Tunis Business School
University of Tunis

Web Service Project Report

Development of a RESTful API for University Residencies in Tunisia

---

Realised by: Safa Zaghdoudi

Academic Advisor : Mr . Montassar Ben Messaoud

Academic Year: 2024-2025

Abstract

This report presents the development of a RESTful API for managing university residencies in Tunisia. The API aims to streamline the process of applying for and managing student housing by providing a centralized platform for both students and administrators.
The backend is implemented using **Python** with **Flask**, interacting with a **MongoDB Atlas Cloud database**.
The frontend is built using **Next.js** with **Tailwind CSS**, offering a user-friendly interface for students to explore available residencies, submit applications, and track their status.
This project addresses the growing demand for efficient and accessible student housing solutions in Tunisian universities by leveraging modern technologies and a robust API architecture.

Contents

List of Figures

List of Tables

General Introduction

In the increasingly interconnected world of today, efficient and accessible access to resources is crucial. This is particularly true for students who often rely on public university-provided housing to support their academic pursuits.
In Tunisia, as the sector of higher education continues to grow, current systems are encountering many problems such as security, crushing, and long processing times. That is why the need for a robust and user-friendly system for managing student residencies becomes increasingly important.
Traditional methods of managing student housing in Tunisia are currently semi automated, students apply for residency online based on their university orientation , get assigned to a residency where they have to go personally bring all their justifications and fill paper-based application forms. Then they their rooms assigned based on the administrator's personal judgment which can sometimes be unfair. This process can be cumbersome and inefficient with limited communication channels. These inefficiencies can lead to delays in processing applications, difficulties and favoring in managing room assignments, and a less than optimal experience for both students and administrators.
To address these challenges, this project aims to develop a modern and efficient solution: a **RESTful API** for managing university residencies in Tunisia. This API will provide a centralized platform for students to apply for residency online, fill out the form with all necessary information such as preferred roommate or disease status, track their application status, and communicate with administrators. For administrators, the API will streamline the process of managing residency information, automatically assigning rooms, and communicating with students.
Using the power of modern technologies, such as **cloud computing** and **web APIs**, this project seeks to enhance the student experience, improve administrative efficiency and contribute to the overall success of Tunisian universities.

Chapter 1

Context and Taxonomy

1.1 Context

The Tunisian higher education system is experiencing significant growth, leading to an increased demand for student housing. However, traditional methods of managing student residences often involve manual processes, paper-based applications, and limited communication channels, leading to inefficiencies and delays. These inefficiencies can include:

- Long processing times: Manual application processing can be time-consuming, leading to delays in assigning students to residencies.

- Poor communication: Limited communication channels between students and administrators throughout the application process can create confusion and frustration.

- Difficulty in managing data: Manual data entry and record-keeping can lead to errors and inconsistencies in student information.

- Limited access to information: Students may have limited access to information about available residencies, application deadlines, services provided by residencies and other relevant information.

-Favoritism in room Allocation: administrators often Favor certain applicants based on subjective criteria, such as personal relationships or biases, potentially disadvantaging other deserving students. To ensure equity, it is essential to adopt an objective and transparent system for residency assignments, prioritizing merit-based factors and equal opportunity for all applicants.

To address these challenges and improve the overall student experience, there is a need for a more efficient and streamlined automated system for managing student residencies in Tunisian universities.

