# Bobby - System architecture

Arun, Ganga, Sai Bhargav

September 2025

## 1 School Management System Architecture

The School Management System is a Django-based web application designed to manage students, teachers, classes, timetables, announcements, and academic records. The system implements role-based access control, where each participant (Admin, Teacher, Student) interacts with the system through a web interface. All communication occurs over standard HTTP/HTTPS, with JSON used for asynchronous operations such as voting on announcements. The system leverages Django sessions to maintain authentication and authorization state.

The architecture is modular, separating concerns into dedicated Django apps and views. Each module handles specific functionality, such as student registration, timetable management, announcements, and marks management. Data is stored in a relational database (e.g., SQLite, PostgreSQL) using Django's ORM. Security-critical modules, such as CAPTCHA validation and file upload handling, incorporate measures to prevent automated attacks, although some vulnerabilities are intentionally included for learning purposes.

### 1.1 Roles and Participants

#### 1.1.1 Administrator (Admin)

The Admin is responsible for managing the overall school system. Admins have the highest level of access and can perform the following actions:

- Add, edit, or delete subjects, teachers, classrooms, and timeslots.
- Approve or reject students and teachers based on their respective teacher registration and student application data.
- Generate the class timetable based on teacher availability, subjects, and class requirements.
- View all timetable entries with filters for class, subject, teacher, and day.
- Manage student academic records, including marks and grades.
- Monitor announcements, registration data, and system configurations.

- Create pin-board announcements, Monitor announcements and answers the comments
- Review system logs and handle error reports.

The Admin interacts with the system via the web interface and is authenticated using the `Admin_login` session decorator. All requests are verified against the session to ensure authorized access.

#### 1.1.2 Teacher

Teachers are responsible for delivering lessons, managing marks, interacting with student-related data and reviewing announcements. Their interactions include:

- Viewing their assigned timetable entries for specific days.
- Accessing subjects and class information relevant to their teaching assignments.
- Reviewing their own profile, uploading profile photo, document and modification of their own profile.
- Posting announcements that students can view and vote on.
- Uploading documents or resources, with file type and size validations.
- Recording student marks, assessments, and exam scores in the system.
- Reviewing pin board announcements and adding their valuable inputs or questions through comments.
- Handling errors in form submissions and logging any critical events.

Teachers are authenticated using the `Teacher_login` session decorator. They only see data related to their assignments, ensuring separation of concerns and data privacy.

#### 1.1.3 Student

Students are the end users who consume educational content, track academic performance, and interact with the system. Student actions include:

- Registering via a web form with CAPTCHA validation to prevent automated submissions.
- Viewing their class timetable and filtering entries by day.
- Reviewing their own profile, uploading profile photo and modification of their own profile.

- Viewing teacher announcements and voting via upvote/downvote mechanisms.
- Tracking personal information, class level, timetable assignments, and academic records.
- Accessing marks and grades assigned by teachers.
- Reviewing pin board announcements and adding their questions through comments
- Receiving notifications on errors or invalid submissions.

Students are authenticated using the `Student_login` session decorator. Access is restricted to data relevant to their class and assignments.

## 1.2 Workflow and Data Flow

The system operates in the following structured workflows:

- **Registration:** Students and Teachers submit registration forms, validated with CAPTCHA and optional file uploads. Errors during registration are logged and reported. Validated data is stored in the corresponding `StudentReg` or `TeacherReg` tables.
- **Student Application:** Students submit an application form prefilled with registration data (class, name, DOB, email) from the lastest entry of `StudentReg` table. They add required details such as nationality, blood type, address, parent info, previous school, and grades. The validated data are stored in the `StudentApplication` table. Errors during application are logged and reported.
- **Authentication:** Upon login, the system sets session variables to identify the user role (Admin, Teacher, Student) and restrict access to views accordingly. Failed login attempts are logged.
- **Approval Management:** Admin reviews student applications and teacher registrations to verify personal data, addresses, and documents. Valid requests are approved, creating user credentials and sending login details by email. Invalid requests are rejected and removed from the database. All actions are secured with session checks and recorded to ensure that only verified users gain access.
- **Timetable Generation:** Admin adds subjects, teachers, rooms, and timeslots, then generates the timetable using an algorithm that assigns teachers and rooms to periods. Conflicts are detected, reported, and logged.

