



**Tunis Business  
University of Tunis**

## **REST API TuniChampion**

### **IT325 FINAL PROJECT REPORT**

**Submitted By:**

Amal Jawahdou

Major : Information Techonology

Minor : Business Analytics

January, 2024



## **Declaration of Academic Ethics**

I, Amal Jawahdou, declare that this written report, submitted for the IT325 subject's final project, is my own work that reflects my own ideas in my own way, and that any external ideas or words included in this paper have been properly cited and acknowledged respectively.

I affirm that I have adhered to respect all the academic ethics of honesty and integrity. And, I have ensured that all the mentioned information in this report is free from misguidance, fallacies, and misrepresentation.

I am fully cognizant that any violation of these ethical principles may lead to serious disciplinary actions, and I am committed to accept the consequence of any breaches.

Amal Jawahdou

**DATE:** January, 2024

## **Abstract**

[1]Tunisian athletes continue to shine, at home and abroad, in a variety of [2]sports categories. Yet because of a lack of understanding among the wider Tunisian masses, there is no fully comprehensive awareness or access to their works. This is all the more surprising when you consider that we are in an age of rapid development for [3]multimedia communications and online services, yet there still is no single platform focused entirely on Tunisian athletes achievements, their upcoming events or competitions to which you can [4]book tickets.

The lack of this kind of centralized and easily accessible information powerfully suppresses the sporting participation rate. Moreover it prevents these athletes from receiving support that they deserve at home. This growing gap highlights the pressing need for a grassroots specialized channel. This sort of a platform would be like a spotlight for the Tunisian athletes, letting supporters on both sides hold together in one place can only allow even more people to follow them throughout from start to finish.

We are ready to fulfil this demand by creating a customized, Tunisia-oriented restful application programming interface (API) specifically designed for that purpose. The goal of this project is not only to honor their achievements but also foster a feeling of solidarity, promote interest in Tunisian sports and even more importantly [5]increase support for athletes that represent Tunisia.

***Keywords :: Tunisian athletes, Sports , Multimedia communications, book ticketing ,increase support***

# TABLE OF CONTENTS

<b>Declaration of Academic Ethics</b>	<b>i</b>
<b>Abstract</b>	<b>ii</b>
<b>1 INTRODUCTION</b>	<b>1</b>
1.1 Background . . . . .	1
1.2 Motivation . . . . .	1
1.3 Objectives . . . . .	1
1.4 Problem statement . . . . .	2
1.5 Scope of Project . . . . .	2
<b>2 Work Explanation</b>	<b>1</b>
2.1 Database Management . . . . .	1
2.1.1 Database Structure . . . . .	1
2.1.2 Database Management System : MySQL . . . . .	1
2.1.3 Database migrations with Alembic and Flask-Migrate . . . . .	2
2.2 Python / Flask Contribution . . . . .	2
2.2.1 Athlete Requests . . . . .	3
2.2.2 Event Requests . . . . .	5
2.2.3 Ticket Requests . . . . .	6
2.2.4 Payment Requests . . . . .	8
2.2.5 User Requests . . . . .	10
2.3 User Authentication JWT . . . . .	13
2.4 Insomnia Testing . . . . .	14
2.5 Swagger API Documentation . . . . .	15
2.6 HTML/CSS/JS Front-end . . . . .	15
2.7 Git / GitHub for Version Control . . . . .	16
2.8 Render for Hosting . . . . .	16
<b>3 Future Enhancements</b>	<b>17</b>
<b>4 CONCLUSION</b>	<b>18</b>
<b>A APPENDIX</b>	<b>19</b>

# **1. INTRODUCTION**

In this report, I delved into the conceptualization and implementation of an innovative platform TuniChampion an API designed to revolutionize the accessibility and management of Tunisian athletes' information, their achievements, upcoming events, and seamless ticket booking experiences.

## **1.1. Background**

Tunisian athletes has rich history of athletic brilliance on both domestic and global levels across various sports disciplines. Despite these achievements, a notable disparity exists within the realm of accessible information about these athletes. In an era characterized by technological advancements and online connectivity, a dedicated platform spotlighting Tunisian athletes and their accomplishments remains conspicuously absent.

## **1.2. Motivation**

The driving force behind my initiative stems from the observable absence of a consolidated platform sharing the achievements of Tunisian athletes. TuniChampion responds to the growing need for a centralized space dedicated solely to the achievements of these athletes. In today's dynamic landscape, characterized by the quest for expedited services and accurate information, TuniChampion stands poised to bridge this informational void.