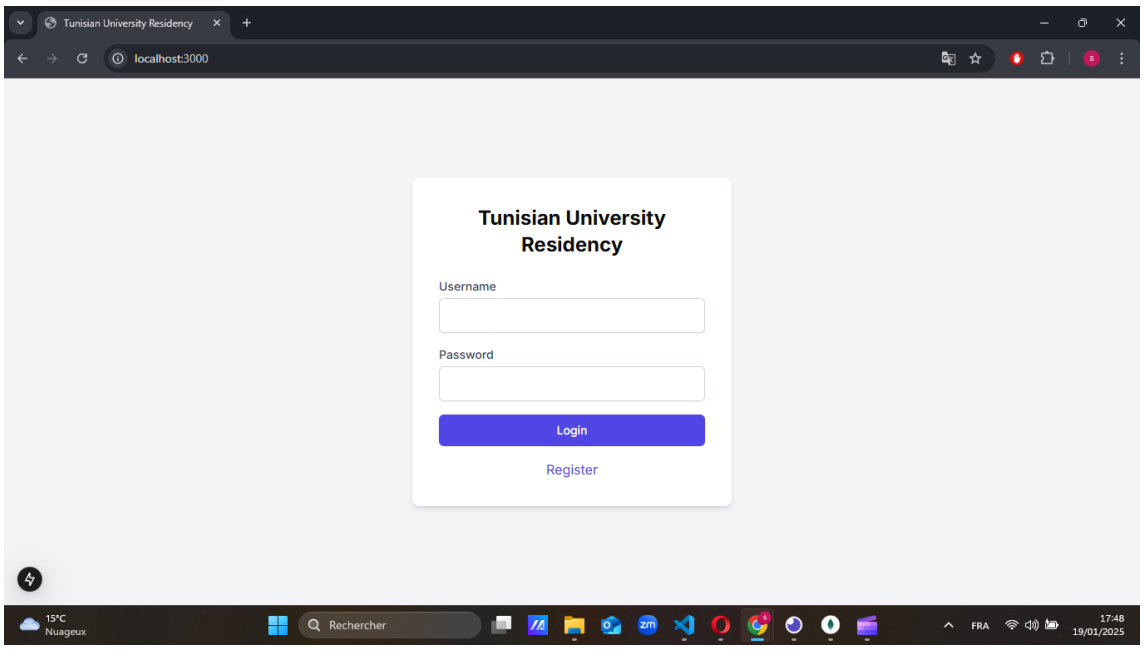

Figure 1.1: How the website works

1.2 Taxonomy

1.2.1 Restful APIs

A REST API, also known as a RESTful API, is a simple, uniform interface that is used to make data, content, algorithms, media, and other digital resources available through web URLs. REST APIs are the most common APIs used across the web today. REST API standards are [4] :

- **Use of a uniform interface (UI)**: To have a uniform interface, multiple architectural constraints are required to guide the behavior of components. Additionally, resources should be unique so they are identifiable through a single URL.
- **Client-Server Architecture**: The uniform interface separates user concerns from data storage concerns. The client's domain concerns UI and request-gathering, while the server's domain concerns focus on data access, workload management, and security.
- **Statelessness**:Request from client to server must contain all of the information necessary so that the server can understand and process it accordingly. The server can't hold any information about the client state.
RESTful resource caching: Data within a response to a request must be labeled as cacheable or non-cacheable.
- **Layered system**: REST allows for an architecture composed of hierarchical layers. In doing so, each component cannot see beyond the immediate layer with which they are interacting.
- **Code on demand**: Because REST APIs download and execute code in the form of applets or scripts, there's more client functionality. Oftentimes, a server will send back a static representation of resources in the form of XML or JSON.

1.2.2 Importance of CRUD Operations in Data Management Systems

CRUD operations provide a structured and consistent way to interact with data, ensuring data integrity and facilitating efficient data management. CRUD operations are fundamental to data management systems. They refer to the following functions[5]:
- Create: Adding new data to the system (e.g., creating a new residency).
- Read: Retrieving existing data from the system (e.g., fetching a list of available residencies).
- Update: Modifying existing data (e.g., updating residency information).
- Delete: Removing data from the system (e.g., deleting a residency).

In the context of our API, CRUD operations are implemented for both residencies and student applications.

1.3 Similar APIs Research

While several online platforms and systems exist for managing student housing, many are either proprietary or lack the flexibility and customization options offered by a dedicated API. Some examples include:

- Office Of University Services for the North Region (OOUN)[2]:The Ministry has its own internal systems for managing student housing in the North, which is often accessible to external developers.
- Office of University Services for the Central Region (OOUC)[1] : also managed by the Ministry.
- Office of University Services for the South Region (OOUS)[3]: also managed by the Ministry.
- Commercial Housing Platforms: Platforms like Airbnb and Zillow focus on residential rentals, not specifically on student housing needs.

This project aims to develop a more open and adaptable solution specifically tailored to the needs of Tunisian universities. By implementing a RESTful API, we can enable greater integration with other university systems and facilitate the development of custom applications for students and administrators.