- **Timetable Viewing:** Students and Teachers fetch timetable entries filtered by their class or assigned schedule. ORM errors or invalid queries are logged.
- **Profile Management:** Students can view and update their personal details, including address and profile photo, while key fields such as name, email, and date of birth remain read-only. Teachers can edit most profile details, upload documents, and photos, but cannot change email or date of birth. Admins can view their own profile, but cannot edit any fields. All changes are securely saved in the database, with session checks enforcing role-based access.
- **Marks Management:** Teachers enter marks for students for assignments, exams, or assessments. Admins can review or update marks as needed. Students can view their marks and calculate performance summaries. Errors in marks entry are validated and logged.
- **Pin board Management:**Admin posts official announcements, which students and teachers can view. Teachers and students can comment on notices, while all entries are shown newest first with pagination. Admin can respond to comments. Display names are resolved from session-linked accounts. All actions are session-controlled and logged.
- **Announcements:** Teachers post announcements. Students view announcements and cast votes, which are tracked in the `AnnouncementVote` table. JSON is used for asynchronous vote updates. Invalid vote submissions are logged.
- **Data Integrity and Security:** The system enforces session-based access control. Modules implement additional validation (CAPTCHA, file type checks). Certain vulnerabilities are intentionally included for educational purposes (CAPTCHA replay, duplicate voting). All exceptions and critical operations are logged for auditing.

## 1.3 System Communication and Integration

All modules communicate internally through Django's ORM and request-response cycle. Asynchronous operations, such as voting on announcements, use AJAX with JSON payloads. Session management ensures role-specific access control.

- Admin, Teacher, and Student communicate via HTTP requests to Django views.
- JSON is used for client-server interactions in interactive features (e.g., voting, timetable updates, marks submission).
- ORM queries ensure referential integrity across models like `Student`, `Teacher`, `TimetableEntry`, `AnnouncementVote`, and `Marks`.

- Security-sensitive operations, including registration, voting, and marks updates, are logged and validated against session credentials.
- Exception handling is implemented across modules to ensure errors are captured, reported, and logged for auditing purposes.

### 1.4 Security Considerations

The system-level security considerations include:

- Role-based access control using session decorators (`Admin_login`, `Teacher_login`, `Student_login`).
- Input validation for forms, file uploads, and academic records.
- CAPTCHA validation for automated bot prevention (with intentional replay vulnerability for study purposes).
- Vote integrity concerns, as duplicate votes are currently possible (educational vulnerability).
- Data privacy, ensuring students only access their own marks and timetables.
- Optimized ORM queries to prevent unintentional data leaks.
- Logging of critical operations, errors, and exceptions for auditing and accountability.
- Error handling mechanisms across workflows to provide clear feedback and maintain system stability.

## 2 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.

## 2.1 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`.

## 2.2 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.

## 2.3 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.

### 2.4 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.

## 3 Student Application Module

The `student_application.py` module implements the core functionality for submitting a student application. It integrates with existing student registration records to prefill form data where available and stores validated applications in the StudentApplication model. This feature provides the entry point for new students to apply to the school system.The main code module is located at `/login/view/student_application.py`. Although no deliberate vulnerability is introduced here, insecure coding practices may lead to issues such as unvalidated input, stored XSS, or weak data handling. The feature therefore requires careful form validation and safe template rendering.

### 3.1 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.
- Custom models from *login.models*: `StudentApplication` for storing final application data and `StudentReg` for storing student registration data; the latest record is used for pre-filling.
- Custom forms from *login.forms*: `StudentApplicationForm` for validating and cleaning submitted application data.

### 3.2 Prefill Logic

The view fetches the most recent `StudentReg` record using:

```
latest_reg = StudentReg.objects.last()
```

If present, relevant fields (name, Date of birth, gender, email, class level) are pre-populated into the form. Gender codes (M, F, O) are mapped into human-readable strings. This feature improves user experience by reducing redundant typing.

### 3.3 Main Function: `student_application_view`

The function provides two request flows:

- **GET request**
  - Initializes a blank `StudentApplicationForm`.
  - If a `StudentReg` record exists, fields are prefilled with existing values.
  - Renders the `studentapplication.html` template with the form.
- **POST request**
  - Processes form submission via `StudentApplicationForm`.
  - On successful validation:
    - Extracts cleaned data.
    - Creates a new `StudentApplication` record with the supplied details (student information, parent info, previous school information).
    - Adds a success message and redirects to the index page.
  - On invalid submission, the form is redisplayed with error messages.