## **1.3. Objectives**

1. Enhancing accessibility to Tunisian athletes' information and events while fostering a stronger connection between the athletes and their supporters.
2. Developing an API that streamlines processes related to athlete information management, event tracking, and convenient ticket booking.

#### **1.4. Problem statement**

The absence of a dedicated platform documenting Tunisian athletes' achievements TuniChampion aims to address this by consolidating athlete information and event details, eliminating the current information gap.

#### **1.5. Scope of Project**

TuniChampion aims to build a Restful API that brings together knowledge about Tunisian athletes , relative event, and management of easy ticket reservations for those events.

To achieve this, the project uses Visual Studio Code with Python and Flask as the main back-end framework and SQL-Alchemy for managing the database.

It also prioritizes security by utilizing Passlib for password hashing and JWT Extended for authentication.

On the front-end side, the project is developed with HTML, CSS, and JavaScript, using Fetch API to communicate with the back-end API asynchronously.

In addition , I've used insomnia for testing and Git-hub and Git-Version Control to assist the work flow of the project . And finally, thanks to Render , I've succeeded to host the application platform interface online .

## 2. Work Explanation

This section will describe briefly and consistently the architecture of the project and the different implementations and technologies used in it.

### 2.1. Database Management

In this part, I will try to define the structure of the database used for this API and the kinds of relationships among its different dimensions.

**Data source** :: all data used in the database was generated by a generative pre-trained transformer artificial intelligence tool.

#### 2.1.1. Database Structure

This database schema diagram was generated on the phpMyAdmin administration platform. **phpMyAdmin is a free software tool written in PHP, intended to handle the administration of MySQL over the Web.[1]**

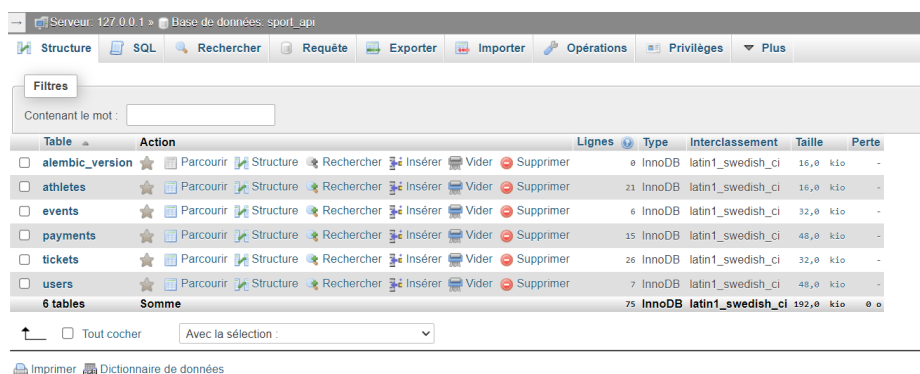


Table	Action	Lignes	Type	Interclassement	Taille	Perte
<input type="checkbox"/> alembic_version	Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	latin1_swedish_ci	16,0 kio	-
<input type="checkbox"/> athletes	Parcourir Structure Rechercher Insérer Vider Supprimer	21	InnoDB	latin1_swedish_ci	16,0 kio	-
<input type="checkbox"/> events	Parcourir Structure Rechercher Insérer Vider Supprimer	6	InnoDB	latin1_swedish_ci	32,0 kio	-
<input type="checkbox"/> payments	Parcourir Structure Rechercher Insérer Vider Supprimer	15	InnoDB	latin1_swedish_ci	48,0 kio	-
<input type="checkbox"/> tickets	Parcourir Structure Rechercher Insérer Vider Supprimer	26	InnoDB	latin1_swedish_ci	32,0 kio	-
<input type="checkbox"/> users	Parcourir Structure Rechercher Insérer Vider Supprimer	7	InnoDB	latin1_swedish_ci	48,0 kio	-
<b>6 tables</b>	<b>Somme</b>	<b>75</b>	<b>InnoDB</b>	<b>latin1_swedish_ci</b>	<b>192,0 kio</b>	<b>0 0</b>

Figure 2.1: phpMyAdmin TuniChampion Database's Tables

#### 2.1.2. Database Management System : MySQL

**MySQL is an Oracle-backed open source relational database management system (RDBMS) based on Structured Query Language (SQL).** [2] I have used MySQL to store the needed data in the tables to store, access and manipulate it easily.

