Backend Document

Introduction:

This comprehensive backend documentation guide aims to provide a detailed overview of the functionalities and architecture of our course management project and it’s backend functionalities. The Academix Portal is designed to streamline the management of courses, assignments, announcements, and student interactions within an academic environment.

Project Goals:

- Efficient course administration
- Seamless student enrollment and interaction
- Simplified assignment creation and grading
- Centralized platform for announcements and queries
- User-friendly interfaces for both administrators and students

Project Architecture:

The Academix Portal follows a modular architecture, utilizing the Django web framework for backend development and incorporating frontend technologies for a dynamic and responsive user experience.

Frontend:

The frontend of the application is developed using the following technologies:

- **HTML, CSS, JavaScript:**
  - Responsible for creating the structure, style, and interactivity of the user interface.
- **Django Templates:**
  - Utilized for rendering dynamic content and embedding backend data into HTML.

Backend:

The backend of the application leverages the Django framework, Python, and a relational database for data storage.

- **Django Framework:**
  - Manages routing, views, models, and integrates with the frontend for seamless user interactions.
- **Database Integration:**
  - Utilizes Django models to define the structure of the database, including entities such as courses, announcements, assignments, and student submissions. Sqlite and AWS database services are used to store data and files.
- **Authentication and Authorization:**
  - Implements user authentication and authorization mechanisms to control access to different functionalities based on user roles.

Project Structure:

The project is organized into various components, each serving a specific purpose:

- **Views:**
  - Handles user requests and orchestrates the flow of data between the frontend and backend.
- **Models:**
  - Defines the structure of the database and represents entities such as courses, announcements, assignments, and user profiles.
- **Templates:**
  - Contains HTML templates for rendering dynamic content and presenting data to users.
- **Static Files:**
  - Stores static assets such as CSS stylesheets, JavaScript files, and images.

Key Functionality:

1. **Course Management:**
   - Creation, editing, and deletion of courses.
   - Enrollment of students in courses.
2. **Announcements:**
   - Posting, editing, and deletion of announcements within specific courses.
3. **Assignments:**
   - Creation, editing, and deletion of assignments with associated deadlines.
   - Submission and grading of student assignments.

4. **Materials:**
    - Uploading and organizing course materials.
5. **Queries and Feedback:**
    - Student queries and feedback mechanisms for improved communication.
6. **User Profiles:**
    - Viewing and managing student profiles.
7. **Search Functionality:**
    - Search features for courses, student lists, and dashboards.

Backend Documentation

Technology Stack

- **Django**: The backend framework chosen for its robustness and scalability.
- **Python**: Utilized as the primary programming language for backend logic.
- **SQLite**: The default database engine, chosen for simplicity and ease of deployment.

Project Structure

Academix_Portal App

Models

1. `student_profile`: Stores information about student profiles.
2. `faculty_profile`: Stores details about faculty profiles.
3. `Course`: Defines the structure for storing course-related information.
4. `Assignment`: Contains details about assignments within a course.
5. `Submission`: Holds data about student submissions.
6. `Announcements`: Manages course announcements.
7. `Material`: Stores information about course materials.
8. `feedback`: Captures user feedback.
9. `query`: Manages student queries.

Views

1. **User Management Views**:
   - `home`: Renders the homepage.
   - `register`: Handles user registration.
   - `login_func`: Manages user logins.
   - `verifyRegistration`: Verifies user registration using OTP.
   - `log_out`: Logs out the user.
2. **Course Management Views**:
   - `admindashboard`: Renders the admin dashboard.
   - `mycourse`: Displays course details.
   - `mycourse_search`: Handles course search functionality.
   - `edit_course`: Allows editing of course details.
   - `announcements`: Displays course announcements.
   - `addannouncement`: Manages the addition of announcements.
   - `delete_announcement`: Deletes course announcements.
   - `createassignment`: Creates assignments within a course.
   - `add_assignment`: Adds assignments to a course.
   - `view_assignments`: Displays assignments for a course.
   - `edit_assignment`: Allows editing of assignment details.
   - `delete_assignment`: Deletes assignments.
   - `add_submission`: Manages the addition of assignment submissions.