Chapter 2

Design and Methodology

2.1 Design

In the design phase of the project, I focused on developing a RESTful API tailored for Tunisian University Residency Management. The API serves as not only a managing Residencies, Student applications, and review but also registration and login information. The API's endpoints are designed to allow administrators to easily retrieve, update, and manage University Residencies information and students to retrieve residencies' data and manage their application information . This design aims to facilitate the process, ensuring that they can access accurate, up-to-date information faster, allowing for better and effective management. By integrating this system with future tools, such as tracking and mapping technologies, it enhances the overall management process, reducing delays, and improving operational efficiency.

2.1.1 Class Diagram

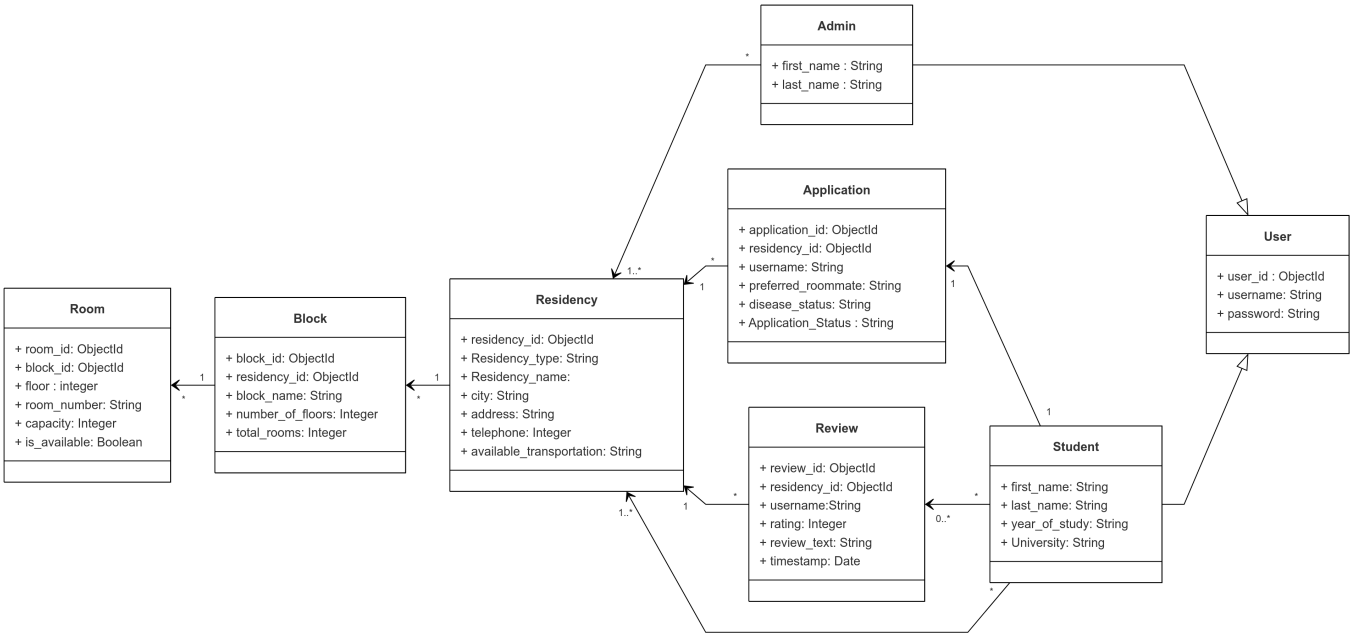

Figure 2.1: Class Diagram

2.1.2 Database Design

Like shown in figure 2.1 the database design is the following :

Users Model : This model is composed of :

- **user_id**: The primary key of the users table indicating the id of the user.
- **username** : A string indicating the name of the user and it is the foreign key linking to the student_data table.
- **password** : A string of the hashed password of the user.

Admin Model : This model inherits its attributes from the users table and adds the following ones:

- **first_name**: A string indicating the admin's first name.
- **last_name** : A string indicating the admin's family name.

Student Model : This model inherits its attributes from the users table and adds the following ones:

- **first_name**: A string indicating the student's first name.
- **last_name** : A string indicating the student's family name.
- **year_of_study** : A string indicating the current academic year.
- **University** : A string indicating the student's university.

Application Model : This model is composed of :

- **application_id**: The primary key of the Application table.
- **residency_id** : the foreign key referencing the primary key of the residency model and indicates the residency the student is applying for.
- **username** : A string indicating the n me of the student who made the application.
- **preferred_roommate** : A string indicating the name of the roommate the student wants to be partnered with.
- **disease_status** : A string indicating whether the student has a disease that allows them to get an individual room.
- **Application_status** : A string indicating the status of the application.