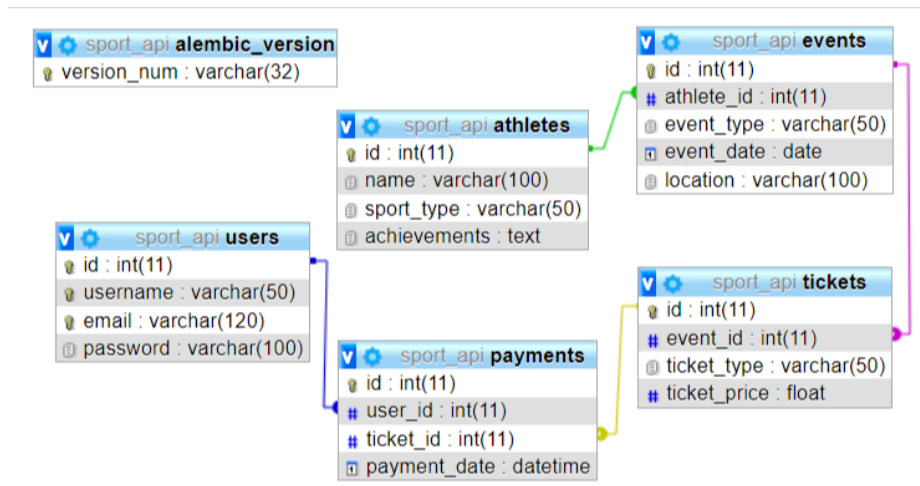


Figure 2.2: RDB Schema of the project's database

### 2.1.3. Database migrations with Alembic and Flask-Migrate

**Flask-Migrate** is an extension that integrates **Alembic**, a database migration tool, with **Flask**, making it easy to manage changes to your database schema over time.[3] In this section, my focus has been on managing the synchronization of database changes with the HTTP requests made to the API.

```

app.config["SQLALCHEMY_DATABASE_URI"] = 'mysql+pymysql://amal:amal@localhost/sport_api'
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
db.init_app(app)
api = Api(app)
migrate = Migrate()
  
```

## 2.2. Python / Flask Contribution

**Flask** is a web framework for Python that allows developers to build lightweight web applications quickly and easily.[4]

```

1 FLASK_APP = app
2 FLASK_DEBUG = True
  
```



### 2.2.1. Athlete Requests

– Get all athletes data :

```
@athlete_bp.response(200)
def get(self):
    athletes = Athlete.query.all()
    serialized_athletes = [serialize_athlete(athlete) for athlete in athletes]
    return jsonify(serialized_athletes), 200
```

– Create new athlete :

```
@athlete_bp.arguments(AthleteSchema)
@athlete_bp.response(201)
def post(self, athlete_data):
    athlete = Athlete(**athlete_data)
    try:
        db.session.add(athlete)
        db.session.commit()
        return {"message": "Athlete created"}, 201
    except SQLAlchemyError:
        abort(500, message="An error occurred while adding the athlete.")
```

– Get specific athlete data by id :

```
@athlete_bp.route("/athletes/<int:athlete_id>")
class SingleAthleteView(MethodView):
    @athlete_bp.response(200)
    def get(self, athlete_id):
        athlete = Athlete.query.get_or_404(athlete_id)
        serialized_athlete = serialize_athlete(athlete)
        return jsonify(serialized_athlete), 200
```

– Update specific athlete by id :

```
@athlete_bp.arguments(AthleteSchema)
@athlete_bp.response(200)
def put(self, athlete_data, athlete_id):
    athlete = Athlete.query.get_or_404(athlete_id)
    if athlete:
        athlete.name = athlete_data["name"]
        athlete.sport_type = athlete_data["sport_type"]
        athlete.achievements = athlete_data["achievements"]
        db.session.commit()
        serialized_athlete = serialize_athlete(athlete)
        return jsonify(serialized_athlete), 200
    else:
        abort(404, message="Athlete not found")
```

– Delete specific athlete by id :

```
@athlete_bp.response(204)
def delete(self, athlete_id):
    '''jwt = get_jwt()
    if not jwt.get("is_admin"):
        abort(401, message="Admin privilege required.")'''
    athlete = Athlete.query.get_or_404(athlete_id)
    db.session.delete(athlete)
    db.session.commit()
    return {"message": "Athlete deleted"}, 201
```

### 2.2.2. Event Requests

- Get all events data :

```
@event_bp.route("/events")
class EventListResource(MethodView):
    @event_bp.response(200, EventSchema(many=True))
    def get(self):
        events = Event.query.all()
        serialized_events = [serialize_event(event) for event in events]
        return jsonify(serialized_events), 200
```

