

Tennis Grand Slams Titles REST API

IT325 Web Services Final Project

by

Mohamed Aziz Ajmi

January 2024

BA/IT Senior Student



Information Technology Minor

Tunis Business School

Ben Arous, TUNISIA.

2023-2024

Declaration of Academic Ethics

I, Mohamed Aziz Ajmi, hereby declare that the work presented in this report, titled "Tennis Grand Slams REST API Report," is my own. I confirm that I have not copied or used the work of others without proper acknowledgment. The ideas presented in this report are the result of my own original research and analysis. Any contributions from external sources are clearly and appropriately cited. I understand the importance of academic integrity and pledge to uphold the highest standards of honesty and ethical conduct in all aspects of my academic work.

Date: January 21st, 2024

Mohamed Aziz Ajmi

Abstract

The "Tennis Grand Slams REST API Report" details the development and implementation of a Flask web application that serves as a REST API for retrieving information about tennis Grand Slams champions. This report provides an overview of the project's objectives, methodologies, and outcomes.

The application utilizes various web development technologies, including Flask for the backend, MySQL for the database, and Streamlit for the user interface. Security measures such as JWT access tokens are implemented to ensure secure communication.

The report discusses the HTTP methods used in the API, the structure of the backend database, the security methods employed, and the user interface created using Streamlit. Screenshots and examples illustrate the functionalities of the application.

Through this project, insights into web services, database management, and API development are gained. The conclusion reflects on the achievements, challenges faced, and potential future enhancements.

Contents

Abstract	2
1 Introduction	4
2 Explanation of the work carried out	5
2.1 BeautifulSoup/Requests Contribution	5
2.2 MySQL Contribution	6
2.3 Flask Contribution	6
2.4 Python Contribution	7
2.5 All the HTTP Methods used	7
2.5.1 GET Requests	7
2.5.2 POST Requests	10
2.5.3 PUT Requests	12
2.5.4 DELETE Requests	13
2.6 Insomnia Contribution	13
2.7 Streamlit Contribution	14
2.8 Swagger Contribution	14
2.9 Database Structure	16
2.9.1 Tables	16
3 Conclusion	18

Chapter 1

Introduction

My final Project for the IT325 Web Services course this semester consists of a RESTful API developed using Flask Python and Beautiful soup. The resources were stored thanks to the implementation of MySQL.

I have also used technologies such as Insomnia, VSCode, Swagger documentation and Git-Version Control to maximize the project's quality.

This project contains all the CRUD Operations, secured with session authentication and hashing passwords. Multiple Insomnia Snippets were also used in order to test this project's efficiency.

This project aims to develop a comprehensive and user-friendly Tennis Grand Slams API that provides details on champions, competitions history and a way to predict future champions.

It generates an updated database of titles and champions and it can be accessed only through registered users.

Chapter 2

Explanation of the work carried out

2.1 BeautifulSoup/Requests Contribution

Beautiful Soup is a Python library that is used for web scraping purposes. It allows a programmer to extract data from a website by parsing the HTML or XML of a webpage. BeautifulSoup provides a number of useful methods and attributes to navigate, search, and modify the parse tree, making it easier to extract the data you're interested in. [1]

The requests library is a popular Python library for making HTTP requests. It allows you to send HTTP requests using Python, and it provides a number of useful methods for handling the response [2].

I have used BeautifulSoup/Requests to extract data of historical tennis champions and the history of grand slams winners since 1897. Implementation code :

```
apiproject > ws2.py > ...
1  import requests
2  from bs4 import BeautifulSoup
3  import pandas as pd
4  import lxml
5  import mysql.connector
6
7
8  url="https://en.wikipedia.org/wiki/List_of_Grand_Slam_men%27s_singles_champions"
9  data = requests.get(url)
10 soup=BeautifulSoup(data.text,'html.parser')
11 tables= soup.findAll('table')
12 table=tables[4]
13 t=table.findAll('tr')
```

2.2 MySQL Contribution

MySQL is a popular open-source relational database management system (RDBMS). It is commonly used for storing, managing, and retrieving data in various applications, including web development projects. MySQL is known for its performance, reliability, and ease of use. [?]

