The main code module is located at */login/login.py*.

This module ensures that only authenticated users gain access to the system and that their activities are logged for accountability. It also integrates error handling when access is denied or when log files are unavailable.

11.1 Dependencies

The module depends on:

- `django.shortcuts` for rendering templates and managing redirects.
- `django.http.HttpResponse` for returning responses such as log file contents.
- `login.models.Login` for accessing user credentials and roles.
- `login.forms.NewLoginForm` for validating login inputs.
- Utility function `simple_hash` from *login.views* for password hashing.
- Error handlers from *error_ views.py* for permission denial and missing resources.
- Python's `logging` module for recording login attempts.
- Python's `os` module for log file management.

11.2 Main Functions

View Logs

- Function: `view_logs`
- Allows administrators to access login activity logs.
- Validates that the requesting user has admin privileges before serving the log file.
- Unauthorized access attempts are blocked with a permission error.

Serve Log File

- Function: `serve_log_file`
- Reads and displays the contents of the `login_activity.log` file.
- If the log file is missing, raises a custom "Page Not Found" error.

Index (Login Page)

- Function: `index`
- Validates submitted credentials using `NewLoginForm`.
- Compares entered passwords with stored hashes to authenticate users.
- On success, assigns role-specific session variables and redirects to the appropriate dashboard (Admin, Teacher, or Student).
- Logs both successful and failed login attempts for accountability.

Logout

- Function: `logout_view`
- Clears the user session completely and redirects to the index page.

11.3 Security Considerations

This module strengthens authentication by logging all login attempts and enforcing role-based access. However, intentional weaknesses are present for demonstration purposes:

- Passwords are hashed with a custom `simple_hash` function, which is weaker than Django's built-in password hasher and could be vulnerable to cracking.

- Log files are stored on disk and displayed directly via HTTP without sanitization, which may expose sensitive information to unauthorized viewers if misconfigured.
- Role-based session flags are used instead of Django's more secure built-in authentication system, reducing overall robustness.

These weaknesses highlight the importance of using secure frameworks for password storage, log handling, and session management.

12 Marks Management Module

The `marks.py` module provides functionality for teachers to add marks for students and for students to view their marks. It supports role-based workflows: teachers can enter marks for a class and subject, while students can view their own marks with optional filtering and sorting. The main code module is located at */login/marks.py*.

12.1 Dependencies

The module relies on several Django components and Python standard libraries:

- `django.shortcuts` for rendering templates and managing redirects.
- `django.contrib.messages` for user-facing notifications.
- `django.views.decorators.cache.never_cache` to prevent caching of sensitive pages.
- `django.http.JsonResponse` for AJAX responses.
- `django.shortcuts.get_object_or_404` for safe object retrieval.
- `datetime` and `json` from Python standard library.
- Custom models from *login.models*: `Student`, `Marks`, `ClassSection`, `Subject`, `TeacherAvailability`.
- Custom forms from *login.forms*: `EnterStudentMarksForm`, `SelectClassSubjectForm`.

12.2 Session Decorator

The decorator `session_required` enforces role-based access control by checking session keys. It ensures that only authenticated users with the correct role (teacher or student) can access marks functionality. Invalid sessions are flushed, and the user is redirected to the application index.

12.3 Main Functions

The module contains several core functions, which can be grouped by teacher and student responsibilities:

Teacher Workflows

- **Add Marks Step 1** (`add_marks_step1`): Allows teachers to select class, subject, exam type, exam date, and total marks for entering student results.
- **Add Marks Step 2** (`add_marks_step2`): Enables teachers to input marks for all students in the selected class and subject. Prevents duplicate entries and validates input.

Student Workflows

- **View Marks** (`view_marks`): Displays marks for the logged-in student. Provides filtering by subject and class section, and allows sorting by marks.
- **Set Student Session from Storage** (`set_sid_from_storage`): Updates the `student_username` session variable via AJAX to synchronize frontend local storage with server session.

