JMTAS: Journal of Multimedia Technology and Applied Software
Published: 30 April 2025
E-ISSN: 3047 - 4760

# Dual Mode Design On Backend Development Of Django-Based Kegiatan Belajar Mengajar (KBM) Application

[1st] Raden D Ahmad H*, [2nd] Davit Hermawan*, [3rd] Endra Abdul Hadi* , [4th] Nurdiansyah Permana*, [5th] Dani Rohpandi*
Teknologi Rekayasa Perangkat Lunak 1, Politeknik Mardira Indonesia 1, Teknologi Rekayasa Multimedia 2, Politeknik Mardira Indonesia 2, Teknologi Rekayasa Perangkat Lunak 3, Politeknik Mardira Indonesia 3, Teknologi Rekayasa Multimedia 4, Politeknik Mardira Indonesia 4, Teknik Informatika 5, STMIK Mardira Indonesia 5.
*email : radenspot@gmail.com 1*, davitkopites96@gmail.com 2*, abdulhadi.endra@gmail.com 3*, nurdiansyahpermana9@gmail.com 4*, danirtms@gmail.com. 5**

## ABSTRACT

**Educators often face significant administrative burdens from tasks like lesson planning and reporting, which impede efficiency and reduce direct student interaction time. This research aimed to design and develop a backend architecture for a learning activity management (KBM) application featuring a novel dual-mode system to serve both independent teachers and formal institutions. Employing an Agile methodology and the Django framework, the study developed a robust backend with a comprehensive database, models, views, and URL configurations. White-box testing was conducted to validate the internal logic. The findings confirm the successful implementation of a dual-mode architecture using Role-Based Access Control (RBAC), which effectively segregates data and permissions according to user roles. The developed prototype demonstrates a scalable and secure foundation for the application. The primary implication of this research is a validated architectural model for multi-tenancy educational software, offering a practical solution to streamline administrative workflows and enhance educator productivity. Future work should focus on frontend development and user acceptance testing.**
**Keywords: Backend Development; Django; Dual-Mode System; Educational Technology; Role-Based Access Control**

## ABSTRAK

**Pendidik seringkali menghadapi beban administratif yang cukup menyita waktu, seperti tugas-tugas perencanaan dan pelaporan, yang menghambat efisiensi dan bahkan mengurangi waktu interaksi dengan siswa. Penelitian ini bertujuan merancang dan mengembangkan arsitektur backend untuk aplikasi manajemen kegiatan pembelajaran (KBM) yang melalui sistem admin dual-mode untuk melayani kebutuhan guru yang mengajar secara independen dan guru yang tergabung dalam institusi formal. Melalui metodologi Agile, penelitian ini mengembangkan backend yang kuat dengan basis data komprehensif, model, tampilan, dan konfigurasi URL khas kerangka kerja Django. Pengujian white-box dilakukan untuk memvalidasi logika internal. Temuan ini mengkonfirmasi keberhasilan implementasi arsitektur dual-mode menggunakan Kontrol Akses Berbasis Peran (RBAC), yang secara efektif memisahkan data dan izin sesuai dengan peran pengguna. Prototipe yang ditampilkan menunjukkan fondasi yang dapat diperluas dan aman bagi pengembangan lanjut aplikasi. Implikasi utama dari penelitian ini adalah model arsitektur yang divalidasi untuk perangkat lunak pendidikan multi-tenan, menawarkan solusi praktis untuk merampingkan alur kerja administrasi dan meningkatkan produktivitas pendidik. Penelitian lanjut dapat memfokuskan diri pada pengembangan frontend dan pengujian penerimaan pengguna.**
**Kata kunci: Pengembangan Backend; Django; Sistem Dual-mode; Teknologi Pendidikan; Kontrol Akses Berbasis Peran**

## INTRODUCTION

Educators, encompassing both institutional teachers and independent tutors, frequently encounter significant challenges in time management and operational efficiency due to a substantial administrative workload. Among the most time-consuming responsibilities are the formulation of detailed lesson plans and the generation of student performance reports. This often manual and repetitive process not only diminishes the productive time available for direct student interaction but also underscores the necessity for a robust digital solution (Grissom & Loeb, 2011). The problem is further compounded by a bifurcated requirement for two distinct yet interconnected usage modes. On one hand (Mode 1), educators as end-users require a practical, flexible, and customizable tool to facilitate lesson planning in accordance with their individual teaching styles. On the other hand (Mode 2), educational institution administrators need a centralized system capable of monitoring, standardizing, and recapitulating data from all teaching staff to ensure quality control and generate comprehensive institutional reports.
The proposed solution is the development of a learning activity management application that offers scientific contributions from

two primary aspects: its system approach and its development technology. The first contribution lies in a dual-mode user approach specifically designed to accommodate two user personas within a single integrated platform: a personal mode for teachers/tutors and an administrative mode for institutions. This approach is a practical implementation of the Role-Based Access Control (RBAC) principle, which ensures that each user can only access features and data pertinent to their role, thereby enhancing both usability and data security. The implementation of RBAC is widely recognized as a standard for managing access rights in complex, multi-user systems (Sandhu et al., 1996). From a technological standpoint, the application's backend will be developed using the Django framework as its foundation. Django's advantages lie in its Object-Relational Mapping (ORM) architecture, robust built-in security features, and effective automation of the admin interface. The utilization of a high-level framework like Django has been shown to significantly enhance developer productivity and application reliability compared to development from scratch (Prechelt, 2000).The objective of this research is to design the backend architecture for a learning management application that accommodates dual-mode functionality for both individual and institutional use. Several prior studies highlight the relevance of this work. Research by Siregar, confirms the urgent need for applications to alleviate teachers' administrative loads, cautioning that their design must be carefully considered to avoid becoming a new burden (Sofyan Siregar et al., 2024). Concurrently, a study on RBAC by Putrawan, demonstrates that it is a proven and secure approach for educational applications with diverse user roles (Putrawan & Harahap, 2024). Furthermore, research by Sabita identifies the Django framework as a solid choice for building educational information systems, citing its tangible benefits in development speed, security, and scalability (Sabita et al., 2022).
Functionally, the scope of this research is strictly confined to backend development and does not encompass the features of a full-fledged Learning Management System (LMS), such as online examinations, student discussion forums, interactive video materials, financial management, or human resources information systems. The research is limited to the design of the database schema and the prototyping of Django-specific models, views, and URLs. Application testing will be conducted using the native Django Test Suite, which involves executing automated white-box tests to validate the functionality of the developed models and views.