**POST Request**

- Processes submitted form data using `StudentApplicationForm(request.POST)`.
- If the form is valid:
  - Extracts `form.cleaned_data`.
  - Creates a `StudentApplication` record with explicit field mapping:
    - **Student information:** first name, last name, Date of birth, gender, email, class level, mobile, nationality, blood group.
    - **Student address:** street, house, city, state (optional), postal.
    - **Parent information:** first name, last name, email, mobile, emergency contact.
    - **Parent address:** street, house, city, state (optional), postal.

    - **Previous school info:** school name, class/grade, TC number, street, house, city, state, postal.
  - Adds a success message and redirects to the index page.
- If the form is invalid, the form with errors is re-rendered for user correction.

### 3.4 Security Considerations

- Only saves `form.cleaned_data`, ensuring that only validated input is persisted.
- Assumes templates correctly escape user-provided content to prevent XSS.
- Prefill uses the latest `StudentReg` record; ensure this behavior is acceptable and does not expose unintended data.
- No deliberate vulnerabilities are present in this module.

### 3.5 Security Considerations

- Only saves `form.cleaned_data`, ensuring that only validated input is persisted.
- Assumes templates correctly escape user-provided content to prevent XSS.
- Prefill uses the latest `StudentReg` record; ensure this behavior is acceptable and does not expose unintended data.
- No deliberate vulnerabilities are present in this module.

## 4 Student Approval Module

The `student_approval.py` module enables administrators to manage pending student applications. Admins can review submitted applications, approve them (creating both a student record and a corresponding Login account), or reject them (deleting the application record).The main code module is located at:`/login/view/student_approval.py`

This module demonstrates secure session handling via a decorator and enforces role-based access control. While the code itself is functional and secure, passwords are generated as plain tokens and must be hashed before production deployment.

### 4.1 Dependencies

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

- **Python standard libraries**

  - `secrets`, `string`, `random`, `hashlib` — for generating usernames, passwords, and hashing (simple MD5 in current implementation).

- **Django libraries**
  - `django.shortcuts.render`, `redirect`, `get_object_or_404` — template rendering, redirection, and object fetching.
  - `django.contrib.messages` — for displaying flash messages to the admin.

- **Project models**
  - `StudentApplication` model for pending student applications.
  - `Student` model for approved student records.
  - `Login` model for user authentication table, storing username, password, role, and email.

- **Project decorators**
  - `session_required` (from `login.views`) which enforces that only authenticated admins can access these views.

## 4.2 Session Decorator

All views in this module are decorated with: `@session_required('Admin_login')`. This ensures that only authenticated administrators can access student approval or rejection functionalities. If a session is invalid or missing, the user is redirected to the application index.

## 4.3 Main Functions

### Admin Workflows

#### List Pending Students (`student_approval`)

- Retrieves all `StudentApplication` records.
- Filters out applications already approved (by checking existing emails in `Login` with role `'Student'`).
- Renders `login/student_approval.html` template with the list of pending students.

**Approve Student (`approve_student`)**

- Accepts a `student_id`.
- Fetches the corresponding `StudentApplication` record.
- Checks whether the student email is already approved; if so, issues a warning message.
- Generates a random username (3 letters + 3 digits) and a random password token (8-character URL-safe string).
- Creates a new `Student` record using all application details (personal, address, parent info).
- Creates a new `Login` record for authentication purposes, with plain token as password (note: should be hashed in production).
- Displays a success message and redirects back to pending applications.

**Reject Student (`reject_student`)**

- Accepts a `student_id`.
- Fetches the corresponding `StudentApplication` record.
- Deletes the record.
- Shows an informational message and redirects to the pending applications page.

## 4.4 Security Considerations

No intentional vulnerabilities exist in this module, though production deployments must ensure proper password hashing and possibly audit logs for administrative actions.

- **Session enforcement:** All views require `Admin_login` session, preventing unauthorized access.
- **Data integrity:** Explicit field mapping ensures all relevant student and parent information is transferred correctly from `StudentApplication` to `Student`.
- **Duplicate check:** Approval function ensures the same email cannot be approved multiple times.
- **Deletion safety:** Rejection permanently deletes the student application record.

# 5 Teacher Approval Module

The `teacher_approval.py` module enables administrators to manage pending teacher registrations. Admins can review submitted registrations (`TeacherReg`), approve them (creating both a `Teacher` record and a corresponding `Login` account), or reject them (deleting the registration record). The main code module is located at: `/login/view/teacher_approval.py`

This module uses secure session enforcement and explicit approval logic. While it is secure in its basic operation, care must be taken to handle file uploads (teacher documents) safely.