12.4 Security Weakness

The module has potential security risks:

- Students could manipulate session data to view other students' marks.
- Teachers might accidentally or maliciously enter marks for unauthorized classes or subjects.
- Input validation prevents duplicate entries but does not fully prevent session tampering.

13 Password Management Module

The `password.py` module provides functionality for users to change their password and set or update a security question. It supports role-based workflows: all users (Admin, Teacher, Student) can update their password after entering the current password correctly, and optionally configure their security question. The main code module is located at *IntegratedApplication/bobby/login/views/password.py*.

13.1 Dependencies

The module relies on several Django components and custom forms:

- `django.shortcuts` for rendering templates and managing redirects.
- `django.contrib.messages` for user-facing notifications.
- Custom models from *login.models*: `Login`, `Login2`.
- Custom forms from *login.forms*: `ChangePasswordForm`, `SecurityQuestionForm`.
- Custom views/utilities from *login.views*: `simple_hash` for password hashing.

13.2 Session Decorator

Session management is handled implicitly by checking the `login_username` and `current_role` session variables. If a session is invalid or missing, the user is redirected to the login index page.

13.3 Main Functions

The module contains two core functions, which apply to all roles (Admin, Teacher, Student):

- **Change Password** (`change_password`): Allows a logged-in user to update their password. Validates the current password, ensures the new passwords match, updates the hashed password in both `Login` and `Login2` models, and provides success/error messages. All users follow the same workflow.
- **Set/Update Security Question** (`security_question`): Enables a user to configure or update their security question and answer. Validates input and stores the data in the `Login` model.

13.4 Security Weakness

Potential security risks include:

- If session variables are manipulated, a user could potentially attempt unauthorized password changes.
- No rate-limiting is implemented; repeated attempts could facilitate brute-force attacks.
- Security question functionality is optional, and skipping it could leave accounts with weaker recovery options.

14 Student Question Posting Module

The `post_questions.py` module enables students to post questions to the learning platform. It ensures that only authenticated students can submit questions, and stores submissions in the database. The main code module is located at *integratedApplication_ questions.py*.

This functionality lays the foundation for an interactive environment, allowing students to engage actively and fostering communication and collaboration.

14.1 Dependencies

The module relies on the following Django components and custom forms/models:

- `django.shortcuts.render` for template rendering.
- `django.shortcuts.redirect` for page redirection.
- `django.shortcuts.get_object_or_404` for safe object retrieval.
- Custom models from *login.models*: `Student`, `Question`.
- Custom forms from *login.forms*: `QuestionForm`.
- Custom session management from *login.views*: `session_required` decorator.

14.2 Session Decorator

The decorator `session_required` enforces that only authenticated students can access this functionality. Invalid or missing sessions result in redirection to the student login page.

14.3 Main Functions

The module contains a single core function for student workflow:

- **Post Question** (`post_questions`): Validates the student's session, retrieves the logged-in student object, and handles form submission. On POST request, the question text is validated and saved to the `Question` model. On GET request, an empty form is rendered for the student. Successful submissions redirect back to the student dashboard.

14.4 Security Weakness

Potential security concerns include:

- Malicious input in question text could lead to stored XSS attacks if not properly sanitized.
- Session tampering could allow unauthorized users to attempt posting, though the decorator mitigates this risk.
- No rate-limiting or spam prevention, allowing repeated submissions.

15 Search Module

The `search.py` module provides search functionality for students and teachers. It allows users to search for other students or teachers based on name queries. The main code module is located at *integratedApplication/ bobby/ login/views/ search.py*.

15.1 Dependencies

The module relies on several Django components and custom forms/models:

- `django.shortcuts.render` for rendering templates.
- `django.db.connection` for executing raw SQL queries.
- Custom models from *login.models*: `Student`, `Teacher`.
- Custom forms from *login.forms*: `SearchFormStudent`, `SearchFormTeacher`.