## METHODOLOGY

This research is grounded in the Agile methodology, which is characterized by its emphasis on iterative development, flexibility, and close collaboration between developers and end-users. The principles of Agile align well with modern Model-View-Controller (MVC) frameworks such as Django, as this approach facilitates the rapid implementation of changes and immediate response to feedback (Ruby et al., 2013), enabling the delivery of functional features in short, incremental cycles (Hoda, 2019).The specific method employed is Rapid Application Development (RAD), which leverages component-based development and the utilization of automated tools.

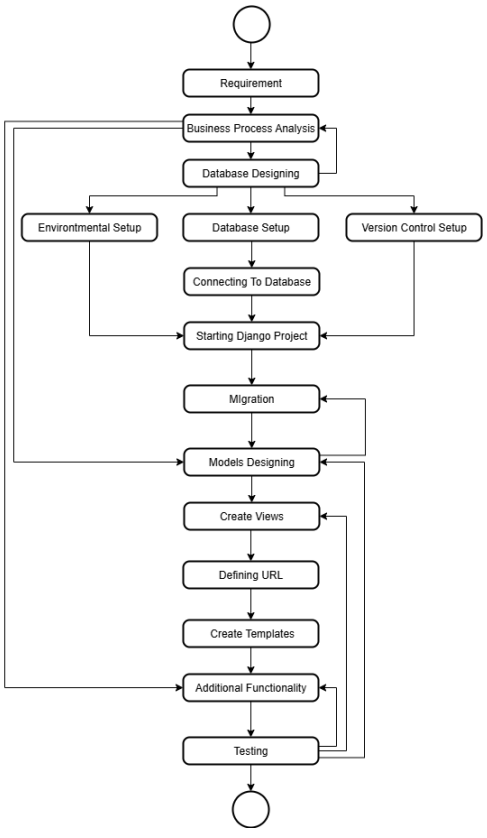


**Figure 1.**
Development Stages

Published: 30 April 2025
E-ISSN: 3047 - 4760

The choice of the Django framework is a direct implementation of the RAD methodology, as it provides a rich set of pre-built components, thereby obviating the need for developers to build foundational elements from scratch (Agrawal, 2024). Furthermore, features such as Django's auto-generated admin panel position it as a high-level platform conducive to the rapid prototyping of backend architecture and business logic (Murugesan et al., 2001).

Figure 1 illustrates the backend development stages of a Django application, from initial conceptualization to final testing. The process commences with "Requirement" identification , followed by "Business Process Analysis" to thoroughly understand the operational workflows. Subsequently, "Database Designing" is conducted to structure the application's database. Concurrently, three preparatory steps are undertaken: "Environmental Setup" to configure the development environment , "Database Setup" for specific database configurations , and "Version Control Setup" to establish a version management system.
Following these preparatory phases, the next steps involve "Connecting To Database" and "Starting Django Project". This is succeeded by "Migration" to apply model changes to the database. The process then moves to "Models Designing," focusing on the design of data models within Django. Subsequently, "Create Views" is performed to define application logic , "Defining URL" to map URLs to specific views , and "Create Templates" to develop the user interface. The development workflow also incorporates the addition of "Additional Functionality". It is important to note that the process allows for iterative refinement, with the possibility of returning to "Business Process Analysis" or "Models Designing" if modifications or enhancements are required. The final stage is "Testing," which ensures all functionalities operate correctly.

## RESULT AND DISCUSSION

**The core business process**, which illustrates the system's dual-mode architecture, is depicted in Figure 2. This architecture defines two primary user roles—Admin and Teacher—each with distinct sub-roles and corresponding privileges. The Admin role is bifurcated. A Super Admin possesses the highest system-wide access rights, with unrestricted control over the entire platform. In contrast, an Institutional Admin is responsible for one or more specific educational institutions. This role holds full authority to manage all master data and transactional records—such as academic years, classes, students, and schedules—exclusively within the context of their affiliated institution(s).

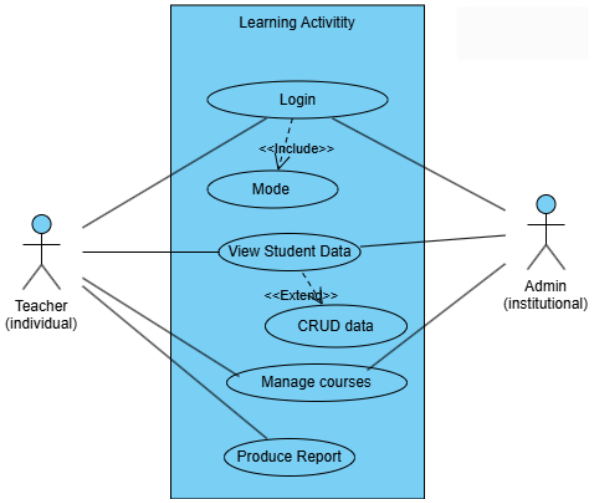


**Figure 2.**
Dual-Mode Process

Similarly, the Teacher role operates in two distinct capacities. An individual tutor functions as an independent entity, managing a self-contained set of data (e.g., personal academic terms, private classes, and individual students). Alternatively, an institutional teacher is formally affiliated with an institution within the system. In this capacity, they manage all data related to learning activities (known in Indonesian context as Kegiatan Belajar Mengajar or KBM) strictly within the scope and policies of the institution where they teach.

**The database**, part of ERD's shown in Figure 3, is designed on the MySQL platform to support a dual-mode learning activity management system, accommodating both the independent mode for individual educators and an integrated mode for educational institutions. The schema is structured to manage all facets of the teaching and learning process. A summary of the primary tables is provided below.