## 5.1 Dependencies

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

- **Python standard libraries**
  - `secrets`, `string`, `random`, `hashlib` for generating usernames, passwords, and hashing (simple MD5 in current implementation).
- **Django libraries**
  - `django.shortcuts.render`, `redirect`, `get_object_or_404` template for rendering, redirection, and object fetching.
  - `django.contrib.messages` for displaying flash messages to the admin.
- **Project models**
  - `TeacherReg` model for pending teacher registrations.
  - `Teacher` model for approved teacher records.
  - `Login` model for user authentication table, storing username, password, role, and email.
- **Project decorators**
  - `session_required` (from `login.views`) which enforces that only authenticated admins can access these views.

## 5.2 Session Decorator

All views in this module are decorated with: `@session_required('Admin_login')`. This enforces that only authenticated administrators can access teacher approval or rejection views. Invalid sessions are redirected to the application index.

## 5.3 Main Functions

**Admin Workflows**

**List Pending Teachers (`teacher_approval`)**

- Retrieves all `TeacherReg` records.
- Filters out registrations already approved (by checking existing emails in `Login` with role `'Teacher'`).
- Renders `login/teacher_approval.html` template with the list of pending teachers.

**Approve Teacher (`approve_teacher`)**

- Accepts a `teacher_id`.
- Fetches the corresponding `TeacherReg` record.
- Checks whether the teacher email is already approved; if so, issues a warning message.
- Generates a random username (3 letters + 3 digits) and a random password token (8-character URL-safe string).
- Creates a new `Teacher` record using registration details: first name, last name, DOB, gender, email, and uploaded document.
- Creates a new `Login` record for authentication purposes, with the generated plain password (**should be hashed in production**).
- Displays a success message and redirects back to the pending registrations page.

**Reject Teacher (`reject_teacher`)**

- Accepts a `teacher_id`.
- Fetches the corresponding `TeacherReg` record.
- Deletes the record.
- Shows an informational message and redirects to the pending registrations page.

## 5.4 Security Considerations

No intentional vulnerabilities exist in this module, though production deployments must ensure proper password hashing and possibly audit logs for administrative actions.

- **Session enforcement:** All views require an admin session, preventing unauthorized access.

- **File safety:** Teacher documents (`document` field) must be handled carefully when serving to prevent unauthorized access.
- **Duplicate check:** Approval function ensures the same email cannot be approved multiple times.
- **Deletion safety:** Rejection permanently deletes the teacher registration record.

# 6 Student Profile Module

The `student_profile.py` module enables students to view and update their personal profile. It integrates data from both the `Student` model and `StudentApplication` model, allowing students to update fields selectively. Profile photo upload is supported, with changes applied only to modified fields. The main code module is located at: `/login/view/student_profile.py`

This module demonstrates secure session handling, selective field updates, and file upload support. No deliberate vulnerabilities are present, assuming proper template escaping and safe file handling.

## 6.1 Dependencies

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

- **Django libraries**
  - `django.shortcuts.render`, `redirect`, `get_object_or_404` are used for template rendering, redirection, and safe object fetching.
  - `django.contrib.messages` is used for displaying flash messages to the user.
- **Project models**
  - `Login` is a authentication table to retrieve the current user session.
  - `Student` is a main student profile model.
  - `StudentApplication` is a source for prefilled student application data.
- **Project forms**
  - `StudentProfileForm` is responsible for validating user-submitted profile updates.
- **Project decorators**
  - `session_required` ensures only authenticated students can access the profile.

## 6.2 Session Decorator

The view is decorated with: `@session_required('Student_login')`. This ensures that only logged-in students can access the profile page. Invalid or missing sessions result in a redirect to the student index page.

## 6.3 Main Function: `student_profile`

### GET Request

- Retrieves the logged-in student's `Login` object via username stored in session.
- Fetches corresponding `Student` and `StudentApplication` objects.
- Prepares initial form data using `StudentApplication` fields.
- Renders `login/student_profile.html` with:
  - `form` — prefilled `StudentProfileForm`.
  - `profile_photo` — current profile photo from `Student`.

### POST Request

- Instantiates `StudentProfileForm(request.POST, request.FILES)`.
- On valid form submission:
  - Extracts `cleaned_data`.
  - Compares each field against `Student` and `StudentApplication` objects to track changes.
  - Updates only fields that have changed:
    - **Student fields** (including profile photo if uploaded).
    - **StudentApplication fields** (if applicable).
  - Saves changes selectively using `update_fields` for efficiency.
  - Displays success message and redirects to the profile page.
- On invalid submission:
  - Form is redisplayed with errors for correction.