- `edit_submission`: Allows editing of submitted assignments.
- `view_students_submission`: Displays submissions by students.
- `grade_student_submission`: Allows grading of student submissions.
- `materials`: Displays course materials.
- `addmaterial`: Manages the addition of course materials.
- `view_query`: Displays student queries.
- `add_query`: Manages the addition of queries.
- `reply_query`: Allows replying to student queries.
- `fb_response`: Handles user feedback.
- `addcourse`: Adds a new course.
- `add_course_to_user`: Enrolls a user in a course.
- `students_assignment`: Manages assignments for students.

3. **Other Views**:
   - `coursedashboard`: Renders the course dashboard.
   - `coursedashboard_search`: Handles course dashboard search functionality.
   - `actions`: Manages general actions.
   - `actions_student`: Handles actions specific to students.
   - `student_list`: Displays the list of students in a course.
   - `student_list_search`: Handles student list search functionality.
   - `view_profile`: Displays the profile of a student.

URLs

Defines URL patterns for routing requests to the appropriate views.

```
urlpatterns = [
    path('error',views.error, name="error"),
    path('', views.home, name="HomePage"),
    path('login/<str:loginid>', views.login_func, name="login_func"),
    path('register', views.register, name="Register"),
    path('mycourse', views.mycourse, name="mycourse"),
    path('mycourse/search', views.mycourse_search, name="mycourse_search"),
    path('mycourse/<str:course_id>/editcourse', views.edit_course, name="edit_course"),
    path('mycourse/<str:course_id>/unenrollcourse', views.unenroll, name="unenroll"),
    path('mycourse/<str:course_id>/deletecourse', views.deletecourse, name="deletecourse"),
    path('student_profile', views.students_profile, name="students_profile"),
    path('mycourse/<str:course_id>', views.announcements, name="announcements"),
    path('mycourse/<str:course_id>/addannouncement', views.addannouncement, name="addannouncement"),
    path('mycourse/<str:course_id>/deleteannouncement/<str:ann_id>', views.delete_announcement, name="delete_ann
    path('mycourse/<str:course_id>/createassignment', views.createassignment, name="createassignment"),
    path('mycourse/<str:course_id>/addassignment', views.add_assignment, name="add_assignment"),
    path('mycourse/<str:course_id>/viewassignments', views.view_assignments, name="view_assignments"),
    path('mycourse/<str:course_id>/<str:name>/editassignment', views.edit_assignment, name="edit_assignment"),
    path('mycourse/<str:course_id>/<str:name>/deleteassignment', views.delete_assignment, name="delete_assignmen
    path('mycourses/<str:course_id>/<str:name>/addsubmission', views.add_submission, name="add_submission"),
    path('mycourses/<str:course_id>/<str:name>/editsubmission', views.edit_submission, name="edit_submission"),
    path('mycourse/<str:course_id>/<str:name>/viewstudentssubmission', views.view_students_submission, name="vie
    path('mycourse/<str:course_id>/<str:name>/viewstudentssubmission/<str:sub_id>/grade', views.grade_student_su
    path('mycourse/<str:course_id>/materials', views.materials, name="materials"),
    path('mycourse/<str:course_id>/addmaterial', views.addmaterial, name="addmaterial"),
    path('mycourse/<str:course_id>/viewquery', views.view_query, name="view_query"),
    path('mycourse/<str:course_id>/addquery', views.add_query, name="add_query"),
    path('mycourse/<str:course_id>/replyquery/<str:qid>', views.reply_query, name="reply_query"),
    path('mycourse/<str:course_id>/feedback', views.fb_response, name="feedback"),
    path('coursedashboard',views.coursedashboard,name="coursedashboard"),
    path('coursedashboard/search',views.coursedashboard_search,name="coursedashboard_search"),
    path('otp_ver', views.verifyRegistration, name="otp_ver"),
    path('addcourses' , views.addcourse , name = "Add_course"),
    path('enroll/<str:course_id>' , views.add_course_to_user , name="enroll"),
    path('mycourse/<str:course_id>/studentlist',views.student_list , name= "student_list"),
    path('mycourse/<str:course_id>/studentlist/search',views.student_list_search , name= "student_list_search"),
    path('mycourse/<str:course_id>/studentlist/<str:id>/viewprofile',views.view_profile , name= "view_profile"),
    path('student_profile/change',views.update_profile, name="update_profile"),
    path('<str:course_id>/<str:student>',views.students_assignment , name= "students_assignment"),
    path('logout/',views.log_out,name='log_out'),
    path('reset_password/',
         auth_views.PasswordResetView.as_view(template_name="password_reset.html"),
         name="reset_password"),
    path('reset_password_sent/',
         auth_views.PasswordResetDoneView.as_view(template_name="password_reset_sent.html"),
         name="password_reset_done"),
    path('reset/<uidb64>/<token>/',
         auth_views.PasswordResetConfirmView.as_view(template_name="password_reset_form.html"),
         name="password_reset_confirm"),
    path('reset_password_complete/',
         auth_views.PasswordResetCompleteView.as_view(template_name="password_reset_done.html"),
         name="password_reset_complete"),
]
```