Review Model : This model is composed of :

- **review_id**: The primary key of the Review table.
- **residency_id** : the foreign key from the residencies table and indicated the residency the student is reviewing.
- **username** : A string indicating the n me of the student who made the review.
- **rating** : An integer on a scale of 5 that the student is giving to that residency and its services.
- **review_text** : A string where the student explains the rating it gave and the details .
- **timestamp** : the date the review was submitted.

Residency Model : This model is composed of :

- **residency_id**: The primary key of the Residencies table.
- **Residency_type** : A string indicating the type of the Residency (Public University Residency).
- **Residency_name** : A string indicating the name of the Residency.
- **city** : A string indicating the location of the residency.
- **Address** : A string indicating the exact address of the Residency.
- **Telephone** : An integer of the administration telephone number.
- **Available_transportation** : A string of the available transportations.

Block Model : This model is composed of :

- **block_id**: The primary key of the Block table.
- **residency_id** : The foreign key from the residency table.
- **block_name**: A string indicating the name of the block.
- **number_of_floors** : An Integer indicating the number of floors.

- **total_rooms** : An integer indicating the total number of rooms.

Room Model : This model is composed of :

- **room_id** : The primary key from the room table indicating the room's id.
- **block_id**: The foreign key from the Block table.
- **floor** : An Integer indicating the floor where the room is located.
- **room_number** : An Integer indicating the room's number.
- **capacity** : An integer indicating the total number of student the room can host.
- **is_available** : A Boolean indicating whether the room is empty or not.

2.1.3 Relationship between tables

- Admin-User: the Admin inherits attributes from the User table.

- Student-User: the Student inherits attributes from the User table.

- Student-Application: One-to-one (a student can submit one application).

- Residency-Application: One-to-Many (a residency can have multiple applications).

- Student-Review: zero-to-Many (a student can submit multiple reviews).

- Residency-Review: One-to-Many (a residency can have multiple reviews).

- Student-Residency: Many-to-Many (All students can view all the residencies data).

- Admin-Residency: Many-to-Many (All admins can manage all the residencies data).

- Residency-Block: One-to-Many (a residency can have multiple blocks).

- Block-Room: One-to-Many (a block can have multiple rooms).

2.2 Description of Endpoints

2.2.1 What is an endpoint

API endpoints serve as the specific digital locations where client requests for information are sent by one program to retrieve the digital resource that exists there. They're the points at which the client and the server communicate, enabling two applications to share resources.

2.2.2 Authentication endpoints

Table 2.1 below represents the Authentication process endpoints:

HTTP method	Endpoint Path	Description
POST	/auth/register	Register a new user (admin or student).
POST	/auth/login	Obtain the JWT token to log in the user.
POST	/auth/logout	Logout a user by invalidating the JWT token.

Table 2.1: Authentication Endpoints

2.2.3 Administrator endpoints

Table 2.2 below represents the Administrator endpoints:

HTTP method	Endpoint Path	Description
GET	/residencies	Fetch all residencies.
GET	/residency/<residency_id>	Get residency details.
POST	/residencies	Create a new residency.
PUT	/residency/<residency_id>	Update a residency.
DELETE	/residency/<residency_id>	Delete a residency.
GET	/<residency_id>/blocks	Fetch all blocks by residency.
GET	/blocks/<block_id>	Fetch a block by id.
POST	<residency_id>/blocks	Create a new block.
PUT	/blocks/<block_id>	Update a block.
DELETE	/blocks/<block_id>	Delete a block.
GET	/<block_id>/rooms	Fetch all rooms by block.
GET	/rooms/<room_id>	Fetch a room by id.
POST	<block_id>/rooms	Create a new room.
PUT	/rooms/<room_id>	Update a room.
DELETE	/rooms/<room_id>	Delete a room.
GET	/applications	Fetch all student applications.
GET	/applications/<application_id>	Fetch an application by id.
GET	/reviews	Fetch all student reviews.
GET	/reviews/<review_id>	Fetch a review by id.