- Create new event :

```
@event_bp.arguments(EventSchema)
@event_bp.response(201)
def post(self, data):
    event = Event(**data)
    db.session.add(event)
    db.session.commit()
    return {"message": "Event created"}, 201
```

- Get specific event data by id :

```
@event_bp.route("/events/<int:id>")
class EventResource(MethodView):
    # Get an event by ID
    @event_bp.response(200, EventSchema)
    def get(self, id):
        event = Event.query.get_or_404(id)
        return event, 200
```

– Update specific event by id :

```
# Update an event by ID
@event_bp.arguments(EventSchema)
@event_bp.response(200, EventSchema)
def put(self, data, id):
    event = Event.query.get_or_404(id)
    if event:
        event.athlete_id = data["athlete_id"]
        event.event_type = data["event_type"]
        event.event_date = data["event_date"]
        event.location = data["location"]
    else:
        event = event(id=id, **data)
    db.session.add(event)
    db.session.commit()
    return event
```

– Delete specific event by id :

```
# Delete an event by ID
@event_bp.response(204)
def delete(self, id):
    jwt = get_jwt()
    if not jwt.get("is_admin"):
        abort(401, message="Admin privilege required.")
    event = Event.query.get_or_404(id)
    db.session.delete(event)
    db.session.commit()
    return {"message": "event deleted"}, 204
```

### 2.2.3. Ticket Requests

– Get all tickets data :

```
@ticket_bp.route("/tickets")
class TicketListResource(MethodView):
    @ticket_bp.response(200, TicketSchema(many=True))
    def get(self):
        tickets = Ticket.query.all()
        return [ticket for ticket in tickets], 200
```

– Get specific ticket data by id :

```
@ticket_bp.response(200, TicketSchema)
def get(self, id):
    ticket = Ticket.query.get_or_404(id)
    return ticket , 200
```

– Get specific ticket data by event-id :

```
@ticket_bp.route("/tickets/event/<int:event_id>")
class TicketResource(MethodView):
    @ticket_bp.response(200, TicketSchema(many=True))
    def get(self, event_id):
        tickets = Ticket.query.filter_by(event_id=event_id).all()
        if not tickets:
            abort(404, message="No tickets found for this event")
        return [ticket for ticket in tickets], 200
```

– Post new ticket :

```
@ticket_bp.arguments(TicketSchema)
@ticket_bp.response(201)
def post(self, data):
    ticket = Ticket(**data)
    db.session.add(ticket)
    db.session.commit()
    return {"message": "Ticket created"}, 201
```

– Update specific ticket by id :

```
@ticket_bp.arguments(TicketSchema)
@ticket_bp.response(200, TicketSchema)
def put(self, data, id):
    ticket = Ticket.query.get_or_404(id)
    if not ticket:
        return abort(404, message="Ticket not found")

    event_id = data.get("event_id")
    existing_event = Event.query.get(event_id)
    if not existing_event:
        return abort(400, message="Invalid event_id provided")
    ticket.event_id = event_id
    ticket.ticket_type = data.get("ticket_type")
    ticket.ticket_price = data.get("ticket_price")
    db.session.commit()
    return ticket
```

– Delete specific ticket by id :

```
@ticket_bp.response(204)
def delete(self, id):
    jwt = get_jwt()
    if not jwt.get("is_admin"):
        abort(401, message="Admin privilege required.")
    ticket = Ticket.query.get_or_404(id)
    db.session.delete(ticket)
    db.session.commit()
    return {"message": "ticket deleted"}, 204
```

#### 2.2.4. Payment Requests

– Get all payment transactions data :

```
@payment_bp.response(200, PaymentSchema(many=True))
def get(self):
    payments = Payment.query.all()
    return [payment for payment in payments], 200
```

– Create new payment transaction :

```
@payment_bp.arguments(PaymentSchema)
@payment_bp.response(201, PaymentSchema)
def post(self, data):
    user_id = data.get("user_id")
    ticket_id = data.get("ticket_id")

    # Check if the provided user_id exists in the users table
    if db.session.query(User.query.filter_by(id=user_id).exists()).scalar():
        # Check if the provided ticket_id exists in the tickets table
        if db.session.query(Ticket.query.filter_by(id=ticket_id).exists()).scalar():
            new_payment = Payment(**data)
            db.session.add(new_payment)
            db.session.commit()
            return new_payment, 201
        else:
            return abort(400, message="Invalid ticket_id provided")
    else:
        return abort(400, message="Invalid user_id provided")
```

