Engenharia e Gestão de Serviços

Universidade de Aveiro

João Ferreira, Rodrigo Rodrigues, Francisco Cardita, Guilherme Craveiro

Engenharia e Gestão de Serviços

Final Report

Universidade de Aveiro

João Ferreira, Rodrigo Rodrigues, Francisco Cardita, Guilherme Craveiro
(103625) joaop.ferreira@ua.pt, (102573) rlr@ua.pt, (103574) gjscraveiro@ua.pt

June 2, 2024

Índice

Chapter 1

About

Prison guards currently check their schedules on a sheet pinned to a dashboard. This method, while straightforward, has several limitations, including accessibility issues and potential for errors.

To address these challenges, there is a need for a more accessible and reliable system for registering events such as shift changes and meetings.

Implementing an efficient scheduling and event registration system will not only streamline operations but also enhance the overall effectiveness of prison management. This report explores the development and implementation of such a system, aiming to provide a comprehensive solution to the current scheduling inefficiencies.

Chapter 2

Provided Systems

To effectively address these needs, a comprehensive system has been developed to offer the following services:

Calendar: Users can check their work schedule and respective shifts. This feature allows for easy access to current and future schedules, ensuring guards are always informed about their duties. Additionally, users can also schedule new shifts as needed.

Authentication: Each user has a pre-defined account. By using their credentials, they can securely access the system. This ensures that only authorized personnel can view and modify their schedules, maintaining the integrity and confidentiality of the information.

Notification: Users can set their notification preferences, choosing the advance notice time and the method of notification (email or text message). This feature ensures that users receive timely reminders about important events such as meetings and shift changes. Notifications are automatically associated when events are added to the calendar, enhancing the reliability and effectiveness of the scheduling system.

Chapter 3

Architecture

ALGUEM Q ESCREVA QQR COISA

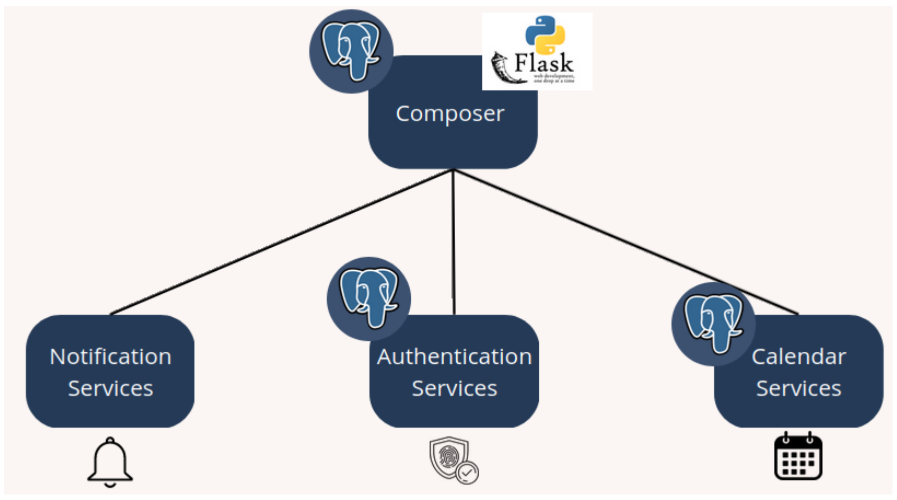


Fig.1: System Architecture

Chapter 4

Authentication

We developed a sophisticated authentication technique that makes use of multiple important technologies in order to guarantee safe and effective system access:

OAuth 2.0 Framework: The framework for authentication that we employed was OAuth 2.0. Users can give third parties access to their resources without disclosing login information. This guarantees that the authentication procedure is safe and easy to use.

Flask API: Our authentication microservice's central component, Flask API, controls the authentication procedure and storage for the user information.

PostgreSQL Database: Our authentication system's database is managed using PostgreSQL.

Flask, PostgreSQL, and OAuth 2.0 work together to offer a complete and safe authentication solution that protects critical data integrity and confidentiality and limits system access to only authorised individuals.

4.1 Login Methods

Our solution allows users to authenticate themselves using a variety of techniques, giving them flexibility and convenience:

Self Authentication: Users can interact with the Flask web app via browser and log in using their accounts, which are securely verified against our authentication system database. The user can create their account, and the information is going to be hashed and stored in the authentication database (SOA).

Google Authentication: We support Google Authentication in addition to self-authentication. By using the OAuth 2.0 protocol, this solution enables users to log in with their Google accounts. The Google OAuth server receives the users' redirection. Users are then sent back to the application based on how the Google Developer Console is configured.