15.2 Session Decorator

No session decorator is applied in this module; it is assumed that access is controlled via higher-level authentication checks.

15.3 Main Functions

The module contains two primary functions, one for searching student and one for searching teacher. These functions handle the search workflow, form validation, query execution, and result rendering.

15.4 Main Functions

The module contains two primary functions, one for student searches and one for teacher searches. These functions handle the search workflow, form validation, query execution, and result rendering.

Student Search Workflow (`search_student`)

- Displays a search form to the user.
- Accepts a name query submitted via POST request.
- Constructs and executes a raw SQL query on the `login_student` table to find matches by `student_first_name`.
- Retrieves query results and formats them as dictionaries.
- Renders the `login/search.html` template with the form, search results, and query.

Teacher Search Workflow (`search_teacher`)

- Displays a search form to the user.
- Accepts a name query submitted via POST request.
- Constructs and executes a raw SQL query on the `login_teacher` table to find matches by `firstname`.
- Retrieves query results and formats them as dictionaries.
- Renders the `login/search_teacher.html` template with the form, search results, and query.

15.5 Security Weakness

The module contains a significant security vulnerability:

- The use of raw SQL queries with direct string interpolation exposes the application to SQL Injection attacks.
- User input is not sanitized or parameterized, allowing malicious users to manipulate queries and potentially access unauthorized data.

16 Student Registration Module

The `student_registration.py` module provides functionality for new students to register via an online form. The registration workflow includes form submission, CAPTCHA validation, and database entry of student details. The CAPTCHA is intended to prevent automated submissions, but its implementation contains a deliberate vulnerability. The main code module is located at *integratedApplication\bobby\login\views\student_ registration.py.*

16.1 Dependencies

The module relies on the following Django components, libraries, and custom models/forms:

- `django.shortcuts` for rendering templates and handling redirects.
- `django.contrib.messages` for user notifications.
- `django.core.cache` for storing and retrieving CAPTCHA reuse attempts.
- `django.http.JsonResponse` for returning JSON responses in replay attack scenarios.
- `time` from Python standard library for timestamp tracking.
- `captcha.models.CaptchaStore` and `captcha.helpers.captcha_image_url` for CAPTCHA generation and validation.
- Custom model from `login.models`: `StudentReg`.
- Custom form from `login.forms`: `StudentRegistration`.
- Custom helper: `validate_captcha_manual` for manual CAPTCHA verification.

16.2 Session Decorator

No session decorator is applied in this module, as registration is intended for new, unauthenticated users.

16.3 Main Functions

The module contains a single core function:

Student Workflow (`student_registration`)

- Displays the student registration form along with a CAPTCHA challenge.
- Accepts POST submissions and performs manual CAPTCHA validation.
- Tracks CAPTCHA reuse attempts within a defined time window (20 seconds).
- If CAPTCHA is reused more than the limit, responds with a JSON message confirming replay attack success.

- If the form is valid, stores student information (`firstname`, `lastname`, `dob`, `gender`, `email`, `classlevel`) in the `StudentReg` table.
- Returns success notifications and redirects the student upon successful registration.

16.4 Security Weakness

This module intentionally implements CAPTCHA in a vulnerable way:

- The same CAPTCHA response can be reused multiple times within a 20-second window.
- This design enables replay attacks, allowing attackers to bypass CAPTCHA protection.
- Exploiting this vulnerability, attackers can flood the system with fake registrations, leading to spam data, resource exhaustion, and database integrity issues.

17 Student Timetable Module

The `student_timetable.py` module provides functionality for students to view their academic timetable. It ensures that only authenticated students can access timetable data by validating sessions before fetching records. The function retrieves the logged-in student's class level and matches it with timetable entries, optionally filtering by day. The main code module is located at *integratedApplication\bobby\login\views\student_timetable.py*.

17.1 Dependencies

The module depends on the following Django components and models:

- `django.shortcuts.render` for rendering templates.
- Custom models from `login.models`: `Student`, `TimetableEntry`.
- Custom decorator from `login.views`: `session_required`.

