

University of Aveiro

Web Semantic

HotelStats

Katarina Gacina, Adrian Murillo Moreno, Mia Krsticevic

Aveiro, March 2025.

Contents

1. Introduction	3
2. Data, sources, and transformation process	3
Data Transformation to RDF	4
3. Operations on the data (SPARQL queries)	4
4. Application functionalities (UI)	6
4.1. Dashboard	6
4.2. Manage Reservations	7
5. Conclusions	9
6. Configuration required to run the application	10

1. Introduction

HotelStats is an online tool created to handle, assess, and present hotel booking information using modern semantic technologies.

The initiative utilizes a real dataset that gathers reservations from two kinds of hotels: a city hotel and a resort hotel and changes the unprocessed information into an organized RDF format. It employs the schema.org vocabulary along with specific properties, providing clarity and adaptability in semantics.

The RDF information is stored in a **GraphDB triple store**, allowing users to apply **SPARQL** to request, combine, and extract valuable insights from the data. These insights, for example, include:

- List of reservations in each country
- Distribution of hotel meal types
- Reservations per year grouped by month
- Average stay duration per year grouped by month

The system additionally enables **interactive reservation management**, letting admin users add, change, view or remove bookings through a web interface.

A **Django-based dashboard** shows previously mentioned statistics and graphs, with interactive visuals.

This project shows how semantic technologies can improve conventional data processes, making hotel booking information easier to access, search, analyze and share.

2. Data, sources, and transformation process

The information used for this project was obtained from Kaggle, specifically from the following dataset: [Hotel Booking](https://www.kaggle.com/datasets/alexisbisaia/hotel-booking). This dataset provides extensive information about bookings from two types of hotels: a City Hotel and a Resort Hotel. It includes over 119,000 entries and has 36 features for every booking, like the date of arrival, number of guests, meal choices, country of origin, booking status, average daily rate (ADR), room type, and the channels through which the booking was made.

Data Transformation to RDF

To facilitate semantic searching and integration, the CSV file was converted into RDF (Resource Description Framework) format by employing a Python script. This script carries out several important functions:

1. Parsing and Sanitization:

- The script reads the CSV and cleans values (e.g., removing invalid characters, trimming whitespace).
- Null or missing values are safely ignored to prevent malformed triples.

2. Triple Generation:

- Each reservation is represented as a subject with a unique URI:
<http://example.org/booking/{ID}>
- Attributes are mapped to either standard vocabulary terms from schema.org or to custom predicates under the namespace <http://example.org/>.

3. Datatype Annotation:

- Literals are annotated with the appropriate XSD datatypes, such as [xsd:boolean](http://www.w3.org/2001/XMLSchema#boolean), [xsd:integer](http://www.w3.org/2001/XMLSchema#integer), [xsd:gYear](http://www.w3.org/2001/XMLSchema#gYear), and [xsd:date](http://www.w3.org/2001/XMLSchema#date), ensuring correctness and interpretability.

4. Export Format:

- The final RDF graph is exported in **N-Triples (.nt)** format, which is fully compatible with **GraphDB** and other semantic triple stores.

This transformation process lays the foundation for semantic querying via **SPARQL**, enabling meaningful analysis and dynamic interaction with the data through the system's web interface.

3. Operations on the data (SPARQL queries)

This section shows the SPARQL queries implemented for utilizing data in our web application.

Regarding admin management of the database, we implemented an insert query for adding new booking into the existing hotel booking database, as shown in Figure 1. Note that only those properties which are currently utilized for our web application usage are being added.

```
booking_uri = f"<http://example.org/booking/{id}>"

sparql = SPARQLWrapper(GRAPHDB_ENDPOINT_STAT)

query = f"""
    PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

    INSERT DATA {{
        {booking_uri} <http://schema.org/addressCountry> "{data['country']}" .
        {booking_uri} <http://schema.org/arrivalDateYear> "{data['year']}"^^xsd:gYear .
        {booking_uri} <http://schema.org/arrivalDateMonth> "{data['month']}" .
        {booking_uri} <http://example.org/arrivalDay> "{data['day']}"^^xsd:integer .
        {booking_uri} <http://example.org/weekNights> "{data['weekNights']}"^^xsd:integer .
        {booking_uri} <http://example.org/weekendNights> "{data['weekendNights']}"^^xsd:integer .
        {booking_uri} <http://schema.org/hotel> "{data['hotelType']}" .
        {booking_uri} <http://example.org/isCanceled> "0"^^xsd:boolean .
        {booking_uri} <http://schema.org/meal> "{data['mealType']}" .
    }}
"""
```