- *educational_institution*: Stores foundational data for educational institutions, including name, address, contact information, National School Principal Number (NPSN), head of institution, and logo.
- *app_users*: Manages application user credentials and roles. It includes username, password hash, email, role assignment (e.g., super admin, institutional admin, individual teacher, institutional teacher), and a foreign key linking to educational_institution for affiliated users.

- *teacher*: Contains specific profile data for teachers, such as full name, National Teacher ID (NIP/NUPTK), and employment status, linked via a one-to-one relationship with the app_users table.
- *student*: Stores profile information for students, including full name, student ID numbers (NIS/NISN), gender, date of birth, student type, status, and institutional affiliation.
- *academic_year*: Defines academic periods with start and end dates, active status, and an ownership link to either an institution or an individual teacher.
- *class*: Manages class data, including class name, assigned homeroom teacher (teacher_id), grade level, major/stream, the associated academic_year, and institutional affiliation.
- *subject*: Stores information about academic subjects, such as name, code, subject group, curriculum, and its affiliation with either an institution or an individual teacher.
- *lesson_schedule*: Details the recurring class schedules, specifying the assigned teacher, subject, class, academic year, day, time, room, and schedule status.
- *learning_activity_log*: Records daily teaching and learning activities. Key fields include date, time, instructional materials covered, assignments, notes, student attendance count, activity status, and foreign keys referencing the relevant teacher, class, subject, and schedule.
- *student_attendance*: A detailed log of student attendance for each learning_activity_log entry or other specified attendance events.
- *lesson_plan*: Contains the formal Lesson Plan, detailing learning objectives, materials, instructional activities, assessment methods, and reflections.learning_objective_element: Manages the specific learning objectives or achievement elements for each subject, grade level, and essential topic.lesson_plan_objectives: A junction table linking lesson_plan entries with their corresponding learning_objective_element records.
- *lesson_plan_resource*: Manages the resources and references used within a lesson_plan.
- *teacher_institution_affiliation*: A history log of a teacher's affiliations with various educational institutions over
- *time.teacher_subject_assignment*: A junction table that maps which subjects are taught by a specific teacher at a particular
- *institution.student_class_history*: Maintains a historical record of the classes a student has been enrolled in for each academic

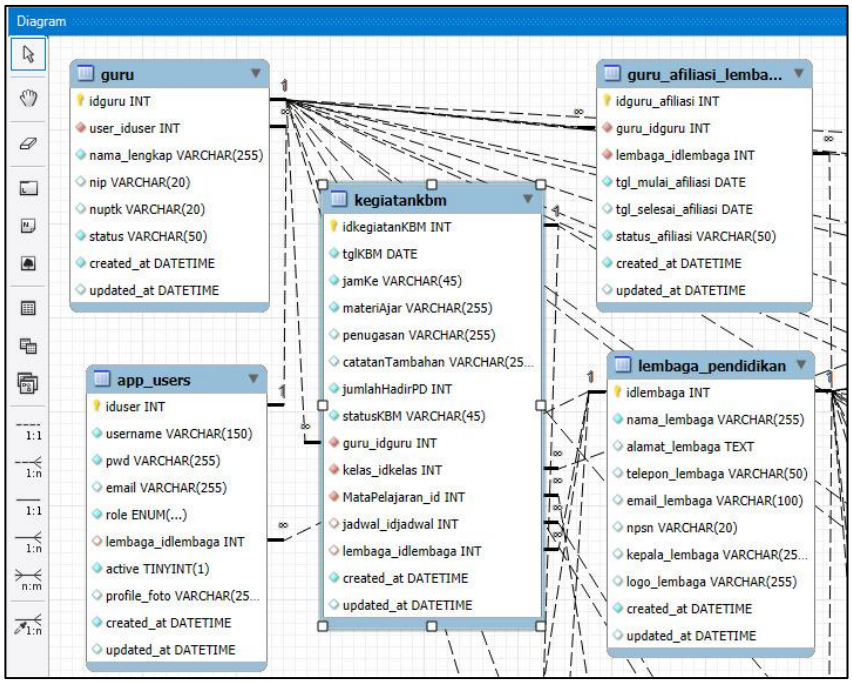


**Figure 3.**
Part of ERD's KBM Application

- *year.alumni_data*: Stores data for students after graduation, including contact information and post-graduation education or employment
- *history.teacher_activity_type*: Defines the types of non-teaching professional activities a teacher can perform, along with their standard time volume (e.g., in hours).
- *teacher_activity_log*: Records a teacher's non-teaching activities, with references to the activity type, date, description, and time volume.
- *additional_duty*: Defines types of supplementary roles or tasks a teacher can be assigned (e.g., Head of Library).
- *teacher_additional_duty_assignment*: A junction table assigning teachers to specific additional duties.

- *additional_duty_period*: Records the specific time frame during which a teacher holds an additional duty.In summary, this database schema is comprehensively designed to support robust learning activity management. It facilitates complete tracking of data—from teacher and student profiles to schedules, lesson plans, daily activities, and attendance—for both independent educators and those operating within an institutional framework.

**The initial Project Setup** followed the standard, prescribed workflow for scaffolding a Django application. The process commenced with the creation and activation of a dedicated virtual environment to ensure dependency isolation. Subsequently, the Django framework was installed within this isolated environment. The core project directory structure was then generated using the *django-admin startproject* command. Following the project's creation, a new application was initialized using *python manage.py startapp*. To integrate this new application, it was formally registered by adding its configuration class to the INSTALLED_APPS list within the project's *settings.py* file. Finally, the initial database schema was established by executing the migrate command, and the built-in development server was launched via *runserver* to verify the successful completion of the setup process (Django Software Foundation, 2025).