These login methods ensure that users have multiple secure and convenient options to access the system, enhancing the overall user experience.

4.2 Endpoints

- **authentication/login [POST, GET]** - Self authentication and respective token after successful login
- **authentication/signup [POST]** - Create a new user
- **authentication/login/google [POST]** - Create a new user
- **authentication/logout [GET]** - Logout current user

4.3 Authentication Deployment

The deployment yaml follows the gic repository, implementing the services and the database using secrets for better security.

Coming from a containerized deployment the only changes where in the endpoint name, adding the name defined in the location block behind the already defined endpoints, actions in the front end and some url's that redirect to other API's. The configMap will start the PostgreSQL inside the kubernetes deployment. The nginx configuration is also based on the gic repository altering the server name and proxy pass to point to the correct upstream.

The service is then exposed with Traefik and nginx and accessible with the http://grupo8-egs-deti.ua.pt/authentication

Chapter 5

Calendar

To build the calendar feature of our system, we relied on the following technologies:

evoCalendar: We employed the evoCalendar JavaScript plugin to construct the calendar interface. This plugin provides a user-friendly and visually appealing way to display schedules and events, enhancing the overall user experience.

FastAPI: To establish the backend API for managing calendar-related functionalities, we utilized FastAPI, a modern web framework for building APIs with Python.

PostgreSQL: For data storage and management, we chose PostgreSQL as our database system.

These technologies combined enable us to provide a robust and efficient calendar solution, empowering users to manage their schedules effectively within the system.

5.1 Endpoints

5.1.1 GET /v1/schedules

Summary: List all schedules
Operation ID: listSchedules
Responses:

- 200: A list of schedules
- 403: Invalid or expired token

Description: Retrieve a list of all schedules.

5.1.2 POST /v1/schedules

Summary: Create a new schedule
Operation ID: createSchedule
Request Body:

- application/json: Schedule

Responses:

- 201: Schedule created
- 403: Invalid or expired token

5.1.3 GET /v1/schedule/{id}

Summary: Get a schedule by ID
Operation ID: getSchedule
Parameters:

- id: string (uuid)

Responses:

- 200: Schedule retrieved
- 403: Invalid or expired token
- 404: Schedule not found

5.1.4 PUT /v1/schedule/{id}

Summary: Update a schedule by ID
Operation ID: updateSchedule
Parameters:

- id: string (uuid)

Request Body:

- application/json: ScheduleUpdate

Responses:

- 200: Schedule updated
- 403: Invalid or expired token
- 404: Schedule not found

5.1.5 DELETE /v1/schedule/{id}

Summary: Delete a schedule by ID
Operation ID: deleteSchedule
Parameters:

- id: string (uuid)

Responses:

- 204: Schedule deleted
- 403: Invalid or expired token
- 404: Schedule not found

5.1.6 GET /v1/schedules/{person_name}

Summary: List all schedules for a person
Operation ID: listSchedulesByPersonName
Parameters:

- person_name: string

Responses:

- 200: A list of schedules
- 403: Invalid or expired token
- 404: No schedules found for this person

5.2 Database Schemas

5.2.1 Schedule

- id: string (uuid)
- title: string
- description: string
- date: string (date-time)
- person_name: string

5.2.2 ScheduleUpdate

- title: string
- description: string
- date: string (date-time)
- person_name: string

Chapter 6

Notifications

To implement the notification system in our application, we utilized the following technologies:

FastAPI Framework: We employed the FastAPI framework to build the API responsible for handling notifications.

SMTP: To send email notifications, we utilized the Simple Mail Transfer Protocol (SMTP).

Twilio: For sending SMS notifications, we integrated the Twilio cloud communications platform into our system.

By leveraging FastAPI, SMTP, and Twilio, we have developed a robust notification system that enhances the user experience by providing timely updates and alerts via both email and SMS.

6.1 Endpoints

For the sake of simplicity, this service has only one endpoint to handle notifications.

6.1.1 POST /api/v1/notifications

Summary: Create notification based on user preferences

Params

```
notification_data = {
    "username": "john_doe",
    "email": "john@example.com",
    "phoneNumber": "+1234567890",
    "systemName": "MySystem",
    "notificationContent": {
        "subject": "Test Notification",
        "body": "This is a test notification."
    },
    "preferences": {
        "smsEnabled": True,
        "emailEnabled": True,
        "pushEnabled": False
    }
}
```

Description: The notification service sends email notification through **SMTP** to the email provided by user in the request body. As for the SMS, it uses the third-party service **Twilio**, sending a request with body, the sender and recipient. Despite the field «pushEnabled», the logic to send push notifications is not implemented. It is, however, future work and would be built upon **FCM**(Firebase Cloud Messaging).