– Get specific payment transaction data by id :

```
@payment_bp.route("/<int:id>")
class PaymentResource(MethodView):
    @payment_bp.response(200, PaymentSchema)
    def get(self, id):
        payment = Payment.query.get_or_404(id)
        return payment, 200
```

– Update specific payment transaction by id :

```
@payment_bp.arguments(PaymentSchema)
@payment_bp.response(200, PaymentSchema)
def put(self, data, id):
    payment = Payment.query.get_or_404(id)
    if payment:
        payment.user_id = data.get("user_id", payment.user_id)
        payment.ticket_id = data.get("ticket_id", payment.ticket_id)
        payment.payment_date = data.get("payment_date", payment.payment_date)
        db.session.commit()
        return payment, 200
    else:
        abort(404, message="Payment not found")
```

- Delete specific payment transaction by id :

```
@payment_bp.response(204)
def delete(self, id):
    jwt = get_jwt()
    if not jwt.get("is_admin"):
        abort(401, message="Admin privilege required.")
    payment = Payment.query.get_or_404(id)
    db.session.delete(payment)
    db.session.commit()
    return {"message": "payment deleted"}, 204
```

### 2.2.5. User Requests

- Get all user data :

```
@user_bp.response(200, UserSchema(many=True))
#Get all users
def get(self):
    users = User.query.all()
    return [user for user in users], 200
```

- Get specific user data by id :

```
@user_bp.route("/<int:id>")
class UserResource(MethodView):
    @user_bp.response(200, UserSchema)
    #Get user by id
    def get(self, id):
        user = User.query.get_or_404(id)
        return user , 200
```



– Create new user :

```
@user_bp.arguments(UserSchema)
@user_bp.response(201, UserSchema)

#Create new user by admin
def post(self, data):
    user = User(**data)
    try:
        db.session.add(user)
        db.session.commit()
    except SQLAlchemyError:
        abort(500, message="An error occurred while adding the user.")
    return user , 201
```

– Update specific user by id :

```
@user_bp.arguments(UserSchema)
@user_bp.response(200, UserSchema)
#Update user by id
def put(self, data, id):
    user = User.query.get_or_404(id)
    if user:
        User.username = data["username"]
        User.email = data["email"]
        User.password = pbkdf2_sha256.hash(data["password"])
    else:
        user = User(id=id, **data)
    db.session.add(user)
    db.session.commit()
    return user , 200
```

– Delete specific user by id :

```
@user_bp.response(204)
#Delete user by id
def delete(self, id):
    user = User.query.get_or_404(id)
    db.session.delete(user)
    db.session.commit()
    return "user deleted", 204
```

## – Login :

```
@user_bp.route("/login", methods=['POST'])
class UserLogin(MethodView):
    @user_bp.arguments(UserLoginSchema)
    def post(self, user_data):
        user = User.query.filter(User.email == user_data["email"]).first()

        if user and pbkdf2_sha256.verify(user_data["password"], user.password):
            access_token = create_access_token(identity=user.id)
            return jsonify({"access_token": access_token, "user_id": user.id}), 200
        else:
            return jsonify({"message": "Invalid credentials"}), 401
```

## – Register :

```
@user_bp.route("/register", methods = ['POST'] )
class UserRegister(MethodView):
    @user_bp.arguments(UserSchema)
    #Register new user
    def post(self, user_data):

        if User.query.filter(User.username == user_data["username"]).first():
            abort(409, message="A user with that username already exists.")
        if User.query.filter(User.email == user_data["email"]).first():
            abort(409, message="A user with that email already exists.")

        user = User(
            username=user_data["username"],
            email = user_data["email"],
            password=pbkdf2_sha256.hash(user_data["password"]),
        )
        db.session.add(user)
        db.session.commit()

        return 'Registration successful', 200
```

### 2.3. User Authentication JWT

**JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely between two parties.[5]**

I've implemented JSON Web Token authentication to ensure the security of API access. Here is the corresponding implementation code:

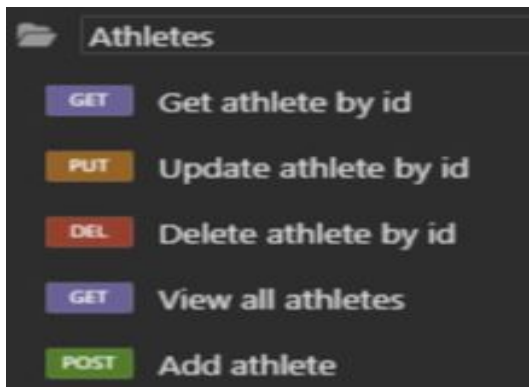
```
jwt = get_jwt()
if not jwt.get("is_admin"):
    abort(401, message="Admin privilege required.")
```