17.2 Session Decorator

The decorator `session_required('Student_login')` enforces that only authenticated students with valid sessions can access the timetable view. If the session is invalid or expired, the function returns an error message to the frontend.

17.3 Main Functions

The module contains a single core function responsible for rendering student timetables:

Student Workflow (`student_timetable_view`)

- Retrieves the logged-in student's username from the session.
- Validates that the username exists in the `Student` table; otherwise, returns an error message.
- Extracts the student's `classlevel` and queries the `TimetableEntry` model for matching timetable records.
- Optionally filters timetable entries by day if a day parameter is provided in the request.
- Optimizes query performance by using `select_related` to fetch related objects such as subject, teacher, room, timeslot, and class section in a single query.
- Renders the `login/student_timetable.html` template with timetable entries, available days, the selected day, and the student's class information.

17.4 Security Weakness

The module does not explicitly introduce intentional vulnerabilities. However, possible concerns include:

- Insufficient input validation on the optional `day` parameter could lead to unexpected behavior if improperly handled.
- Debugging statements (e.g., `print`) expose internal information, which should be removed in production environments to prevent information leakage.

18 Teacher Registration Module

The `teacher_registration.py` module provides functionality for registering new teachers via an online form. The workflow includes form submission, CAPTCHA validation, and file upload handling. The module demonstrates both CAPTCHA replay vulnerabilities and weak file validation practices, highlighting security pitfalls in registration systems. The main code module is located at *integratedApplication\bobby\login\teacher_registration.py*.

18.1 Dependencies

The module relies on the following Django components, libraries, and custom models/forms:

- `django.shortcuts` for rendering templates and handling redirects.
- `django.contrib.messages` for user notifications.
- `django.core.cache` for tracking CAPTCHA reuse attempts.
- `django.http.JsonResponse` for returning JSON responses during replay attack confirmation.
- `time` from Python standard library for timestamp tracking.
- `captcha.models.CaptchaStore` and `captcha.helpers.captcha_image_url` for CAPTCHA generation and validation.
- Custom model from `login.models`: `TeacherReg`.
- Custom form from `login.forms`: `TeacherRegistration`.
- Custom helper: `validate_captcha_manual` for manual CAPTCHA verification.

18.2 Session Decorator

No session decorator is applied in this module, as teacher registration is intended for new, unauthenticated users.

18.3 Main Functions

The module contains a single core function:

Teacher Workflow (`teacher_registration`)

- Displays the teacher registration form along with a CAPTCHA challenge.
- Accepts POST submissions and performs manual CAPTCHA validation.
- Tracks CAPTCHA reuse attempts within a defined time window (20 seconds).
- If CAPTCHA is reused more than the limit, responds with a JSON message confirming replay attack success.

- Validates submitted teacher details and saves them to the `TeacherReg` table.
- Handles optional file uploads:
  - Checks if the uploaded file extension matches allowed formats (`.jpg`, `.png`, `.pdf`, `.doc`, `.docx`, `.jpeg`).
  - Ensures that file size does not exceed 7 MB.
  - Associates the valid uploaded document with the teacher's record.
- Returns success notifications and redirects to the homepage upon successful registration.

18.4 Security Weakness

This module contains two intentional vulnerabilities:

- **CAPTCHA Replay Attack:** The CAPTCHA validation allows the same token to be reused multiple times within a short time window, enabling attackers to bypass CAPTCHA protection and flood the system with fake registrations.
- **Weak File Validation:** The file upload validation only checks extensions and size. This approach can be easily bypassed (e.g., by renaming a malicious executable to a valid extension), leading to potential remote code execution or malware storage.