Table 2.2: Administrator Endpoints

2.2.4 Student endpoints

Table 2.3 below represents the student endpoints:

HTTP method	Endpoint Path	Description
GET	/residencies	Fetch all residencies and their information.
GET	/residencies/<residency_id>	Return the information of the residency by its id.
POST	/applications	Add an application to the database.
DELETE	/applications/<application_id>	Delete the application associated with the id.
POST	/reviews	Add a review to the database.
DELETE	/reviews/<review_id>	Delete the review associated with the id.

Table 2.3: Student Endpoints

2.3 Data Connection

The backend establishes a secure connection to the **MongoDB Atlas Cloud** database using the **pymongo** library. The connection string is defined in the **create_app** function within app.py.

2.3.1 Client creation

The code creates a **MongoClient** instance using the connection string stored in the application configuration. This connection string includes authentication details such as username and password and the database cluster URL.

2.3.2 Database Attachment

The **MongoClient** instance is used to access the desired database. This database (residency_db) is then attached to the Flask application object (app). This allows other parts of the application to interact with the database through the app.db object.

Chapter 3

Implementation

3.1 System Architecture

The system architecture of this train management system was designed to ensure scalability and ease of integration. It consists of four major lavers :

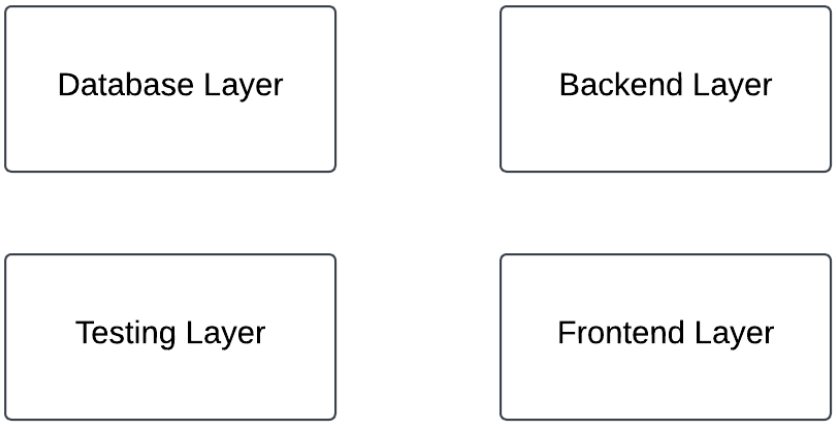

Figure 3.1: System Architecture

3.2 Database layer

A relational database was created in order to store structured data about residencies,users,and student_data and to define the relationships between each table, which makes data retrieval easier.

3.3 Back-end layer

The backend of the university residency system was developed using Flask, a lightweight and modular Python framework. The backend is structured into multiple files, each serving a distinct purpose to ensure clarity, maintainability, and scalability.