[illegible]

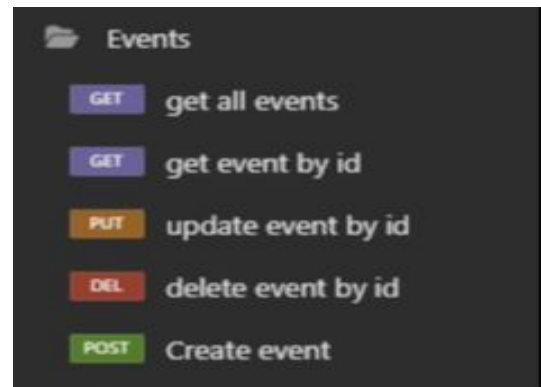
## 2.4. Insomnia Testing

**Kong Insomnia is a collaborative open source API development platform that makes it easy to build high-quality APIs — without the bloat and clutter of other tools.[6]**

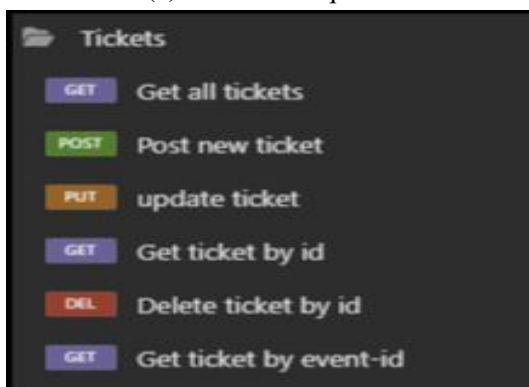
In this section, my focus has been on testing the HTTP requests made to the API.



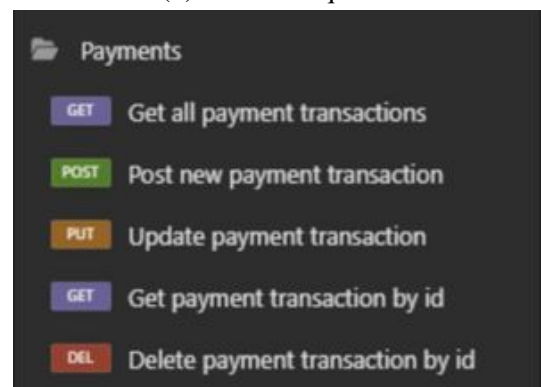
(a) Athletes Requests



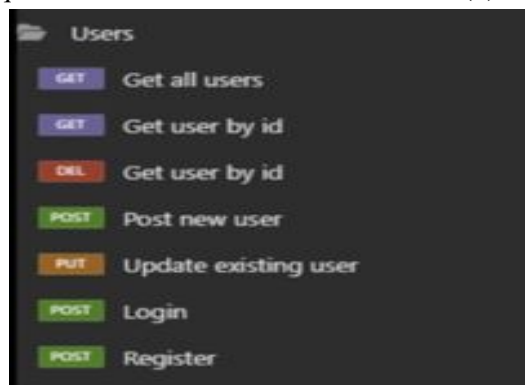
(b) Events Requests



(c) Tickets Requests



(d) Payments Requests

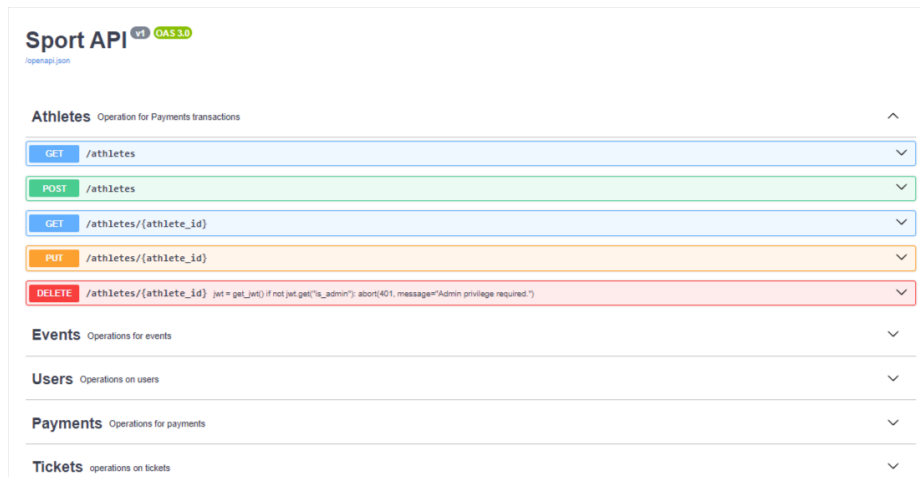


(e) Users Requests

Figure 2.3: Main captions for all endpoints.