Chapter 7

Composer

For the composer feature, we used the following technologies:

PostgreSQL: As the backbone of our database system, PostgreSQL ensures efficient and reliable data storage and management for the composer module.

Flask + Python: We leveraged Flask, a lightweight and versatile web framework for Python, to build the backend logic of the composer module.

HTML + JavaScript + CSS: For the frontend development of the composer module, we utilized a combination of HTML, JavaScript, and CSS.

By integrating PostgreSQL for data storage, Flask for backend logic, and HTML, JavaScript, and CSS for frontend development, we have developed a robust and user-friendly composer module that empowers users to create and manage content effectively within our system.

7.1 Authentication Service Integration

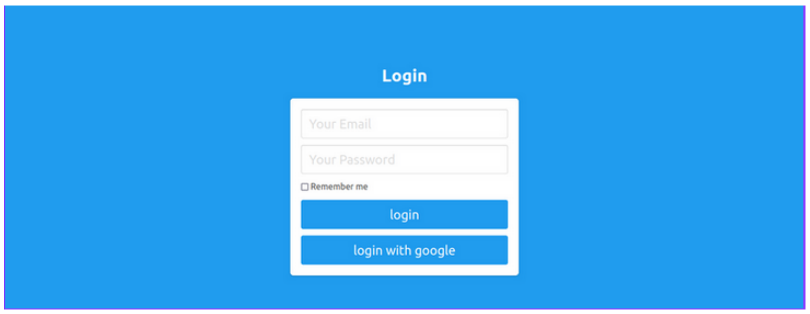
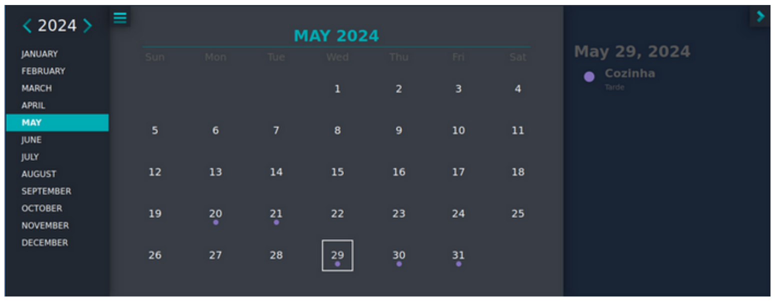


Fig.2: Authentication Service Integration

7.2 Calendar Service Integration



Sun	Mon	Tue	Wed	Thu	Fri	Sat
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

Fig.3: Calendar Service Integration

7.3 Notification Service Integration

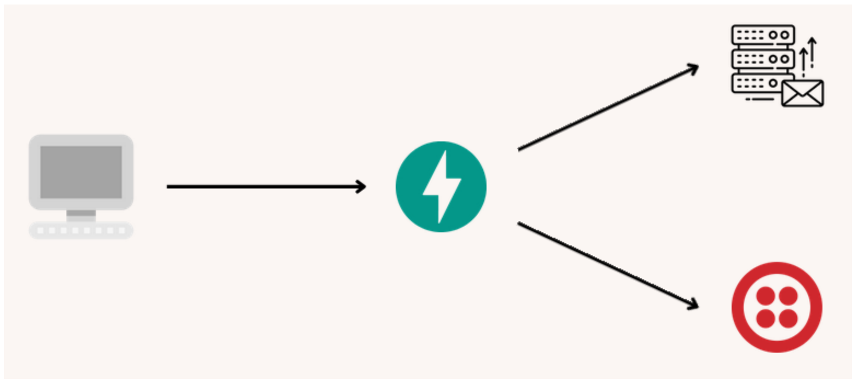


Fig.4: Notification Service Integration

Chapter 8

Link to the repository

CellWatch - REPOSITORY

Chapter 9

Contributions

This project was a collaborative effort between João Ferreira, Rodrigo Rodrigues, Francisco Cardita and Guilherme Craveiro. Each team member contributed significantly to the various stages of the project.

Our Contributions are:

- João Ferreira (103625) - 25%
- Rodrigo Rodrigues (102573) - 25%
- Francisco Cardita (97640) - 25%
- Guilherme Craveiro (103574) - 25%