Admin Configuration

Configuration to make models accessible via the Django admin interface.

```
admin.site.register(student_profile)
admin.site.register(faculty_profile)
admin.site.register(Course)
admin.site.register(Assignment)
admin.site.register(Submission)
admin.site.register(Announcements)
admin.site.register(Material)
admin.site.register(feedback)
admin.site.register(query)
```

Course Management System (Project Level)

Settings

Configuration settings for the Django project, including database settings, middleware, and installed apps.

URLs

Project-level URL configuration to include app-level URLs.

Utils

`send_email_to_client`

Utility function for sending verification codes to users via email.

APIs and Endpoints

1. Error Page

- **Endpoint:** `/error`
- **Method:** GET
- **Request:** None
- **Response:** Displays the error page.

2. Home Page

- **Endpoint:** `/`
- **Method:** GET
- **Request:** None
- **Response:** Displays the home page.

3. Login - Student

- **Endpoint:** `/login/student`
- **Method:** POST
- **Request:** User login credentials.
- **Response:** Successful login or error message.

4. Login - Faculty

- **Endpoint:** `/login/faculty`
- **Method:** POST
- **Request:** Faculty login credentials.
- **Response:** Successful login or error message.

5. User Registration

- **Endpoint:** `/register`
- **Method:** POST
- **Request:** User registration details.
- **Response:** Successful registration or error message.

6. Add a New Course

- **Endpoint:** `/addcourse`
- **Method:** POST
- **Request:** Course details.
- **Response:** Successful course addition or error message.

7. Retrieve Enrolled Courses

- **Endpoint:** `/mycourse`
- **Method:** GET
- **Request:** None
- **Response:** Displays the courses enrolled by the user.

8. Search Enrolled Courses

- **Endpoint:** `/mycourse/search`
- **Method:** POST

- **Request:** Search parameters.
- **Response:** Displays search results for enrolled courses.

9. Retrieve Available Courses

- **Endpoint:** `/coursedashboard`
- **Method:** GET
- **Request:** None
- **Response:** Displays available courses.

10. Search Available Courses

- **Endpoint:** `/coursedashboard/search`
- **Method:** POST
- **Request:** Search parameters.
- **Response:** Displays search results for available courses.

11. Edit Course Details

- **Endpoint:** `/editcourse/{course_id}`
- **Method:** POST
- **Request:** Modified course details.
- **Response:** Successful course edit or error message.

12. Unenroll from a Course

- **Endpoint:** `/unenroll/{course_id}`
- **Method:** POST
- **Request:** None
- **Response:** Unenrolls the user from the specified course.