## 6.4 Data Mapping

- **Student model updates**
  - Any field present in `StudentProfileForm` that exists in `Student` and differs from the current value.
  - `profile_photo` handled separately; saved if a new file is uploaded.

- **StudentApplication updates**
  - The fields that exist in `StudentApplication` and differ from the values submitted are updated via `.update()`.
- **Read-only fields**
  - Fields marked as read-only in the form are never modified.

### 6.5 Security Considerations

Intentional vulnerabilities are not present in this module.

- **Session enforcement:** Only authenticated students can access this view.
- **File uploads:** Profile photo files must be stored securely, and the files that are served should prevent unauthorized access.
- **Template safety:** Text fields rendered in templates must be escaped to prevent XSS.
- **Selective updates:** Only modified fields are saved, preventing unintended overwrites.
- **Data consistency:** Synchronizes changes across both `Student` and `StudentApplication` tables.

## 7 Teacher Profile Module

The `teacher_profile.py` module allows teachers to view and update their profile information, including uploading profile photos and documents. It also provides a document download endpoint.

This module is intentionally designed with insecure coding practices for lab/demo purposes. Vulnerabilities such as CSRF bypass and Insecure Direct Object Reference (IDOR) are introduced to demonstrate real-world attack scenarios.The main code is located at: `/login/view/teacher_profile.py`

### 7.1 Dependencies

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

- **Django libraries**
  - `django.shortcuts.render`, `redirect` for template rendering and redirection.
  - `django.contrib.messages` for flash messages for user feedback.
  - `django.http.FileResponse`, `Http404` for serving file downloads and handling errors.

  - `django.views.decorators.csrf.csrf_exempt` for disables CSRF protection.

- **Python standard libraries**
  - `os` for file handling (checking existence, deleting old files).

- **Project models**
  - `Login` table for teacher authentication and linking accounts.
  - `TeacherReg` table for original teacher registration data (with uploaded documents).
  - `Teacher` table for permanent teacher profile data.

- **Project decorators**
  - `session_required('Teacher_login')` is ensures only authenticated teachers access these views.

- **Project forms**
  - `TeacherProfileForm` for updating teacher profile fields.

## 7.2 Session Decorator

The view is decorated with: `@session_required('Teacher_login')`. This ensures that only logged-in teachers can access the profile page. Invalid or missing sessions result in a redirect to the teacher index page.

## 7.3 Main Function: `teacher_profile`

`teacher_profile(request)`

- **Purpose:** View and update teacher profile data.
- **Features:**
  - Loads teacher information from both `TeacherReg` and `Teacher`.
  - Allows updating first name, last name, document, and profile photo.
  - Deletes old files (document/photo) when replaced with new ones.
  - Displays feedback messages on successful or failed updates.
- **Vulnerability:**
  - IDOR — accepts `?username=KP0001` to target another teacher's data instead of using the session user.
  - CSRF disabled with `@csrf_exempt`, allowing forged requests.

`download_teacher_document(request)`

- **Purpose:** Allows teachers to download their uploaded documents.
- **Features:**
  - Retrieves the document from either `TeacherReg` or `Teacher`.
  - Returns the file as an attachment (`FileResponse`).
  - Handles missing files gracefully with error messages.
- **Vulnerability:**
  - IDOR — accepts `?username=KP0001` to download another teacher's document.
  - CSRF disabled, allowing attackers to trick teachers into unintended downloads.

### 7.4 Security Weakness

This module deliberately demonstrates insecure practices:

1. **CSRF bypass:** Both profile update and document download are exempt from CSRF checks.
2. **IDOR:** Query parameter `?username=` allows attackers to read or modify other teachers' data.
3. **File upload restrictions bypass:**
   - No server-side validation exists for file type or size.
   - Users can upload files larger than intended or with different types (e.g., PHP, JS), even if the frontend restricts input to PDFs under 1MB.
   - This could lead to malicious uploads, file execution, or server compromise.
4. **File handling risks:** Uploaded files are deleted/replaced directly on the filesystem, which may lead to race conditions or unintended deletions.
5. **Authorization gap:** Authorization is not strictly tied to the session user.