I have used the web-scraped data to build a database containing the tennis champions. Implementation code :

```
1  import requests
2  from bs4 import BeautifulSoup
3  import pandas as pd
4  import lxml
5  import mysql.connector
6
7  connection = mysql.connector.connect(
8      host="127.0.0.1",
9      port="3306",
10     user="root",
11     password="",
12     database="grand_slams")
13 cursor=connection.cursor()
14 cursor.execute("CREATE TABLE champions (Year VARCHAR(255) PRIMARY KEY, Australian_Open VARCHAR(255))")
15
16 url="https://en.wikipedia.org/wiki/List_of_Grand_Slam_men%27s_singles_champions"
17 data = requests.get(url)
```

2.3 Flask Contribution

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. [3]

I used Flask to build my API using decorators as endpoints after connecting it to the database. I also used sessions to make sure that the user needs to login in order to be able to add a prediction of champions to the database, update it or delete it. Jsonify was also imported from flask which helped me convert Python dictionaries to JSON object and send it as the response for an HTTP request.

2.4 Python Contribution

Python is a general-purpose language, which means it can be used to build just about anything, from web applications to desktop applications to scientific applications and data analysis. [4]

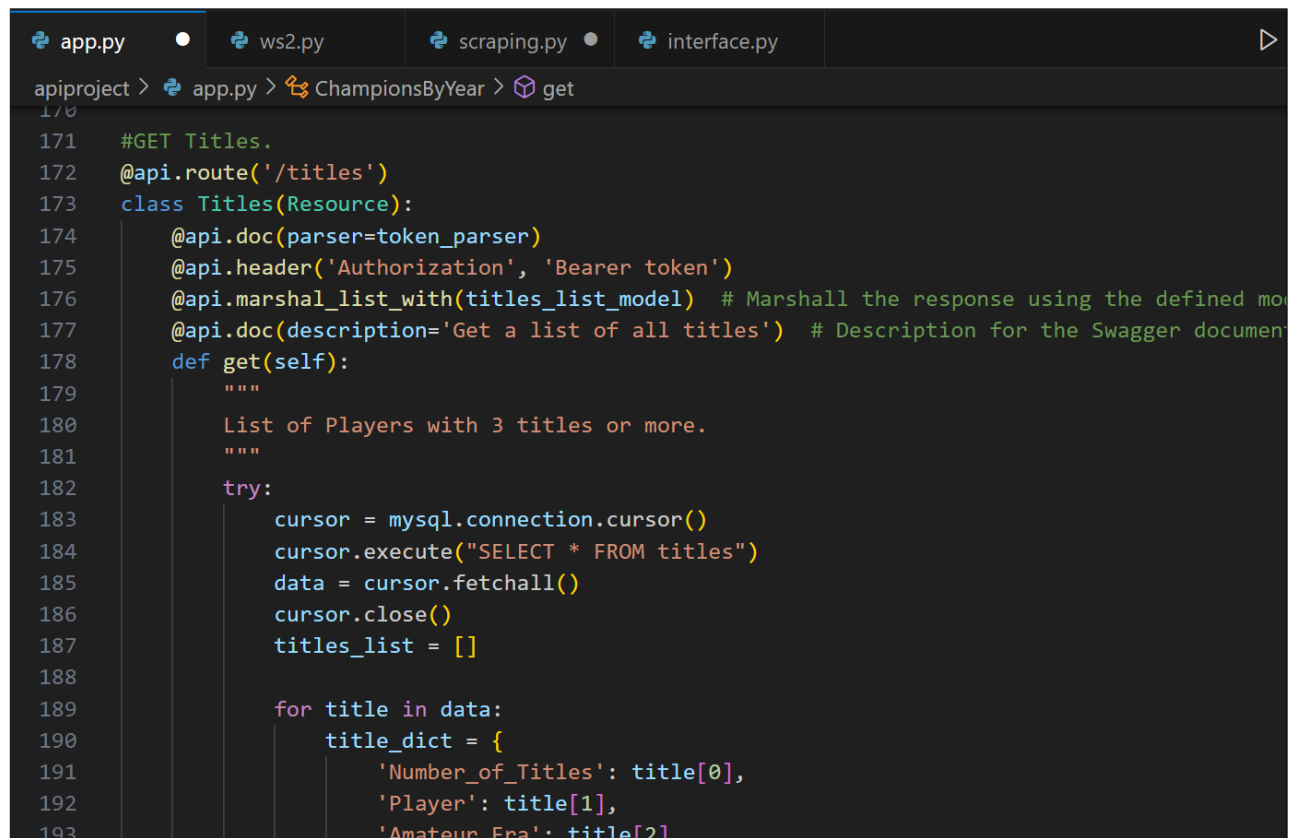
After choosing Flask as my web framework, creating a virtual environment, I used python to create my API's endpoints using specific URL. I added the code to handle requests and responses such as parsing request data, validating input, or interacting with a database.