3.3.1 Back-end structure

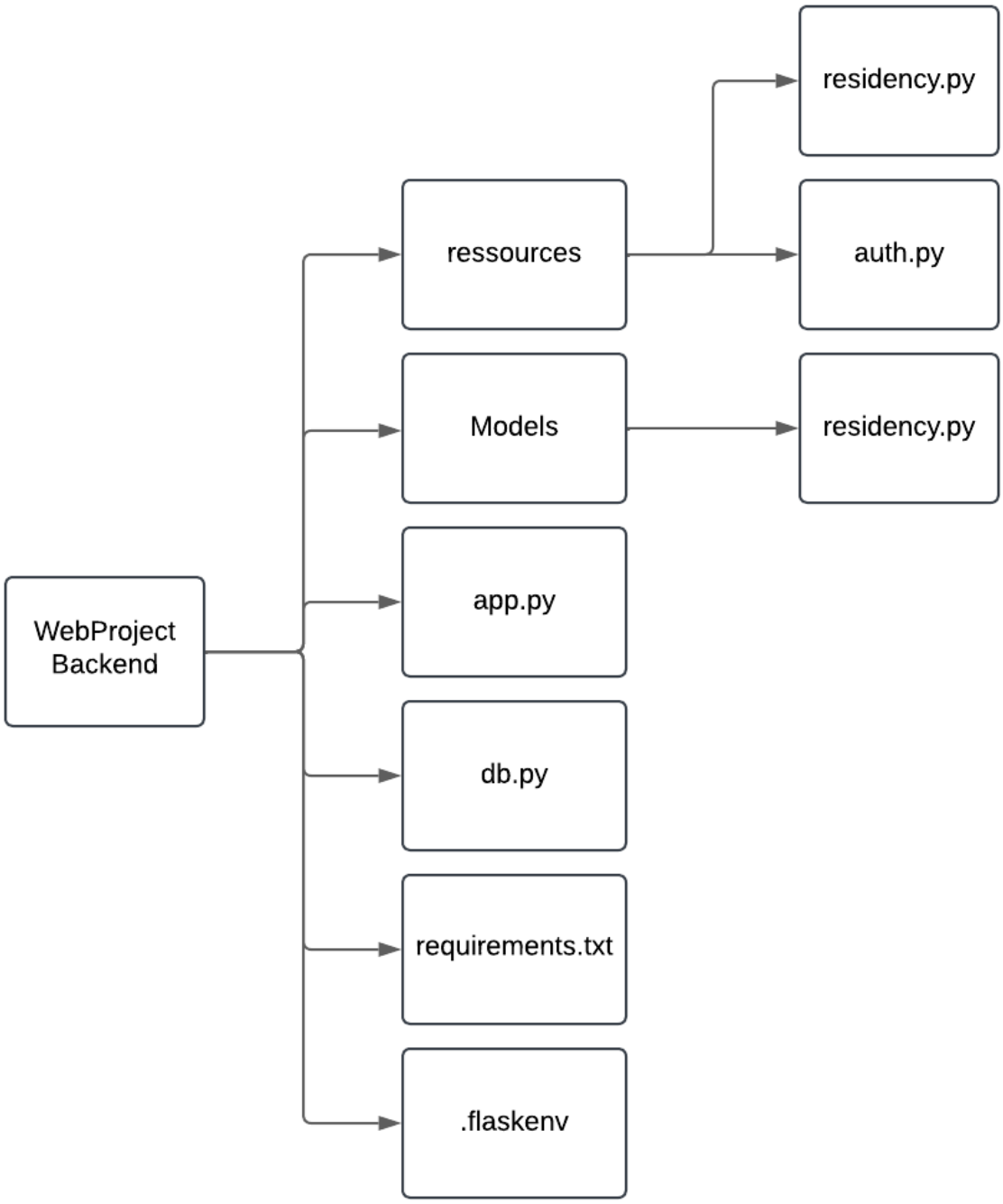

Figure 3.2: Back-end Files Structure

As shown in the figure 3.2, our back-end is structured as follows:

- **ressources** : A folder that contains :
  - **auth.py** : Handles user authentication and authorization. Includes endpoints for user registration (with password hashing), login (with JWT generation), and token validation for protecting routes..
  - **residency.py** : Defines RESTful endpoints for residency-related operations. Admins can manage residencies, while students can apply for residencies and manage their applications. Access is controlled via token-based authentication.
- **Models** : A folder that contains :
  - **residency.py** : Manages database interactions with the residencies and student_data collections. Functions include fetching, inserting, updating, and deleting residency data and managing student applications..
- **app.py** : The central file initializing the Flask application, setting up configurations (e.g., JWT secret key, CORS), connecting to MongoDB, and registering blueprints for authentication and residency management.
- **db.py** : This file Handles the initialization of the MongoDB connection using Flask-PyMongo, promoting modularity by isolating database setup.
- **requirements.txt** : This file lists the dependencies required to run the project.
- **.flaskenv** : This file configures the Flask application environment

3.3.2 Security measures

The system incorporates robust security measures to protect user data and ensure secure communication:

- **Password Hashing**: User passwords are securely stored in the database using the bcrypt hashing algorithm. This ensures that even if the database is compromised, passwords cannot be easily decrypted.

- **JWT Authentication**: Authentication is implemented using JSON Web Tokens (JWTs). Tokens are issued upon successful login and include encoded user information with an expiration time of one hour. The token_required decorator ensures that protected routes can only be accessed with a valid token.

- **Input Validation**: Input data is validated to prevent common attacks such as SQL injection and malformed requests.

- **CORS Configuration**: CORS is configured to allow cross-origin requests only from trusted domains, mitigating the risk of cross-site request forgery attacks.

- **Logging and Error Handling**: All errors are logged for debugging and user-friendly error messages are returned.

3.3.3 Virtual environment

A virtual environment was created to isolate project dependencies from the global Python environment. This ensures that the project runs with the exact versions of libraries specified, avoiding conflicts and compatibility issues.