## 2.5. Swagger API Documentation

Swagger API is a set of open-source tools built to help programmers develop, design, document, and use REST APIs. [7]



## 2.6. HTML/CSS/JS Front-end

In the purpose of empowering the created API , I have tried to build a simple interface using HTML/CSS/JS to test some endpoints such as registration, login, reservation, etc...

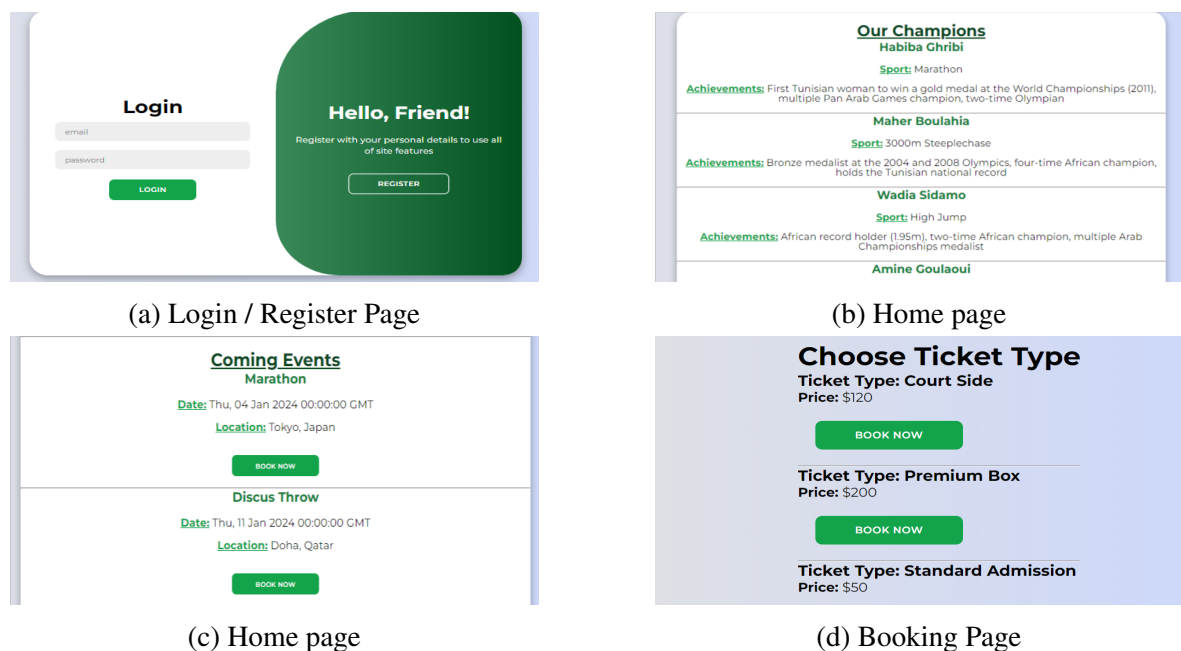
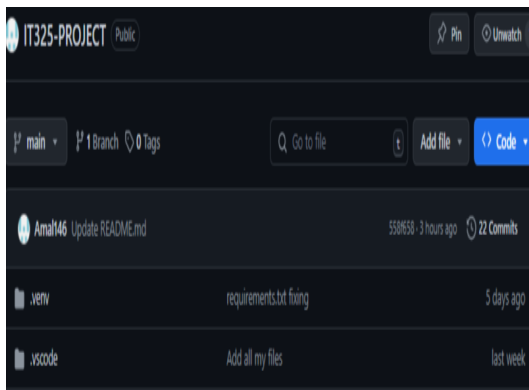


Figure 2.4

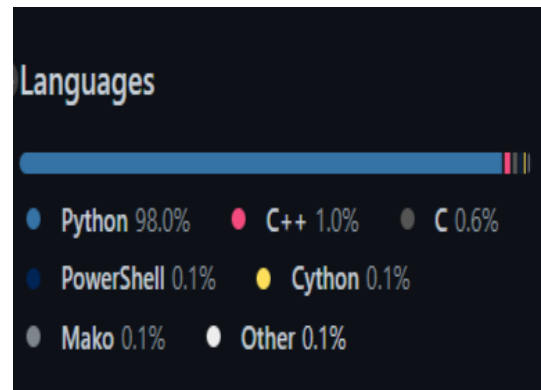
## 2.7. Git / GitHub for Version Control

Git is a free and open-source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.[8] GitHub, a for-profit company, offers a cloud-based Git repository hosting service, facilitating collaboration and code management.