2.5 All the HTTP Methods used

2.5.1 GET Requests

GET /titles

Get the list of Players with 3 or more Grand Slams Titles won in their career. You need to be logged in to get access.



```
app.py  ws2.py  scraping.py  interface.py
apiproject > app.py > ChampionsByYear > get
1/0
171 #GET Titles.
172 @api.route('/titles')
173 class Titles(Resource):
174     @api.doc(parser=token_parser)
175     @api.header('Authorization', 'Bearer token')
176     @api.marshal_list_with(titles_list_model) # Marshall the response using the defined model
177     @api.doc(description='Get a list of all titles') # Description for the Swagger document
178     def get(self):
179         """
180         List of Players with 3 titles or more.
181         """
182         try:
183             cursor = mysql.connection.cursor()
184             cursor.execute("SELECT * FROM titles")
185             data = cursor.fetchall()
186             cursor.close()
187             titles_list = []
188
189             for title in data:
190                 title_dict = {
191                     'Number_of_Titles': title[0],
192                     'Player': title[1],
193                     'Amateur Era': title[2],
```


GET /titles/<string:player name>

Get the total Grand Slams titles won by player. Once again, a login is required.

```
apiproject > app.py > ChampionsByYear > get
207
208 #GET Titles by Player:
209 @api.route('/titles/<string:player_name>')
210 class TitlesByPlayer(Resource):
211     @api.doc(parser=token_parser)
212     @api.header('Authorization', 'Bearer token')
213     @api.marshal_with(player_title_model) # Marshal the response using the defined model
214     @api.doc(params={'player_name': 'Player name to filter titles'}) # Description for the
215     def get(self, player_name):
216         """
217         Total Grand Slams Titles for each player.
218         """
219         try:
220             cursor = mysql.connection.cursor()
221             cursor.execute("SELECT Player, Titles FROM titles WHERE Player LIKE %s", ('%' +
222             result = cursor.fetchone()
223             cursor.close()
224             if result:
225                 database_player, count = result
226                 return {'database_player': database_player, 'argument_player': player_name,
227             else:
228                 return {'player': player_name, 'number_of_titles': 0}
229         except Exception as e:
230             print(f"Error in get_titles_by_player route: {str(e)}")
231             return {'message': 'Internal Server Error', 'message_code': 500}
```

GET /champions

Get the list of Grand Slams Champions from 1897 (start of the competitions) till 2023.

```
apiproject > app.py > ChampionsByYear > get
234 #GET Champions.
235 @api.route('/champions')
236 class Champions(Resource):
237     @api.doc(parser=token_parser)
238     @api.header('Authorization', 'Bearer token')
239     @api.marshal_list_with(champions_list_model) # Marshal the response using the defined
240     @api.doc(description='Get a list of all champions') # Description for the Swagger docum
241     def get(self):
242         """
243         Get Champions List.
244         """
245         try:
246             cursor = mysql.connection.cursor()
247             cursor.execute("SELECT * FROM champions")
248             data = cursor.fetchall()
249             cursor.close()
250
251             champions_list = []
252             for champion in data:
253                 champion_dict = {
254                     'Year': champion[0],
255                     'Australian_Open': champion[1],
256                     'Roland_Garros': champion[2],
257                     'Wimbledon': champion[3],
```