**The Directory Structure** shown in Figure 4 plays a critical role in the development, deployment, and maintenance of the application:

- *.gitignore* (File). This prevents the unnecessary inclusion of ephemeral or sensitive data, such as virtual environments, cache files, and user-uploaded media, thereby maintaining a clean and relevant commit history.
- *kbm_core*(Directory). Represents a core Django application within the project. It typically encapsulates the fundamental functionalities of the KBM system, housing essential components such as data models, view logic, URL routing configurations, and associated templates. Its modular nature promotes separation of concerns and enhances maintainability.
- *kbm_env* (Directory). Serves as the dedicated Python virtual environment for the project. This environment effectively prevents dependency conflicts that could arise when managing multiple Python projects on the same system.
- *kbm_project*` (Directory). Designated as the main Django project directory, this folder contains core project-level configurations and URL routing. Key files within this directory include: `*settings.py*`, the primary configuration file, which defines database connections, registers installed applications, specifies static file directories, and manages various other project-wide settings. *urls.py*, this file establishes the root URL patterns for the entire project, effectively routing incoming requests to the appropriate URLs defined within the registered Django applications. `*manage.py*`, a powerful script used for executing a wide array of administrative tasks within the project. Common operations performed via `manage.py` include, but are not limited to, starting the development server (`runserver`), performing database migrations (`makemigrations`, `migrate`), and initiating new Django applications (`startapp`).

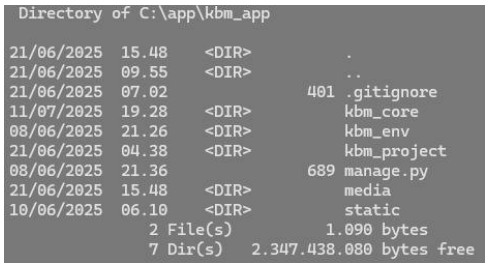


**Figure 4.**
The Directory Structure

- *media* (Directory). Specifically allocated for storing user-uploaded content. Examples include user-submitted images, documents, or other dynamic files generated through user interaction.
- *static* (Directory). Dedicated to housing static project files. Include client-side assets such as CSS, JavaScript files, images, and fonts, which are essential for the visual presentation and interactive functionality of the web application.

**Model Schema Definition (models.py)** commences with the importation of essential components (Figure 5), including models from the Django framework, the Image class from the Pillow library for image handling, *AbstractUser* for implementing a custom user model, and *timezone* for managing timestamps. To enforce data integrity and provide predefined options for model fields, several choice tuples are defined at the outset, such as TINGKAT_CHOICES for class levels and JURUSAN_CHOICES for academic majors. The initial section of the file lists models that are auto-generated by Django's native authentication and administration systems, including *AuthGroup, AuthPermission*, and *AuthUser*. A key characteristic of these models is the managed = False option within their inner Meta class. This directive instructs Django's migration framework to refrain from creating, modifying, or deleting the corresponding database tables, as they are presumed to exist and be managed by Django's core. Subsequently, the file defines the custom models that constitute the application's core architecture. This begins with *LembagaPendidikan*, which manages institutional details. The *AppUser* model extends Django's *AbstractUser* to incorporate custom roles (Super Admin, Lembaga Admin, Guru Individual, Guru Lembaga) and institutional affiliations. The Guru model stores teacher-specific information and establishes a link to the corresponding *AppUser*. Other foundational models include *Tahunajaran* for managing academic periods, *Matapelajaran* for subjects, and *Kelas* for class details, including the homeroom teacher, grade level, and major. The system also defines *Jeniskegiatanguru* and *Tugastambahan* to categorize non-teaching

activities and supplementary duties for teachers. The application manages student data through the *Pesertadidik* model, which contains details such as student ID numbers (NIS/NISN), gender, and status. The *AlumniData* model is used to track information for students post-graduation.