3.4 Testing layer

Testing was conducted systematically using **Insomnia**, a tool designed for API development and debugging.
Each API endpoint was rigorously tested to ensure it met the required specifications, including returning correct data formats, handling valid and invalid inputs gracefully, and maintaining consistent performance.

3.5 Front-end layer

The frontend of the university residency web service was developed using Next.js with TypeScript and styled with Tailwind CSS. To ensure a seamless user experience, the frontend was designed to provide separate interfaces for administrators and students, enabling functionalities such as managing residencies and applying for accommodations. The frontend communicates with the backend through API endpoints.
While the project design and functionality were supervised by me, the actual development of the frontend was carried out using V0.dev to streamline implementation and save time. This ensured a professional and efficient output that met the project's requirements.

3.6 External APIs

Given the sensitive nature of our project, which involves handling students' personal data and is directly associated with a government ministry, the selection of an external API must be guided by stringent criteria such as security, reliability, and professionalism.
Google Accounts emerges as a highly suitable option for integration as an external API. It simplifies the login process for students, eliminating the need for them to create and remember an additional password. Moreover, Google Accounts is widely accessible and familiar to the current generation of students, ensuring ease of use and adoption.
Additionally, Google Accounts offers robust security measures, including encryption and advanced authentication protocols, which align with the project's objective of safeguarding sensitive personal data. However, it is important to note that utilizing this API requires a verified debit card for setup, which is currently unavailable to the project team. This limitation has been considered in evaluating the feasibility of its integration.

Threats to validity

This project, while addressing many of the challenges associated with traditional residency management, encountered several limitations during its development and implementation:

- **Data Collection**:

Initially, data collection proved challenging due to the lack of datasets specific to Tunisian university residencies. In addition, the Ministry of Higher Education maintains three separate websites for each region (North, South,and Central) leading to fragmented data and requiring significant effort for data consolidation. Another problem was missing information such as the lack of specification of the gender od the students admitted into each residency. To sum up, a sample of 19 gender-neutral residencies was created to be used in the project.

- **Technical Challenges**:

Working with Arabic datasets during data migration and processing presented minor technical challenges so a sample english dataset was used instead for the current release.

- **Future Enhancements:**

- **Streamlining Data Collection**: Future iterations will incorporate web scraping techniques to automate data collection from various sources, ensuring a more comprehensive and up-to-date database.

- **University-Specific Integrations**: Integrating with university databases will enable more precise residency assignments, allowing for the allocation of students to residencies within their respective universities. This integration will also facilitate the implementation of other services, such as transportation.

- **Enhanced Application Process**: To further streamline the application process and reduce paperwork, future versions will incorporate features such as online document uploads, allowing students to submit supporting documents electronically.

Conclusion

The development of the university residency management system reflects a comprehensive approach to addressing real-world challenges in the allocation and management of student residencies. By utilizing a modular and scalable architecture, the project integrates state-of-the-art technologies, ensuring functionality, security, and ease of use.
The backend, implemented with Flask, provides a robust framework for managing user authentication, data operations, and secure interactions. Security measures such as password hashing, JWT authentication, and environment variable management underscore the project's commitment to safeguarding sensitive information. Meanwhile, the implementation of RESTful APIs facilitates seamless communication with the frontend, enabling an intuitive user experience.
Furthermore, the use of MongoDB for data storage ensures flexibility in managing residency and student application data. The adoption of a virtual environment and detailed dependency management highlights the project's adaptability and ease of deployment. Rigorous testing strategies have validated the system's functionality and reliability, providing confidence in its performance under real-world conditions.
In conclusion, this project successfully meets its objectives by delivering a secure, efficient, and user-friendly platform for university residency management. The system's modularity and scalability position it for future enhancements, such as advanced analytics, AI-powered decision-making, or integration with broader university systems. This work not only addresses the immediate needs of administrators and students but also lays the groundwork for continuous innovation in residency services.

Bibliography

[1] Higher Education Ministry. Oouc, 2025. Accessed: 2025-01-19.

[2] Higher Education Ministry. Ooun, 2025. Accessed: 2025-01-19.

[3] Higher Education Ministry. Oous, 2025. Accessed: 2025-01-19.

[4] Postman. Rest api examples, 2025. Accessed: 2025-01-19.

[5] Splunk. Crud operations: A basic introduction, 2025. Accessed: 2025-01-19.