GET /champions/<int:year>

The user will get the list of Grand Slams Champions in a year of his choice.

```

apiproject > app.py > ChampionsByYear > get
267 #Get Champions by Year.
268 @api.route('/champions/<int:year>')
269 class ChampionsByYear(Resource):
270     @api.doc(parser=token_parser)
271     @api.header('Authorization', 'Bearer token')
272     @api.marshal_with(champion_by_year_model) # Marshal the response using the defined model
273     @api.doc(params={'year': 'Year to filter champions'}) # Description for the Swagger doc
274     def get(self, year):
275         """
276         Get champions by chosen year.
277         """
278         try:
279             cursor = mysql.connection.cursor()
280             cursor.execute("SELECT * FROM champions WHERE Year = %s", (year,))
281             data = cursor.fetchone()
282             cursor.close()
283
284             if data:
285                 champion_dict = {}
286                 'Year': data[0],
287                 'Australian_Open': data[1],
288                 'Roland_Garros': data[2],
289                 'Wimbledon': data[3],
290                 'US_Open': data[4]

```

2.5.2 POST Requests

POST /signup

SIGN-UP in the API after specifying a valid username or email and a password in the body of the request. The password entered will be hashed and stored in the database using bcrypt class from "flask bcrypt".

```

apiproject > app.py > ChampionsByYear > get
102 #SIGN-UP
103 @api.route('/signup')
104 class Signup(Resource):
105     @api.expect(signup_model) # Expecting the request payload to match the defined model
106     def post(self):
107         try:
108             data = request.get_json()
109
110             #Ensure all required data is present in the request
111             required_fields = ['username', 'password']
112             if not all(field in data for field in required_fields):
113                 return jsonify({'error': 'Incomplete data'}), 400
114
115             #Extract data from the request
116             username = data['username']
117             password = data['password']
118
119             #Hash the password before storing it in the database
120             hashed_password = bcrypt.generate_password_hash(password).decode('utf-8')
121
122             #Add the new user to the 'users' table
123             cursor = mysql.connection.cursor()
124             cursor.execute("INSERT INTO users (username, password) VALUES (%s, %s)", (username, hashed_password))
125             mysql.connection.commit()

```

POST /login

Use the credentials you entered when registering (username/email and password) to login to the API. This will create a session with username of the user. Upon successful login the user will receive an access token that will grant him access to all the other methods. Token expires after 15minutes.

```
apiproject > app.py > ChampionsByYear > get
138 @api.route('/login')
139 class Login(Resource):
140     @api.expect(login_model) # Expecting the request payload to match the defined model
141     def post(self):
142         try:
143             data = request.get_json()
144             username = data.get('username')
145             entered_password = data.get('password')
146             cursor = mysql.connection.cursor()
147             cursor.execute("SELECT * FROM users WHERE username = %s", (username,))
148             username = cursor.fetchone()
149
150             if username:
151                 #Extract the hashed password from the database
152                 hashed_password = username[1]
153                 bpw=bcrypt.check_password_hash(hashed_password, entered_password)
154                 #Check if the entered password matches the stored hash
155                 if bpw:
156                     #Passwords match, authentication successful
157                     access_token = create_access_token(identity=username)
158                     return jsonify(access_token=access_token)
159                 else:
160                     #Passwords don't match
161                     return jsonify({"message": "wrong username or password"})
```

POST /champions

Predict Grand Slams Winners in the following years (starting from 2024 ongoing).

```

app.py ws2.py scraping.py interface.py
apiproject > app.py > ChampionsPrediction > post
301 @api.route('/champions')
302 class ChampionsPrediction(Resource):
303     @api.expect(champions_prediction_model) # Expecting the request payload to match the defined model
304     @api.doc(parser=token_parser)
305     @api.header('Authorization', 'Bearer token')
306     @api.doc(description='Predict champions for the next years') # Description for the Swagger documentation
307     def post(self):
308         try:
309             data = request.json
310             # Ensure all required data is present in the request
311             required_fields = ['Year', 'Australian_Open', 'Roland_Garros', 'Wimbledon', 'US_Open']
312             if not all(field in data for field in required_fields):
313                 return {'error': 'Incomplete data'}, 400
314             # Extract data from the request
315             year = data['Year']
316             australian_open = data['Australian_Open']
317             roland_garros = data['Roland_Garros']
318             wimbledon = data['Wimbledon']
319             us_open = data['US_Open']
320             # Add the predicted champion to the database for the given year
321             cursor = mysql.connection.cursor()
322             cursor.execute(
323                 "INSERT INTO champions (Year, Australian_Open, Roland_Garros, Wimbledon, US_Open) VALUES (%s, %s, %s, %s, %s)"
324                 (year, australian_open, roland_garros, wimbledon, us_open))