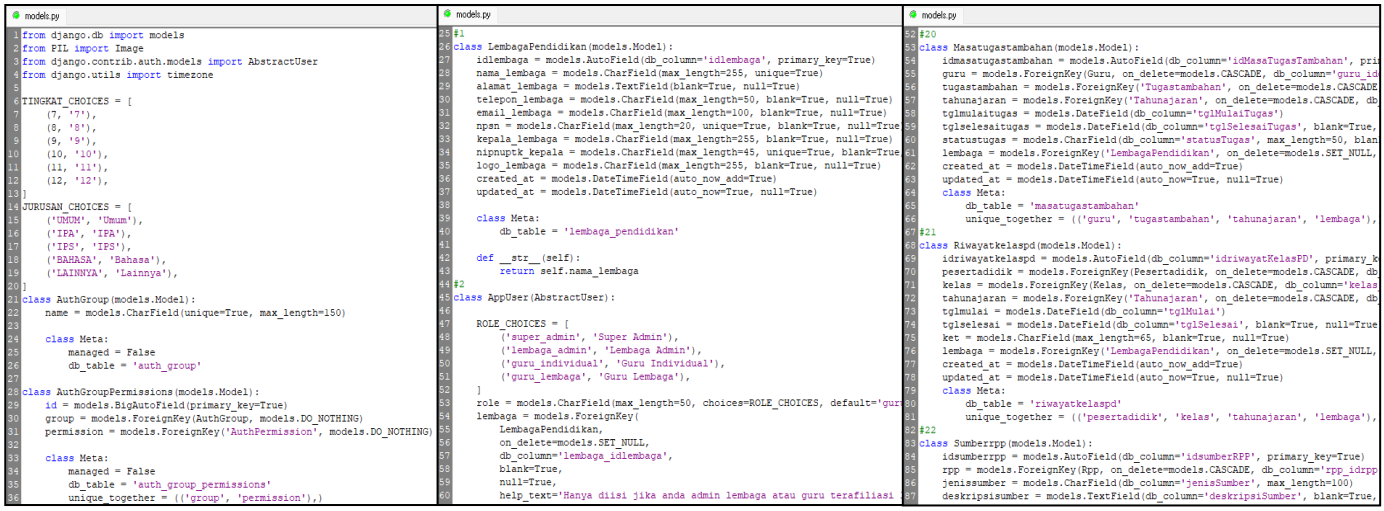

**Figure 5.**
Model Schema Definition (models.py)

The management of teaching and learning activities is orchestrated through a set of interconnected models. *JadwalPelajaran* organizes the schedules for teachers, subjects, and classes, while *Kegiatankbm* serves as a daily log, recording details such as instructional materials, assignments, and attendance counts. Student attendance is specifically recorded via the *Absensipd* model, and a teacher's non-instructional activities are logged in *Catatankegiatanguru*. For instructional planning, the *Elemencapaian* model defines learning objectives based on grade level and subject. The relationship between these objectives and a formal Lesson Plan (RPP) is managed by *the ElemencapaianHasRpp* junction table. The Rpp model itself is a comprehensive entity that stores the details of a lesson plan, including its title, objectives, materials, activities, assessments, and status. Finally, a series of models are defined to manage relational data and ensure data integrity. These include *GuruHasMatapelajaran*, *GuruHasTugastambahan*, *Masatugastambahan*, *Riwayatkelaspd*, and *Sumberrpp*, which respectively handle the many-to-many relationships between teachers and their assigned subjects or duties, maintain a history of student class placements, and manage resources associated with lesson plans. A recurring design pattern across nearly all custom models is the inclusion of *created_at* and *updated_at* timestamp fields for auditing purposes, as well as a foreign key to the *LembagaPendidikan* model to support multi-institutional data segregation.

**View Logic and Implementation (views.py)** encapsulates the application's business logic, handling the interactions between HTTP requests and their corresponding web responses (Figure 6). Fundamentally, each function within this file constitutes a "view" that accepts a user request, processes data—often by interacting with the database models—and subsequently returns a web response, typically by rendering an HTML template. Authentication and Session Management The initial section of the file is dedicated to authentication functionality. The *login_view* function manages the user authentication process, redirecting successfully authenticated users to the appropriate dashboard based on their assigned role (e.g., Super Admin, Lembaga Admin). It also provides user feedback through success or error messages. Conversely, the *logout_view* handles user session termination and redirects the user back to the login page with an informational message. Super Admin Exclusive Functionality A comprehensive set of views for "Manajemen Lembaga Pendidikan" (Institutional Management) is implemented, accessible exclusively by the Super Admin. This includes views for listing, creating, editing, and deleting *LembagaPendidikan* entities. A similar suite of views exists for "Manajemen Pengguna (Super Admin)" (User Management), empowering the Super Admin to manage all *AppUser* accounts across the entire system. Core Entity Management and Access Control The file implements a robust set of views for managing core entities, namely "Manajemen Kelas" (Class Management), "Manajemen Guru" (Teacher Management), and "Manajemen Siswa" (Student Management). For each of these entities (*Kelas*, *Guru*, *Pesertadidik*), a standard suite of views for listing, viewing details, adding, editing, and deleting data is provided. Access control for these views is role-dependent. For instance, the creation and modification of classes, teachers, and students are restricted to users with Super Admin, Lembaga Admin, and occasionally Guru Lembaga roles. A critical feature of these views is the intelligent data filtering mechanism. For any user who is not a Super Admin, all data query sets are automatically filtered based on the logged-in user's institutional affiliation. This ensures that a Lembaga Admin or Guru can only view and manage data pertinent to their own institution, thereby enforcing data segregation and security. These functions also handle form validation, display success or error messages via Django's messaging framework, and utilize a generic template for deletion confirmations. Furthermore, data sorting functionality is implemented for the teacher and student list views, controlled via URL query parameters. General-Access Views Common views, such as home and about, are accessible to all authenticated users. These pages display basic information, such as the current server time and details about the user's role and institutional affiliation.

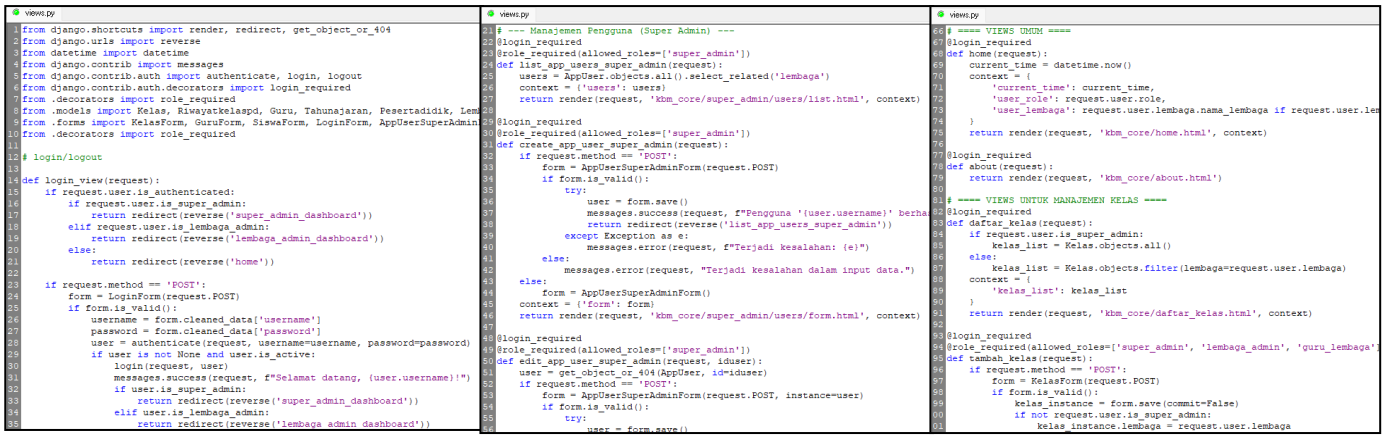


**Figure 6.**
View Logic and Implementation (views.py)

**URL Configuration (urls.py)** (Figure 7) The urls.py file defines the URL routing patterns for the application, mapping various URL endpoints to their corresponding views in views.py. Each path entry within the *urlpatterns* list specifies a URL to be captured, the view function designated to handle it, and a unique name. This naming convention is crucial as it facilitates dynamic and reverse URL resolution within the application's templates and code. General and Core Entity Routes Initially, the script defines base URLs for general-access functionalities, such as the home page (/) and the about page (/about/). Authentication endpoints are also established for login (/login/) and logout (/logout/). Following this, a consistent and RESTful-style URL structure is implemented for the core entity management of Kelas (Classes), Guru (Teachers), and Siswa (Students). For each entity, a complete set of URL patterns is provided to support standard CRUD (Create, Read, Update, Delete) operations. This includes endpoints for listing all items (e.g., /kelas/), adding a new item (/kelas/tambah/), and viewing (/kelas/<int:id_kelas>/), editing (/kelas/<int:id_kelas>/edit/), and deleting (/kelas/<int:id_kelas>/hapus/) a specific item.