19 Timetable Management Module

The `timetable_management.py` module provides functionality for administrators and teachers to manage and view timetables. Administrators can add requirements (subjects, teachers, class sections, rooms, and timeslots), generate timetables based on these inputs, and view class schedules. Teachers can log in to view their assigned timetables. The main code module is located at *integratedApplication\bobby\login\timetable_ management.py.*

19.1 Dependencies

The module relies on the following Django components, models, and forms:

- `django.shortcuts` for rendering templates and handling redirects.
- `django.db.models` for ORM-based queries and filtering.
- `random` from Python standard library for randomized timetable slot assignments.

- Custom models from `login.models`: `TimetableEntry`, `ClassSection`, `Subject`, `TeacherAvailability`, `Room`, `TimeSlot`.
- Custom forms from `login.forms`: `SubjectForm`, `TeacherForm`, `ClassSectionForm`, `RoomForm`, `TimeSlotForm`.
- Custom session management decorator from `login.views`: `session_required`.

19.2 Session Decorator

All views in this module are protected by the `session_required` decorator:

- Administrator views require the session type `'Admin_login'`.
- Teacher views require the session type `'Teacher_login'`.

19.3 Main Functions

Administrator Workflows

- **Create Timetable** (`create_timetable`): Displays the timetable creation homepage.
- **Add Subject** (`add_subject`): Provides a form to add new subjects to the system.
- **Add Teacher** (`add_teacher`): Allows adding teacher details.
- **Add Class Section** (`add_classsection`): Facilitates creation of new class sections.
- **Add Room** (`add_room`): Allows the administrator to add rooms with specified types.
- **Add Time Slot** (`add_timeslot`): Adds available time slots to the schedule pool.
- **Generate Timetable View** (`generate_timetable_view`): Invokes the timetable generation process and displays any unmet requirements (failures).
- **Admin Timetable View** (`admin_timetable_view`): Displays timetables with filtering options by class, subject, teacher, and day.

Teacher Workflows

- **Teacher Timetable View** (`teacher_timetable_view`): Retrieves the logged-in teacher's assigned timetable, with an option to filter by day.

Timetable Generation Logic

- **Generate Timetable** (`generate_timetable`):
  - Clears existing timetable entries.
  - Randomly assigns subjects, teachers, rooms, and timeslots to meet weekly requirements.
  - Enforces teacher constraints such as maximum daily and weekly load.
  - Ensures no conflicts in room, teacher, or class section assignments.
  - Returns a list of failures for unassigned periods (if requirements cannot be fully met).

19.4 Security Weakness

This module does not contain any deliberate or intentional vulnerabilities. However, certain operational limitations may pose risks if not managed carefully:

- The random assignment of teachers, rooms, and timeslots may lead to inefficiencies or unfair workloads without additional scheduling constraints.
- The module relies on session-based access control; secure session management at the framework level is essential to prevent unauthorized access.

20 Manual CAPTCHA Validation Module

The `validate_captcha_manual.py` module provides a helper function for validating CAPTCHA responses submitted by users. It checks the provided CAPTCHA response against entries stored in Django's `CaptchaStore`. This function is used by other modules such as student and teacher registration to verify CAPTCHA inputs. The main code module is located at `view/validate_captcha_manual.py`.

20.1 Dependencies

The module relies on the following components:

- `captcha.models.CaptchaStore` for retrieving stored CAPTCHA entries.
- `captcha.helpers.captcha_image_url` for generating the image URL (though not directly used in validation).

20.2 Session Decorator

No session decorator is applied in this module, as it serves purely as a helper function for CAPTCHA validation and is not directly exposed as a user-facing view.

20.3 Main Functions

The module contains a single core function:

Function Workflow (`validate_captcha_manual`)

- Accepts a CAPTCHA ID (`captcha_id`) and user-provided response (`captcha_response`).
- Retrieves the corresponding CAPTCHA entry from the `CaptchaStore`.
- Validates the response against the stored CAPTCHA value.
- Returns `True` if the response matches, otherwise returns `False`.
- Handles cases where the CAPTCHA entry does not exist by returning `False`.