```

2.5.3 PUT Requests

PUT /champions/<int:year>

A login is required for this request. The user will be able to update his predictions for the following year. The user cannot update any champions that he did not create. Parameters are passed in the body request.

```

t > app.py > ChampionsPrediction > post
# Update champion information for a specific year
@api.route('/champions/<int:year>')
class UpdateChampion(Resource):
    # @api.doc(parser=token_parser)
    @api.header('Authorization', 'Bearer token')
    @api.expect(champions_update_model) # Expecting the request payload to match the defined model
    @api.doc(params={'year': 'Year to update champion information'}, parser=token_parser) # Description
    def put(self, year):
        try:
            data = request.json

            # Ensure at least one field is provided for updating
            if not any(field in data for field in ['Australian_Open', 'Roland_Garros', 'Wimbledon', 'US_Open']):
                return {'error': 'No fields provided for update'}, 400

            cursor = mysql.connection.cursor()

            # Build the update query dynamically based on the provided fields
            update_query = "UPDATE champions SET "
            update_values = []

            for field, value in data.items():
                if field in ['Australian_Open', 'Roland_Garros', 'Wimbledon', 'US_Open']:
                    update_query += f"{field} = %s, "

```

2.5.4 DELETE Requests

DELETE /champions/<int:year>

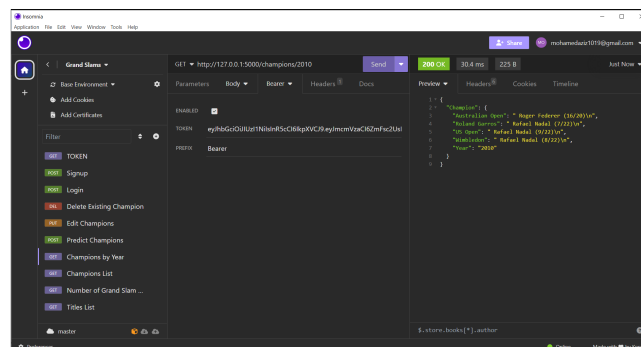
A login is required. The user will be able to delete only the champions that predicted by passing only the year of prediction.

```
apiproject > app.py > ChampionsPrediction > post
371     except Exception as e:
372         print(f"Error in update_champion route: {str(e)}")
373         return {'error': 'Internal Server Error', 'message': str(e)}, 500
374
375
376 #DELETE Champions
377 @api.route('/champions/<int:year>')
378 class DeleteChampion(Resource):
379     @api.doc(parser=token_parser)
380     @api.header('Authorization', 'Bearer token')
381     @api.doc(params={'year': 'Year to delete champion'}) # Description for the Swagger documentation
382     def delete(self, year):
383         try:
384             cursor = mysql.connection.cursor()
385             cursor.execute("DELETE FROM champions WHERE Year = %s", (year,))
386             mysql.connection.commit()
387             cursor.close()
388
389             return {'message': f'Champion for the year {year} deleted successfully'}, 200
390
391         except Exception as e:
392             print(f"Error in delete_champion route: {str(e)}")
393             return {'error': 'Internal Server Error', 'message': str(e)}, 500
394
```

2.6 Insomnia Contribution

Insomnia is an application used for API testing. It is an HTTP client that tests HTTP requests, utilizing a graphical user interface, through which we obtain different types of responses that need to be subsequently validated. [?]

I have used Insomnia for the automatic testing of my API requests, as well as some snippets such as "Response Time less than 200ms", "Status Code is 200"..etc.