## 8 Admin Profile Module

The `admin_profile.py` module allows administrators to view their profile information. Access is controlled via session-based authentication. The main code is located at: `/login/view/admin_profile.py`

## 8.1 Dependencies

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

- **Django components**
  - `django.shortcuts.render`, `redirect` for template rendering and redirection.
  - `django.contrib.messages` for feedback messages for errors or notifications.
- **Project models**
  - `Login` table stores authentication credentials and role information.
- **Project decorators**
  - `session_required('Admin_login')` ensures only authenticated admins can access the profile page.

## 8.2 Session Decorator

The `session_required('Admin_login')` decorator enforces role-based access control. Only users with an active admin session can view the profile. Invalid sessions are flushed, and the user is redirected to the admin login page.

## 8.3 Main Function

**admin_profile(request)**

- **Purpose:** Display the admin's profile information.
- **Logic:**
  - Retrieves the current admin username from the session (`login_username`).
  - Queries the `Login` table for a user with role 'Admin'.
  - Renders `admin_profile.html` with the retrieved data.
  - If no record is found, displays an error message and redirects to the admin login page.

## 8.4 Security Considerations

- **Session enforcement:** Only authenticated admins can access this page.
- **Data exposure:** Only minimal profile information is shown.
- **No deliberate vulnerabilities:** This module is considered secure; improper session handling could expose admin data.

# 9 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.

## 9.1 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`.

## 9.2 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.

## 9.3 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.

### 9.4 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.

## 10 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.

### 10.1 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.

### 10.2 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.

### 10.3 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.

### 10.4 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.

## 11 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.

### 11.1 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.

## 11.2 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.

## 11.3 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.

## 11.4 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.

# 12 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.

## 12.1 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.

## 12.2 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.

### 12.3 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.

### 12.4 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.

## 13 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.

### 13.1 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.

## 13.2 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.

## 13.3 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.

### 13.4 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.

## 14 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.

### 14.1 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.

## 14.2 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.

## 14.3 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.

# 15 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*.

## 15.1 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`.

## 15.2 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.

## 15.3 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.

## 15.4 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.

# 16 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*.

## 16.1 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.

## 16.2 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.

## 16.3 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.

## 16.4 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.

# 17 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.

## 17.1 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.

## 17.2 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.

## 17.3 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.

## 17.4 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.

# 18 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*.

## 18.1 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`.

## 18.2 Session Decorator

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

## 18.3 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.

## 18.4 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.

### 18.5 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.

## 19 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*.

### 19.1 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.

## 19.2 Session Decorator

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

## 19.3 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.

## 19.4 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.

# 20 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*.

## 20.1 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`.

## 20.2 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.

## 20.3 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.

### 20.4 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.

## 21 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*.

### 21.1 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.

## 21.2 Session Decorator

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

## 21.3 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.

## 21.4 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.

# 22 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*.

## 22.1 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`.

## 22.2 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'`.

## 22.3 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).

## 22.4 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.

# 23 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`.

## 23.1 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).

## 23.2 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.

## 23.3 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`.

## 23.4 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.

# 24 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`.

## 24.1 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.

## 24.2 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.

## 24.3 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.

### 24.4 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.

## 25 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$_a$*nnouncement.py*.

### 25.1 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.

### 25.2 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.

### 25.3 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.

### 25.4 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.

## 26 Admin Pinboard Module

The `admin_pinboard.py` module allows administrators to create and view pinboard announcements. The main code is located at: `/login/view/admin_pinboard.py` functions: `create_pinboard`, `pinboard_list_admin`.

## 26.1 Dependencies

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

- **Django components**
  - `django.shortcuts.render`, `redirect` for template rendering and redirection.
  - `django.contrib.messages` for user notifications for successful or failed actions.
  - `django.core.paginator.Paginator` for handling paginated display of announcements.
  - `django.views.decorators.cache.never_cache` to prevent caching of pinboard pages.
- **Project models**
  - `Pinboard` tablestores announcements and metadata.
  - `Teacher` and `Student` tables are used to display creator names.
- **Project forms**
  - `PinboardForm` validates admin input for announcements.
- **Project decorators**
  - `session_required('Admin_login')` — ensures only admins can create/view announcements.

## 26.2 Session Decorator

The `session_required('Admin_login')` decorator enforces role-based access control. Only authenticated admins can create or view pinboard announcements. Invalid sessions are flushed, and the user is redirected to the admin login page.

The `@never_cache` decorator ensures that paginated announcements are always fetched fresh, preventing sensitive data from being stored in the cache.