13. Delete a Course

- **Endpoint:** `/deletecourse/{course_id}`
- **Method:** POST
- **Request:** None
- **Response:** Deletes the specified course.

14. Retrieve Student List in a Course

- **Endpoint:** `/studentlist/{course_id}`
- **Method:** GET
- **Request:** None
- **Response:** Displays the list of students in the course.

15. Search Students in a Course

- **Endpoint:** `/studentlist/search/{course_id}`
- **Method:** POST
- **Request:** Search parameters.
- **Response:** Displays search results for students in the course.

16. View Profile of a Student in a Course

- **Endpoint:** `/viewprofile/{course_id}/{student_id}`
- **Method:** GET

- **Request:** None
- **Response:** Displays the profile of a specific student in the course.

17. Add a Course to a User

- **Endpoint:** `/addcoursetouser/{course_id}`
- **Method:** POST
- **Request:** None
- **Response:** Adds the specified course to the user.

18. Display Assignment Creation Page

- **Endpoint:** `/createassignment/{course_id}`
- **Method:** GET
- **Request:** None
- **Response:** Displays the page for creating a new assignment in the course.

19. Add a New Assignment to a Course

- **Endpoint:** `/addassignment/{course_id}`
- **Method:** POST
- **Request:** Assignment details.
- **Response:** Successful assignment addition or error message.

20. View Assignments for a Course

- **Endpoint:** `/viewassignments/{course_id}`
- **Method:** GET
- **Request:** None
- **Response:** Displays the assignments in the course.

21. Edit Assignment Details

- **Endpoint:** `/editassignment/{course_id}/{name}`
- **Method:** POST
- **Request:** Modified assignment details.
- **Response:** Successful assignment edit or error message.

22. Delete an Assignment

- **Endpoint:** `/deleteassignment/{course_id}/{name}`
- **Method:** POST
- **Request:** None
- **Response:** Deletes the specified assignment.

23. View Announcements for a Course

- **Endpoint:** `/announcements/{course_id}`
- **Method:** GET
- **Request:** None
- **Response:** Displays announcements for the course.

24. Add a New Announcement to a Course

- **Endpoint:** `/addannouncement/{course_id}`
- **Method:** POST

- **Request:** Announcement details.
- **Response:** Successful announcement addition or error message.

25. Delete an Announcement

- **Endpoint:** `/deleteannouncement/{course_id}/{ann_id}`
- **Method:** POST
- **Request:** None
- **Response:** Deletes the specified announcement.

26. View Materials for a Course

- **Endpoint:** `/materials/{course_id}`
- **Method:** GET
- **Request:** None
- **Response:** Displays materials for the course.

27. Add a New Material to a Course

- **Endpoint:** `/addmaterial/{course_id}`
- **Method:** POST
- **Request:** Material details.
- **Response:** Successful material addition or error message.

28. Add a Submission for an Assignment

- **Endpoint:** `/addsubmission/{course_id}/{name}`
- **Method:** POST
- **Request:** Submission details.
- **Response:** Successful submission addition or error message.

29. Edit a Submission for an Assignment

- **Endpoint:** `/editsubmission/{course_id}/{name}`
- **Method:** POST
- **Request:** Modified submission details.
- **Response:** Successful submission edit or error message.

30. View Submissions for an Assignment

- **Endpoint:** `/viewsubmission/{course_id}/{name}`
- **Method:** GET
- **Request:** None
- **Response:** Displays submissions for the assignment.

31. Grade a Submission

- **Endpoint:** `/gradesubmission/{course_id}/{name}/{sub_id}`
- **Method:** POST
- **Request:** Grading details.
- **Response:** Successful grading or error message.

32. View Feedback Responses for a Course

- **Endpoint:** `/fbresponse/{course_id}`
- **Method:** GET

- **Request:** None
- **Response:** Displays feedback responses for the course.