2.7 Streamlit Contribution

Streamlit is an open-source Python library that is designed for creating web applications for data science and machine learning. It allows developers and data scientists to turn data scripts into shareable web applications quickly and easily. Streamlit is particularly popular for its simplicity and ease of use, making it accessible to a broad audience. [?]

I was able to create an dynamic interface through Streamlit UI because it automatically updates as users interact with the application, providing a responsive and dynamic user experience.

```
395 #Add a route to serve the Streamlit interface
396 @app.route('/streamlit_interface', methods=['GET'])
397 @jwt_required()
398 def serve_streamlit_interface():
399     # Add your Streamlit interface URL here
400     return redirect("http://localhost:8501/") # Upd
401
```

2.8 Swagger Contribution

Swagger is a toolset that helps developers design, build, document, and consume RESTful web services. It includes a specification format (OpenAPI Specification) and a set of tools, including Swagger UI and Swagger Editor, to interact with and visualize APIs. In the context of a Flask API, using Swagger with Flask involves integrating Swagger documentation into your Flask application. This integration allows you to automatically generate an interactive documentation interface for your API, making it easier for developers to understand and test your API endpoints.. [5]

Tennis API ^{1.0}

[Base URL: /]

[/swagger.json](#)

Tennis Grand Slams Champions

Endpoints Default namespace

POST /champions

DELETE /champions/{year}

POST /login

POST /signup

GET /titles List of Players with 3 titles or more

GET /titles/{player_name} Total Grand Slams Titles for each player

Models

2.9 Database Structure

Database is scraped from 2 different tables in a web page: List of Grand Slam men's singles champions .

The database consists of 2 tables which no relations between them.

2.9.1 Tables

Table "titles"

containing 9 columns :

1. titles : Total number of titles.
2. player : Player name.
3. amateur era : number of titles won in the amateur era.
4. open era : number of titles won in the open era.
5. australian open : Australian Open titles won.
6. roland garros : Roland Garros titles won.
7. wimbledon : Wimbledon titles won.
8. us open : US Open titles won.
9. years : Year range from the 1st to the last title won.

Table "champions"

containing 5 columns :

1. Year : ATP Year
2. Australian Open : Winner of AO
3. Roland Garros: Winner of RG
4. Wimbledon: Winner of Wimbledon
5. US Open: Winner of USOpen

Server: 127.0.0.1 » Database: grand_slams » Table: titles										
Browse	Structure	SQL	Search	Insert	Export	Import	Privileges	Operations	Tracking	Triggers
+ Options										
Titles ▼ 1	Player	Amateur_Era	Open_Era	Australian_Open	Roland_Garros	Wimbledon	US_Open	Years		
<input type="checkbox"/> Edit Copy Delete 8	Fred Perry	8	N/A	1	1	3	3	1933–1936		
<input type="checkbox"/> Edit Copy Delete 8	Ivan Lendl	N/A	8	2	3	0	3	1984–1990		
<input type="checkbox"/> Edit Copy Delete 8	Andre Agassi	N/A	8	4	1	1	2	1992–2003		
<input type="checkbox"/> Edit Copy Delete 8	Ken Rosewall	4	4	4	2	0	2	1953–1972		
<input type="checkbox"/> Edit Copy Delete 8	Jimmy Connors	N/A	8	1	0	2	5	1974–1983		
<input type="checkbox"/> Edit Copy Delete 7	Henri Cochet	7	N/A	0	4	2	1	1926–1932		
<input type="checkbox"/> Edit Copy Delete 7	John McEnroe	N/A	7	0	0	3	4	1979–1984		
<input type="checkbox"/> Edit Copy Delete 7	John Newcombe	2	5	2	0	3	2	1967–1975		
<input type="checkbox"/> Edit Copy Delete 7	Mats Wilander	N/A	7	3	3	0	1	1982–1988		
<input type="checkbox"/> Edit Copy Delete 7	René Lacoste	7	N/A	0	3	2	2	1925–1929		
<input type="checkbox"/> Edit Copy Delete 7	Richard Sears	7	N/A	0	0	0	7	1881–1887		
<input type="checkbox"/> Edit Copy Delete 7	William Larned	7	N/A	0	0	0	7	1901–1911		
<input type="checkbox"/> Edit Copy Delete 7	William Renshaw	7	N/A	0	0	7	0	1881–1889		
<input type="checkbox"/> Edit Copy Delete 6	Don Budge	6	N/A	1	1	2	2	1937–1938		
<input type="checkbox"/> Edit Copy Delete 6	Boris Becker	N/A	6	2	0	3	1	1985–1996		
<input type="checkbox"/> Edit Copy Delete 6	Jack Crawford	6	N/A	4	1	1	0	1931–1935		
<input type="checkbox"/> Edit Copy Delete 6	Stefan Edberg	N/A	6	2	0	2	2	1985–1992		
<input type="checkbox"/> Edit Copy Delete 6	Anthony Wilding	6	N/A	2	0	4	0	1906–1913		
<input type="checkbox"/> Edit Copy Delete 6	Laurence Doherty	6	N/A	0	0	5	1	1902–1906		
<input type="checkbox"/> Edit Copy Delete 5	Tony Trabert	5	N/A	0	2	1	2	1953–1955		
<input type="checkbox"/> Edit Copy Delete 5	Frank Sedgman	5	N/A	2	0	1	2	1949–1952		
<input type="checkbox"/> Console Edit Copy Delete 4	Lew Hoad	4	N/A	1	1	2	0	1956–1957		