## 26.3 Main Functions

`create_pinboard(request)`

- **Purpose:** Allow admins to post new announcements to the pinboard.
- **Logic:**
  - Handles POST requests containing `PinboardForm` data.
  - Valid form data is saved, with the `created_by` field set from the current session username.
  - Provides success feedback and redirects to the pinboard list.
  - For GET requests, renders an empty form for input.

`pinboard_list_admin(request)`

- **Purpose:** Display a paginated list of pinboard announcements for the admin.
- **Logic:**
  - Calls the common renderer `pinboard_list_common` with role 'Admin'.
  - Announcements are sorted in reverse chronological order.
  - Display names are resolved using `get_display_name`, combining first/last names of students or teachers, or a default name for admins.
  - Pagination is set to 10 announcements per page.
  - The back link is dynamically set to the admin dashboard URL.

`pinboard_list_common(request, role)`

- **Purpose:** Shared renderer for pinboard listings (Admin, Teacher, Student).
- **Logic:**
  - Annotates announcements with `display_name` based on creator.
  - Handles pagination.
  - Determines dashboard URL for navigation based on role.
  - Renders the common template `pinboard_list.html`.

`get_display_name(username)`

- **Purpose:** Helper function to display a readable name for the announcement creator.
- **Logic:**
  - Checks if username is an admin (`ADM*`).
  - Queries the `Student` or `Teacher` model for matching names.
  - Defaults to the raw username if no match found.

## 26.4 Security Considerations

Intentional vulnerabilities are not present in this module.

- **Template escaping:** Ensure announcement text is escaped to prevent stored Cross-Site Scripting (XSS).
- **Session enforcement:** Only authenticated admins can access creation and listing views.

- **Pagination:** Prevents overloading the interface with large datasets.
- **No explicit file uploads:** Not applicable here, so no file validation needed.

# 27 Student Pinboard Module

The `student_pinboard.py` module allows students to view the pinboard announcements posted by admins or other authorized users. The main code is located at: `/login/view/student_pinboard.py` — function: `pinboard_list_student`.

## 27.1 Dependencies

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

- **Django components**
  - `django.shortcuts.render` for template rendering.
  - `django.core.paginator.Paginator` for handling paginated display of announcements.
  - `django.views.decorators.cache.never_cache` to ensure fresh data for each page load.
- **Project models**
  - `Pinboard` table which stores announcements.
  - `Teacher` and `Student` tables are used to display creator names.
- **Project decorators**
  - `session_required('Student_login')` ensures only logged-in students can view announcements.

## 27.2 Session Decorator

The `session_required('Student_login')` decorator enforces role-based access control. Only authenticated students can view the pinboard list. Invalid sessions are flushed, and the user is redirected to the student login page.

The `@never_cache` decorator ensures that paginated announcements are always fetched fresh, preventing caching of old announcements.

## 27.3 Main Functions

`pinboard_list_student(request)`

- **Purpose:** Display a paginated list of pinboard announcements for students.

- **Logic:**
  - Calls the common renderer `pinboard_list_common` with role 'Student'.
  - Announcements are sorted in reverse chronological order.
  - Display names are resolved using `get_display_name`, combining first/last names of students or teachers, or a default name for admins.
  - Pagination is set to 10 announcements per page.
  - The back link is dynamically set to the student dashboard URL.

`pinboard_list_common(request, role)`

- **Purpose:** Shared renderer for pinboard listings (Admin, Teacher, Student).
- **Logic:**
  - Annotates announcements with `display_name` based on the creator.
  - Handles pagination.
  - Determines dashboard URL for navigation based on role.
  - Renders the common template `pinboard_list.html`.

`get_display_name(username)`

- **Purpose:** Helper function to display a readable name for the announcement creator.
- **Logic:**
  - Checks if username is an admin (`ADM*`).
  - Queries the `Student` or `Teacher` model for matching names.
  - Defaults to the raw username if no match found.

## 27.4 Security Considerations

Intentional vulnerabilities are not present in this module.

- **Read-only access:** Students cannot create, edit, or delete announcements.
- **Template escaping:** Ensure announcement text is escaped to prevent stored XSS.
- **Session enforcement:** Only authenticated students can access this view.
- **Pagination:** Prevents overloading the interface with large datasets.

# 28 Teacher Pinboard Module

The `teacher_pinboard.py` module allows teachers to view the pinboard announcements posted by admins or other authorized users. The main code is located at: `/login/view/teacher_pinboard.py` — function: `pinboard_list_teacher`.