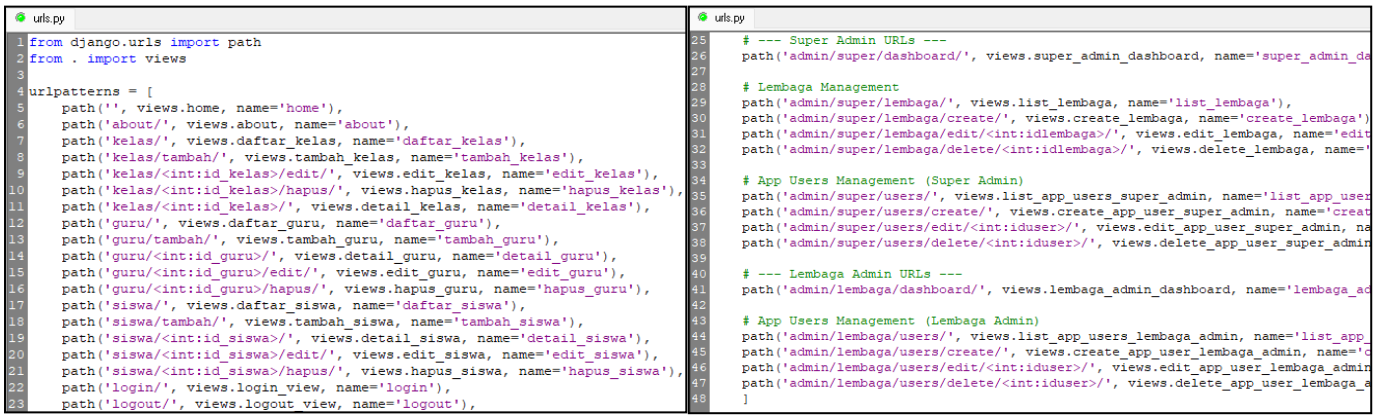


**Figure 7.**
URL Configuration (urls.py)

Role-Based Access Routes A significant portion of the file is dedicated to URL patterns that are segregated by user roles, particularly for the Super Admin and Lembaga Admin.

- Super Admin Routes: A dedicated namespace is provided for the Super Admin. This includes a path to their primary dashboard (/admin/super/dashboard/). Furthermore, a full suite of URLs for the management of educational institutions is defined under /admin/super/lembaga/, encompassing list, create, edit, and delete functionalities. The Super Admin also possesses a distinct set of URLs under /admin/super/users/ for managing all application user accounts system-wide.
- Lembaga Admin Routes: Similarly, the Lembaga Admin is provided with a specific dashboard URL (/admin/lembaga/dashboard/). Their user management capabilities are scoped to their own institution, with corresponding URLs for listing, creating, editing, and deleting users defined under the /admin/lembaga/users/ path.

This hierarchical and role-based URL structure explicitly segregates access paths, directly mirroring and enforcing the access control logic defined within the associated views.