Server: 127.0.0.1 » Database: grand_slams » Table: champions												
Browse		Structure		SQL	Search	Insert	Export	Import	Privileges	Operations	Tracking	
+ Options												
▼ Year ▼ 1		Australian_Open		Roland_Garros		Wimbledon		US_Open				
<input type="checkbox"/>	Edit	Copy	Delete	2023	Novak Djokovic (22/24)	Novak Djokovic (23/24)	Carlos Alcaraz (2/2)	Novak Djokovic (24/24)				
<input type="checkbox"/>	Edit	Copy	Delete	2022	Rafael Nadal (21/22)	Rafael Nadal (22/22)	Novak Djokovic (21/24)	Carlos Alcaraz (1/2)				
<input type="checkbox"/>	Edit	Copy	Delete	2021	Novak Djokovic (18/24)	Novak Djokovic (19/24)	Novak Djokovic (20/24)	Daniil Medvedev (1/1)				
<input type="checkbox"/>	Edit	Copy	Delete	2020	Novak Djokovic (17/24)	Rafael Nadal (20/22) [m]	cancelled (COVID-19 pandemic)	Dominic Thiem (1/1)				
<input type="checkbox"/>	Edit	Copy	Delete	2019	Novak Djokovic (15/24)	Rafael Nadal (18/22)	Novak Djokovic (16/24)	Rafael Nadal (19/22)				
<input type="checkbox"/>	Edit	Copy	Delete	2018	Roger Federer (20/20)	Rafael Nadal (17/22)	Novak Djokovic (13/24)	Novak Djokovic (14/24)				
<input type="checkbox"/>	Edit	Copy	Delete	2017	Roger Federer (18/20)	Rafael Nadal (15/22)	Roger Federer (19/20)	Rafael Nadal (16/22)				
<input type="checkbox"/>	Edit	Copy	Delete	2016	Novak Djokovic (11/24)	Novak Djokovic (12/24)	Andy Murray (3/3)	Stan Wawrinka (3/3)				
<input type="checkbox"/>	Edit	Copy	Delete	2015	Novak Djokovic (8/24)	Stan Wawrinka (2/3)	Novak Djokovic (9/24)	Novak Djokovic (10/24)				
<input type="checkbox"/>	Edit	Copy	Delete	2014	Stan Wawrinka (1/3)	Rafael Nadal (14/22)	Novak Djokovic (7/24)	Marin Čilić (1/1)				
<input type="checkbox"/>	Edit	Copy	Delete	2013	Novak Djokovic (6/24)	Rafael Nadal (12/22)	Andy Murray (2/3)	Rafael Nadal (13/22)				
<input type="checkbox"/>	Edit	Copy	Delete	2012	Novak Djokovic (5/24)	Rafael Nadal (11/22)	Roger Federer (17/20)	Andy Murray (1/3)				
<input type="checkbox"/>	Edit	Copy	Delete	2011	Novak Djokovic (2/24)	Rafael Nadal (10/22)	Novak Djokovic (3/24)	Novak Djokovic (4/24)				
<input type="checkbox"/>	Edit	Copy	Delete	2010	Roger Federer (16/20)	Rafael Nadal (7/22)	Rafael Nadal (8/22)	Rafael Nadal (9/22)				
<input type="checkbox"/>	Edit	Copy	Delete	2009	Rafael Nadal (6/22)	Roger Federer (14/20)	Roger Federer (15/20)	Juan Martín del Potro (1/1)				
<input type="checkbox"/>	Edit	Copy	Delete	2008	Novak Djokovic (1/24)	Rafael Nadal (4/22)	Rafael Nadal (5/22)	Roger Federer (13/20)				
<input type="checkbox"/>	Edit	Copy	Delete	2007	Roger Federer (10/20)	Rafael Nadal (3/22)	Roger Federer (11/20)	Roger Federer (12/20)				
<input type="checkbox"/>	Edit	Copy	Delete	2006	Roger Federer (7/20)	Rafael Nadal (2/22)	Roger Federer (8/20)	Roger Federer (9/20)				
<input type="checkbox"/>	Edit	Copy	Delete	2005	Marat Safin (2/2)	Rafael Nadal (1/22)	Roger Federer (5/20)	Roger Federer (6/20)				
<input type="checkbox"/>	Edit	Copy	Delete	2004	Roger Federer (2/20)	Gastón Gaudio (1/1)	Roger Federer (3/20)	Roger Federer (4/20)				
<input type="checkbox"/>	Edit	Copy	Delete	2003	Andre Agassi (8/8)	Juan Carlos Ferrero (1/1)	Roger Federer (1/20)	Andy Roddick (1/1)				
<input type="checkbox"/>	Edit	Copy	Delete	2002	Thomas Johansson (1/1)	Albert Costa (1/1)	Lleyton Hewitt (2/2)	Pete Sampras (14/14)				
Console	Edit	Copy	Delete	2001	Andre Agassi (7/8)	Gustavo Kuerten (3/3)	Goran Ivanišević (1/1)	Lleyton Hewitt (1/2)				