20.4 Security Considerations

This helper module does not introduce any direct security vulnerabilities on its own. It is a utility function designed for input validation and relies on Django's `CaptchaStore` for secure storage and retrieval of CAPTCHA entries.

21 Announcements Module

The `view_announcement.py` module provides functionality for students to view teacher announcements and interact with them through an upvote/downvote feature. It serves as the foundation of an interactive student announcement system, enabling transparent communication and feedback between teachers and students. The main code module is located at `views/view_announcement.py`.

21.1 Dependencies

The module relies on the following Django components and models:

- `django.shortcuts` for rendering templates and handling redirects.
- `django.db.models.Count, Q` for database aggregation and filtering.
- `login.models.TeacherAnnouncement` for storing announcements.

- `login.models.AnnouncementVote` for tracking student votes on announcements.
- `login.models.Student` for identifying the logged-in student.
- Custom decorator: `session_required` for enforcing session-based access control.

21.2 Session Decorator

The `session_required('Student_login')` decorator ensures that only authenticated students can access the announcement view. If the session is invalid or missing, the user is redirected to the login page.

21.3 Main Functions

The module contains one primary view function:

Student Workflow (`view_announcement`)

- Retrieves the current student's username from the session.
- Ensures that the username exists, otherwise redirects the student to the login page.
- Fetches the `Student` object corresponding to the username.
- Queries all `TeacherAnnouncement` objects, annotating each with the total number of upvotes and downvotes.
- Retrieves the student's existing votes (`AnnouncementVote`) to indicate their voting status.
- Organizes the student's votes into a dictionary for efficient template rendering.
- Prepares a dictionary mapping announcement IDs to vote counts.
- Renders the `view_announcement.html` template with announcements, vote counts, and student vote information.

21.4 Security Considerations

This module does not introduce direct security vulnerabilities. The use of session-based access control ensures that only logged-in students can view and vote on announcements. However, input validation and vote submission logic must be carefully handled in complementary modules to prevent issues such as vote tampering or duplicate voting.

22 Vote Announcement Module

The `vote_announcement.py` module provides functionality for students to submit votes (upvote or downvote) on teacher announcements. The module processes POST requests containing JSON data, validates the vote, records it in the database, and returns updated vote counts as a JSON response. This function is part of the interactive student announcement and voting system. The main code module is located at `integratedApplication/bobby/login/views/vote_announcement.py`.

22.1 Dependencies

The module relies on the following Django components and models:

- `django.http.JsonResponse` for returning JSON responses.
- `json` from Python standard library for parsing request payloads.
- `login.models.AnnouncementVote` for storing votes.
- `login.models.TeacherAnnouncement` for identifying announcements.
- `login.models.Student` for identifying the logged-in student.
- Custom decorator: `session_required` for enforcing session-based access control.

22.2 Session Decorator

The `session_required('Student_login')` decorator ensures that only authenticated students can submit votes. If the session is invalid or missing, the user cannot perform voting actions.

22.3 Main Functions

The module contains a single primary view function:

Student Workflow (`vote_announcement`)

- Accepts a POST request with JSON data specifying the `announcement_id` and `vote_type` ('upvote' or 'downvote').
- Validates the vote type to ensure it is either 'upvote' or 'downvote'.
- Retrieves the currently logged-in student from the session.
- Fetches the corresponding `Student` and `TeacherAnnouncement` objects.
- Records the vote in the `AnnouncementVote` table.

- Computes the updated counts of upvotes and downvotes for the announcement.
- Returns a JSON response containing the status, message, and updated vote counts.

22.4 Security Weakness

This module contains a notable security vulnerability:

- It does not prevent a student from submitting multiple votes on the same announcement.
- As a result, a user can cast repeated votes, potentially skewing the vote counts.
- Exploiting this vulnerability can manipulate perceived popularity or importance of announcements.
- Proper mitigation would include checking for an existing vote by the student before creating a new entry or allowing vote updates instead of duplicates.