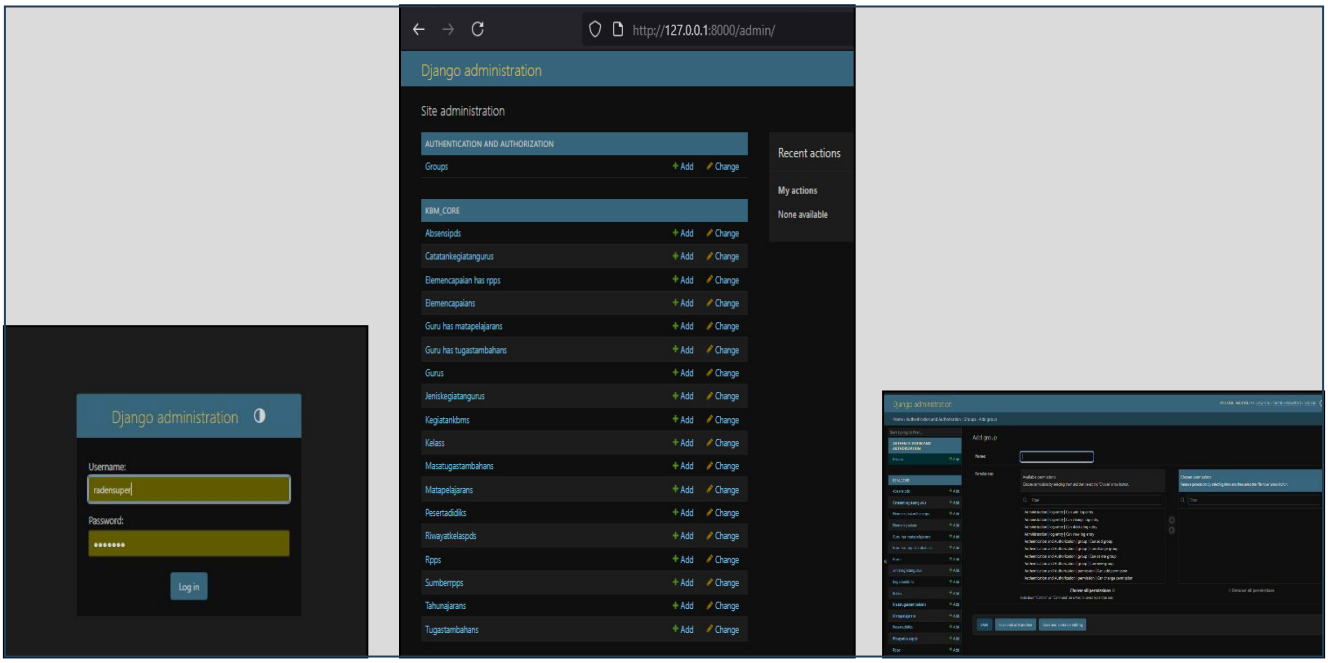


**Figure 8.**
Django Administrative Page

**Template Directory Structure** holds a crucial role as the repository for all HTML template files used to render the client-side user interface (UI). Django have own minimal UI forms to manage administrative stage (Figure 8) While the directory structure is flexible, this project adheres to standard Django conventions to enforce a clear separation between HTML markup and the Python-based logic contained within views. Implementation and Namespacing Convention In this project, the templates directory is located within the application's own directory structure, specifically at kbm_core/templates/kbm_core/. Consequently, individual HTML files are placed within this nested directory (e.g., kbm_core/templates/kbm_core/login.html). This namespacing convention is of paramount importance for preventing template name collisions between different applications within the same project. For instance, if two separate applications were to both contain a template named login.html, Django's template loader can disambiguate them by their unique path (e.g., app_name/login.html). Django automatically discovers templates in this location provided that the application is registered in the INSTALLED_APPS list and the APP_DIRS setting is set to True within the TEMPLATES configuration in settings.py. This separation of concerns inherently promotes modularity and reusability. Templates specific to a particular feature are co-located with the relevant application's code, while globally shared templates can be stored in a centralized, project-level directory. When rendering a view, Django's template loader follows a prescribed search order defined in the TEMPLATES setting. Typically, with APP_DIRS enabled, it will first search within the app-level directories before proceeding to the project-level locations specified in the DIRS list. This well-defined structure effectively facilitates team-based development, code maintainability, and the overall scalability of a Django project by cleanly separating UI concerns. While the creation and styling of templates are generally considered a domain of frontend development, this project includes a prototype *login.html* page. Its inclusion serves as a practical implementation of a template whose directory structure and initial setup fall within the scope of the backend development process (Figure 9).

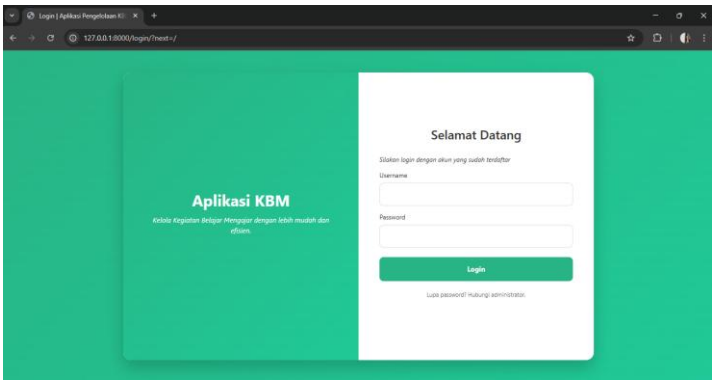


**Figure 9.**
A prototype login.html page

Other pages in Figure 10 show the student HTML page and the teacher HTML page, proving how rapid they are to build the frontend based on the Django framework.

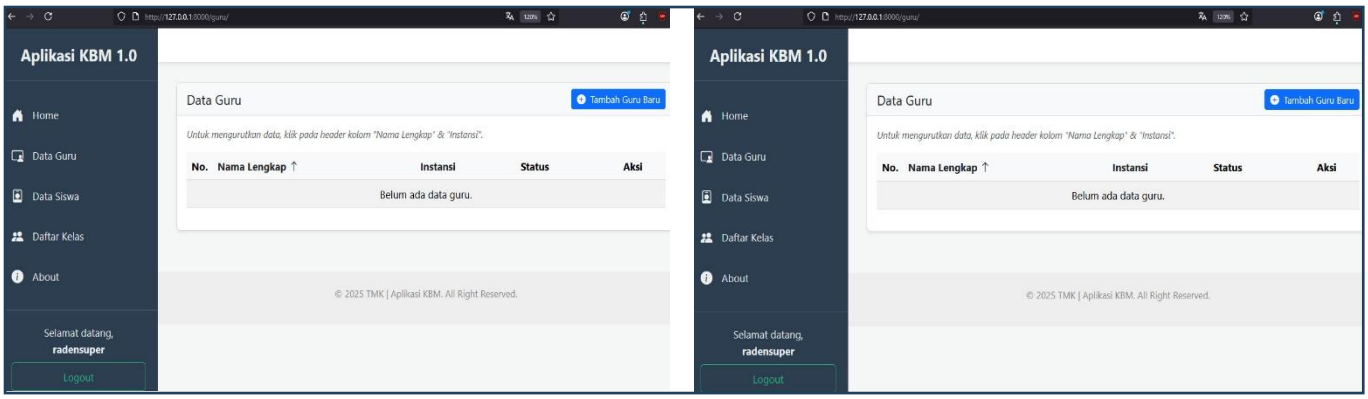


**Figure 10.**
Pages of siswa.html and guru.html

**White-Box Testing Methodology** focuses on scrutinizing the internal structure, design, and implementation of the system to ensure that the code functions as expected. To facilitate this, a dedicated *tests.py* file was created and populated with a comprehensive suite of test scripts (Figure 11). Within this file, the TestCase class, provided by Django's testing framework (which extends Python's standard unittest module), was utilized to structure the tests. All tests were executed from the project's root directory by running the command *python manage.py test*. (Figure 12)

The following sections detail the results of the testing performed within the scope of backend development.

- Model Testing. The objective of model testing was to verify the integrity and correctness of the data layer. The tests confirmed that: Core models such as LembagaPendidikan, AppUser, and Guru were defined with the correct field types and attributes. Data validation rules (e.g., max_length, null, blank) functioned correctly, preventing the entry of invalid data. Inter-model relationships, including ForeignKey and OneToOneField, were established appropriately and enforced relational integrity. Any custom methods defined on the models operated as intended.
- View Testing. View testing involved assessing the application's business logic and its handling of HTTP requests and responses. The tests were designed to: Verify that views return the correct HTTP status codes under various conditions, such as 200 OK for successful requests, 404 Not Found for non-existent resources, and 302 Found for redirects (e.g., after a successful form submission or for unauthenticated access). Confirm that views correctly interact with the database models, forms, and Django's authentication system. Simulate various user scenarios by programmatically logging in as a Super Admin, Lembaga Admin, or Guru. These tests verified that the data retrieved, displayed, or saved was accurate and appropriately scoped according to the user's role and permissions.