Chapter 3

Conclusion

In conclusion, the Tennis Grand Slams API developed in this project offers users a comprehensive platform for exploring the rich history of champions in major tennis tournaments. The application not only provides historical data but also empowers users to actively engage by submitting, updating, or deleting their predictions. The dynamic and user-friendly interface enhances the overall experience, setting it apart from conventional sports apps such as FlashScore. I had fun working on this project because I was working on it while simultaneously watching the Australian Open games. Although, I faced more problems than I imagined, each successful compilation gave me a dose of dopamine and I discovered a lot of new tricks/life hacks that could help me in the future in my professional career.

After long days and sleepless nights I finally managed to finish this project and I realized that APIs are fundamentals to know and that they facilitate a lot of work. Thank you Dr.Montassar for helping us in our learning journey.

Mohamed Aziz Ajmi



Bibliography

- [1] L. Richardson, “Beautiful soup documentation,” *Dosegljivo*: [https://www. crummy. com/-software/BeautifulSoup/bs4/doc/](https://www.crummy.com/software/BeautifulSoup/bs4/doc/). [Dostopano: 7. 7. 2018], 2007.
- [2] D. F. Ningtyas and N. Setiyawati, “Implementasi flask framework pada pembangunan aplikasi purchasing approval request,” *Jurnal Janitra Informatika Dan Sistem Informasi*, vol. 1, no. 1, pp. 19–34, 2021.
- [3] T. T. Tidwell, “Wilhelm schlenk: the man behind the flask,” *Angewandte Chemie International Edition*, vol. 40, no. 2, pp. 331–337, 2001.
- [4] W. Python, “Python,” *Python Releases for Windows*, vol. 24, 2021.
- [5] M. Grinberg, “Flask-socketio documentation,” *línea*. Disponible en: [https://flask-socketio.readthedocs. org/en/latest/](https://flask-socketio.readthedocs.org/en/latest/). [Último acceso: 2015], 2019.