33. View Queries for a Course

- **Endpoint:** `/viewquery/{course_id}`
- **Method:** GET
- **Request:** None
- **Response:** Displays queries for the course.

34. Add a Query for a Course

- **Endpoint:** `/addquery/{course_id}`
- **Method:** POST
- **Request:** Query details.
- **Response:** Successful query addition or error message.

35. Reply to a Query

- **Endpoint:** `/replyquery/{course_id}/{qid}`
- **Method:** POST
- **Request:** Reply details.
- **Response:** Successful reply or error message.

36. Update User Profile

- **Endpoint:** `/updateprofile`
- **Method:** POST
- **Request:** Updated profile details.
- **Response:** Successful profile update or error message.

37. User Logout

- **Endpoint:** `/logout`
- **Method:** POST
- **Request:** None
- **Response:** Successful logout or error message.

Data Flow

User Registration and Login Flow:

- User registers using the `/register` endpoint, providing necessary details.
- Verification emails are sent using the `send_email_to_client` function from `utils.py`.
- After verification, users can log in using the `/login/student` or `/login/faculty` endpoints.
- Email verification is initiated through the `/otp_ver` endpoint.

Course Management Flow:

- Course-related actions are performed through `/coursedashboard` and `/coursedashboard/search`.
- Faculty can add, edit, and delete courses using the `/addcourse`, `/editcourse/{course_id}`, and `/deletecourse/{course_id}` endpoints.
- Students can enroll and unenroll from courses using the `/addcoursetouser/{course_id}` and `/unenroll/{course_id}` endpoints.

- Course information, including student lists, can be retrieved using the `/studentlist/{course_id}`, `/studentlist/search/{course_id}`, `/viewprofile/{course_id}/{student_id}`, and `/mycourse` endpoints.

Assignment Flow:

- Faculty can create, edit, and delete assignments using the `/createassignment/{course_id}`, `/addassignment/{course_id}`, `/editassignment/{course_id}/{name}`, and `/deleteassignment/{course_id}/{name}` endpoints.
- Students can view assignments for a course using the `/viewassignments/{course_id}` endpoint.
- Students can add, edit, and view submissions using the `/addsubmission/{course_id}/{name}`, `/editsubmission/{course_id}/{name}`, and `/viewsubmission/{course_id}/{name}` endpoints.
- Faculty can grade submissions using the `/gradesubmission/{course_id}/{name}/{sub_id}` endpoint.

Announcement Flow:

- Faculty can add and delete announcements for a course using the `/addannouncement/{course_id}` and `/deleteannouncement/{course_id}/{ann_id}` endpoints.
- Students can view announcements for a course using the `/announcements/{course_id}` endpoint.

Material Flow:

- Faculty can add and view materials for a course using the `/addmaterial/{course_id}` and `/materials/{course_id}` endpoints.

Query and Feedback Flow:

- Students can add, view, and reply to queries using the `/addquery/{course_id}`, `/viewquery/{course_id}`, and `/replyquery/{course_id}/{qid}` endpoints.
- Students can submit feedback using the `/fbresponse/{course_id}` endpoint.

User Profile Flow:

- Users can update their profiles using the `/updateprofile` endpoint.

Logout Flow:

- Users can log out using the `/logout` endpoint.

Backend Overview

The Course Management System's backend orchestrates a dynamic and comprehensive educational platform using Django and Python. The backend's core functionalities are encapsulated within the Academix_Portal app, structured around models such as student profiles, faculty profiles, courses, assignments, submissions, announcements, materials, feedback, and queries. User interactions are managed through dedicated views, handling registration, login, and email verification, while Course Management views govern course-related operations. The backend exposes APIs and endpoints for user authentication, course management, assignment and submission processes, and facilitates query handling and feedback submission. The data flow begins with incoming requests, navigating through views that interact with the underlying models and trigger necessary database transactions. Seamless communication between components ensures a smooth user experience, from course enrollment to submission grading. This streamlined data flow is integral to the system's functionality, creating a cohesive and efficient educational environment.