tests.py

```
from django.test import TestCase, Client
from django.urls import reverse
from .models import LembagaPendidikan, AppUser, Guru, Kelas, Tahunajaran

class KBMAppTestCase(TestCase):

    def setUp(self):

        self.lembaga_a = LembagaPendidikan.objects.create(
            nama_lembaga="SMA ABC",
            npsn="12345678"
        )
        self.lembaga_b = LembagaPendidikan.objects.create(
            nama_lembaga="SMP XYZ",
            npsn="87654321"
        )
```

**Figure 11.**
Part code of file test.py

- URL Routing Testing. This phase of testing focused on ensuring the integrity of the URL configuration. The tests validated that: The urls.py configuration correctly maps incoming URL requests to the intended view functions.URL arguments, such

as primary keys (e.g., <int:id_kelas>), were successfully captured from the URL path and passed as arguments to the corresponding view.

```
(kbm_env) C:\app\kbm_app>python manage.py test
C:\app\kbm_app\kbm_env\Lib\site-packages\django\db\models\base.py:368: RuntimeWarning: Model
'kbm_core.elemencapaianhasrpp' was already registered. Reloading models is not advised as it
can lead to inconsistencies, most notably with related models.
  new_class._meta.apps.register_model(new_class._meta.app_label, new_class)
Found 5 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
```

**Figure 12.**
Running test command

## CONCLUSIONS AND SUGGESTIONS

This research has successfully demonstrated the design and implementation of a robust and scalable backend architecture for a learning activity management (KBM) application using the Django framework. The principal contribution of this study is the validation of a dual-mode system architecture, which serves as a novel, practical application of Role-Based Access Control (RBAC) theory to effectively accommodate the disparate operational needs of both independent educators and formal educational institutions within a single, cohesive platform. The theoretical implication of this finding is the emergence of a validated model for multi-tenancy educational software that maintains strict data segregation and role-based permissions. The primary practical implication is the potential to significantly streamline administrative workflows for a wide spectrum of educators, thereby increasing operational efficiency and allowing more time to be dedicated to direct instruction. A key limitation of this research is its exclusive focus on backend development; the user interface and frontend logic were not developed beyond a foundational prototype. Therefore, it is strongly recommended that future research prioritize the development of a comprehensive, responsive, and intuitive user interface (UI/UX) to fully realize the system's potential. Further development could also involve expanding the application's scope by integrating advanced Learning Management System (LMS) functionalities, such as automated assessment tools, student progress analytics dashboards for administrators, and parent communication portals.

Finally, to ensure the application's practical usability and effectiveness, conducting extensive User Acceptance Testing (UAT) with the target user personas—teachers and administrators from various institutional contexts—is highly suggested to gather critical feedback for future iterative improvements.

## THANKS TO

The author wishes to extend sincere gratitude to several individuals whose support was invaluable to the completion of this work. Special thanks are conveyed to esteemed faculty colleagues—Mr. Davit Hermawan, Mr. Endra Abdul Hadi, Mr. Nurdiansyah Permana, and Mr. Yuda Purnama Putra—for their insightful discussions and encouragement. Deep appreciation is also directed to Mr. Enang Rusnandi, M.Kom., the Director of Politeknik Mardira Indonesia (POLTEKMI) Majalengka, for his institutional support and leadership throughout this research.

## BIBLIOGRAPHY

### Journal Article

Grissom, J. A., & Loeb, S. (2011). Triangulating Principal Effectiveness: How Perspectives of Parents, Teachers, and Assistant Principals Identify the Central Importance of Managerial Skills. *American Educational Research Journal*, *48*(5), 1091–1123. https://doi.org/10.3102/0002831211402663

Hoda, R. (Ed.). (2019). *Agile Processes in Software Engineering and Extreme Programming – Workshops: XP 2019 Workshops, Montréal, QC, Canada, May 21–25, 2019, Proceedings* (Vol. 364). Springer International Publishing. https://doi.org/10.1007/978-3-030-30126-2

Murugesan, S., Deshpande, Y., Hansen, S., & Ginige, A. (2001). Web Engineering: A New Discipline for Development of Web-Based Systems. In S. Murugesan & Y. Deshpande (Eds.), *Web Engineering* (Vol. 2016, pp. 3–13). Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-45144-7_2

Prechelt, L. (2000). An empirical comparison of seven programming languages. *Computer*, *33*(10), 23–29. https://doi.org/10.1109/2.876288

Putrawan, & Harahap, A. M. (2024). Implementasi Metode Role-Based Access Control Pada Aplikasi E-Raport di MIN 15 Langkat Berbasis Android. *Jurnal Teknik Informatika Unika ST. Thomas (JTIUST)*, *09.*

Sabita, H., Arkhiansyah, Y., Rahardi, A., & Trisnawati, S. (2022). Implementasi Base View Framework Django pada Pengembangan Sistem Informasi Akreditasi Prodi. *Jurnal SIMADA (Sistem Informasi Dan Manajemen Basis Data)*, *5*(2), 33–37. https://doi.org/10.30873/simada.v5i2.3418

Sandhu, R. S., Coyne, E. J., Feinstein, H. L., & Youman, C. E. (1996). Role-based access control models. *Computer*, *29*(2), 38–47. https://doi.org/10.1109/2.485845

Sofyan Siregar, A., Zulhimma, Z., Oslerking, B., Rambe, A., & Harahap, A. (2024). OPTIMALISASI MANAJEMEN ADMINISTRASI GURU MELALUI APLIKASI SIAGUD DI MAN 2 PADANGSIDIMPUAN. *EDUTECH*, *23*(2), 213–229. https://doi.org/10.17509/e.v23i2.69946

**Book**

Ruby, S., Thomas, D., & Heinemeier Hansson, D. (2013). *Agile web development with Rails 4*. The Pragmatic Bookshelf.

**Website**

Agrawal, A. (2024, January 3). Mastering Rapid Application Development with Django. *Livepositively.Com*. https://cisin.livepositively.com/mastering-rapid-application-development-with-django/

Django Software Foundation. (2025). *Django documentation*. Django Software Foundation. https://docs.djangoproject.com/en/5.2/intro/