The objective of my usage of Git and GitHub was to implement version control for the project, enabling efficient collaboration, tracking changes, and maintaining a history of the codebase.



(a)

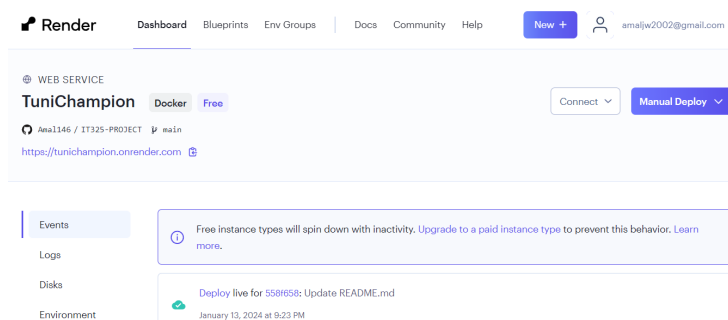


(b)

## 2.8. Render for Hosting

Render is a unified cloud platform designed for building and running apps and websites. It provides free TLS certificates, a global CDN, private networks, and supports auto deploys from Git repositories.[9]

The objective of using Git and GitHub was to implement version control for the project, enabling efficient collaboration, tracking changes, and maintaining a history of the codebase. This approach ensured a smooth development process and easy integration of contributions from multiple team members.



### 3. Future Enhancements

In the future, there are several enhancements that could be made to improve the functionality and user experience of this API. These include:

- **Improved Error Handling:** Implementing more robust error handling can make the API more resilient and easier for clients to use such as in the booking process or the Login one.
- **Rate Limiting:** To protect the API from abuse or excessive requests, rate limiting could be implemented.
- **Additional Endpoints:** Depending on the needs of the users, additional endpoints could be added to provide more functionality.
- **Performance Optimizations:** Performance of the API could be improved through various optimizations, such as caching or query optimization.
- **Integration with External APIs:** Exploring collaborations with external APIs can augment the richness of the data provided by the API. Integration with relevant external services such as google maps or any other sportive API can offer complementary information, expanding the scope and utility of the API.
- **User Authentication and Authorization:** Strengthening security measures through user authentication and authorization mechanisms ensures that sensitive data is accessed only by authorized individuals. OAuth2.0 addition enhances the overall security posture of the API.

## 4. CONCLUSION

In conclusion, the TuniChampion API project has been driven by the necessity to bridge the informational gap surrounding Tunisian athletes. The achievements and milestones reached in the development process lay the foundation for a comprehensive platform that not only celebrates athletic excellence but also fosters a sense of solidarity and support.

It was such an honor that the first API project I build was for the support of my nation. Despite, all the hard times and the difficulties I've faced to develop it, I am incredibly grateful for the experience and the opportunity to contribute in this meaningful way.

I extend my sincere gratitude to my esteemed class professor Dr. Montassar Ben Messaoud whose guidance and mentorship have played a pivotal role in shaping this project. Their insightful feedback and encouragement have been invaluable, contributing significantly to the success and refinement of the TuniChampion API.



## A. APPENDIX

### References

- [1] “Mysql.” <https://www.techtarget.com/searchoracle/definition/MySQL>.
- [2] “phpmyadmin.” <https://www.phpmyadmin.net/>.
- [3] “Flask-migrate.” <https://stackreveal.com/blog/database-migrations-with-flask-migrate/>.
- [4] “Flask.” <https://www.geeksforgeeks.org/flask-tutorial/>.
- [5] “Json web tokens.” <https://jwt.io/>.
- [6] “Kong insomnia.” <https://insomnia.rest/>.
- [7] “Swagger api.” <https://blog.hubspot.com/website/what-is-swagger/>.
- [8] “Git-version control.” <https://git-scm.com/>.
- [9] “Render.” <https://render.com/>.