## 28.1 Dependencies

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

- **Django components**
  - `django.shortcuts.render` — template rendering.
  - `django.core.paginator.Paginator` — handles paginated display of announcements.
  - `django.views.decorators.cache.never_cache` — ensures fresh data for each page load.
- **Project models**
  - `Pinboard` — stores announcements.
  - `Teacher` and `Student` — used to display creator names.
- **Project decorators**
  - `session_required('Teacher_login')` — ensures only logged-in teachers can view announcements.

## 28.2 Session Decorator

The `session_required('Teacher_login')` decorator enforces role-based access control. Only authenticated teachers can view the pinboard list. Invalid sessions are flushed, and the user is redirected to the teacher login page.

The `@never_cache` decorator ensures that paginated announcements are always fetched fresh, preventing caching of old announcements.

## 28.3 Main Functions

`pinboard_list_teacher(request)`

- **Purpose:** Display a paginated list of pinboard announcements for teachers.
- **Logic:**
  - Calls the common renderer `pinboard_list_common` with role 'Teacher'.
  - Announcements are sorted in reverse chronological order.

  - Display names are resolved using `get_display_name`, combining first/last names of students or teachers, or a default name for admins.
  - Pagination is set to 10 announcements per page.
  - The back link is dynamically set to the teacher dashboard URL.

`pinboard_list_common(request, role)`

- **Purpose:** Shared renderer for pinboard listings (Admin, Teacher, Student).
- **Logic:**
  - Annotates announcements with `display_name` based on the creator.
  - Handles pagination.
  - Determines dashboard URL for navigation based on role.
  - Renders the common template `pinboard_list.html`.

`get_display_name(username)`

- **Purpose:** Helper function to display a readable name for the announcement creator.
- **Logic:**
  - Checks if username is an admin (`ADM*`).
  - Queries the `Student` or `Teacher` model for matching names.
  - Defaults to the raw username if no match found.

### 28.4 Security Considerations

- **Read-only access:** Teachers cannot create, edit, or delete announcements.
- **Template escaping:** Ensure announcement text is escaped to prevent stored XSS.
- **Session enforcement:** Only authenticated teachers can access this view.
- **Pagination:** Prevents overloading the interface with large datasets.

## 29 Pinboard Detail And Comments Module

The `pinboard_detail.py` module allows users to view a single pinboard announcement and its associated comments. Logged-in users (Admin, Teacher, Student) can post comments. The main code is located at: `/login/view/pinboard_detail.py`.

## 29.1 Dependencies

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

- **Django components**
  - `django.shortcuts.render, redirect, get_object_or_404` — for template rendering and record lookup.
  - `django.views.decorators.cache.never_cache` — ensures the page always shows fresh comments.
- **Project models**
  - `Pinboard` — announcement model, with related comments.
  - `Teacher` and `Student` — used to display the commenter's name.
- **Project forms**
  - `PinboardCommentForm` — handles comment submission.

## 29.2 Session Decorator

- `@never_cache` ensures users see the latest comments without relying on browser cache.
- Comment posting relies on session keys (`Admin_login`, `Teacher_login`, `Student_login`) to associate the comment with the correct user.
- If no valid session exists, comments are marked as Anonymous.

## 29.3 Main Functions

`pinboard_detail(request, pk)`

- **Purpose:** Display a single pinboard announcement and all its comments.
- **Logic:**
  - Retrieves the announcement using its primary key (`pk`).
  - Populates `display_name` for the creator and all commenters using `get_display_name`.
  - Orders comments chronologically.
  - Handles POST requests to add a new comment:
    - Determines the current user based on session.
    - **VULNERABLE:** Raw user input is saved directly, allowing stored XSS if templates do not escape HTML.
  - Provides `dashboard_url` for a back link, depending on user role.
  - Renders `pinboard_detail.html`.

`get_display_name(username)`

- **Purpose:** Resolve a readable display name for the announcement/comment creator.
- **Logic:**
  - Returns "Administration Office" for admin usernames (`ADM*`).
  - Fetches the first and last name from `Student` or `Teacher` models.
  - Defaults to the raw username if no match is found.

## 29.4 Security Considerations

- **Stored XSS risk:** Comments are saved directly without sanitization. Any HTML or `<script>` tags will execute when viewed.
- **Template escaping required:** Always escape announcement and comment fields in templates to prevent XSS.
- **Session-based user tracking:** Ensures comments are attributed to the logged-in user; falls back to "Anonymous" otherwise.
- **Read/write access:** Only logged-in users can comment; all others have read-only access.