Figure 1: INSERT DATA query

Furthermore, the query in Figure 2 is used for retrieving the highest booking id in the database. Booking id is the last part of booking URI. This query is used before inserting new data so that added booking has the new, highest one booking id, in order to prevent multiple bookings for having the same id number and therefore, it helps in preserving data integrity.

```
query_for_id = """PREFIX schema: <http://schema.org/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?booking (xsd:integer(SUBSTR(STR(?booking), STRLEN(STR(?booking)) - STRLEN(REPLACE(STR(?booking), ".*"/, "")) + 1)) AS ?bookingNumber)
WHERE {
    ?booking schema:hotel ?h .
}
ORDER BY DESC(?bookingNumber)
LIMIT 1
"""
```

Figure 2: SELECT query for retrieving highest booking id

Admin users also have an option to review existing booking by providing its id. The query for getting properties of specific booking is shown in Figure 3.

```
query = f"""
    SELECT ?p ?o WHERE {{ <http://example.org/booking/{str(id)}> ?p ?o . }}
"""
```

Figure 3: SELECT query for retrieving booking properties

If considered non useful, there is possibility for deletion of certain booking, also by providing its id. The delete query from Figure 4 is used for this purpose.

```
query = f"""
    DELETE WHERE {{ <http://example.org/booking/{str(id)}> ?p ?o . }}
"""
```

Figure 4: DELETE query

Moreover, if booking undergoes changes or has mistakes, admin users can update it and this functionality is possible due to query in Figure 5. This query is dynamically created since one can choose to update only one or more properties of specific booking.

```
uri = f"<http://example.org/booking/{data['id']}>"

PREFIXES = """PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>"""

delete_statements = []
insert_statements = []
where_statements = []

print(data["mealType"])

field_mappings = {
    "year": ("<http://schema.org/arrivalDateYear>", '^xsd:gYear'),
    "month": ("<http://schema.org/arrivalDateMonth>", ''),
    "day": ("<http://example.org/arrivalDay>", '^xsd:integer'),
    "is_canceled": ("<http://example.org/isCanceled>", '^xsd:boolean'),
    "mealType": ("<http://schema.org/meal>", ''),
    "weekendNights": ("<http://example.org/weekendNights>", '^xsd:integer'),
    "weekNights": ("<http://example.org/weekNights>", '^xsd:integer')
}

for key, (rdf_property, datatype) in field_mappings.items():
    value = data.get(key)

    if value:
        delete_statements.append(f"{uri} {rdf_property} ?{key} .")
        insert_statements.append(f'{uri} {rdf_property} "{value}" {datatype} .')
        where_statements.append(f"OPTIONAL {{ {uri} {rdf_property} ?{key} . }}"

if not delete_statements:
    return None

query = f"""
{PREFIXES}

DELETE {{
    { ' '.join(delete_statements) }
}}
INSERT {{
    { ' '.join(insert_statements) }
}}
WHERE {{
    { ' '.join(where_statements) }
}}
"""
```

Figure 5: Query for booking properties update

Also, we utilized an ASK query, shown in Figure 6, to check if booking with specified id exists before applying booking modifications or deletion.

```
sparql.setQuery(f"""
ASK {{
    <http://example.org/booking/{str(id)}> ?p ?o .
}}
""")
```

Figure 6: ASK query for checking if booking id exists in the database

Next set of queries is used for utilizing data for display of statistical information.

Query in Figure 7 retrieves hotel meal types and their count in descending order from the database, so that meal type distribution can be examined.

```
sparql.setQuery("""
    PREFIX schema: <http://schema.org/>
    SELECT ?meal (COUNT(?meal) AS ?count) WHERE {
        ?order schema:meal ?meal .
    }
    GROUP BY ?meal
    ORDER BY DESC(?count)
""")
```

Figure 7: SELECT query for retrieving meal data

Query in Figure 8 filters reservations of a certain year and retrieves them grouped by month.

```
sparql.setQuery(f"""PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
    SELECT ?year ?month (COUNT(?booking) AS ?count)
    WHERE {{
        ?booking <http://schema.org/arrivalDateMonth> ?month ;
        <http://schema.org/arrivalDateYear> ?year .

        FILTER(?year = "{year}"^^xsd:gYear)
    }}
    GROUP BY ?year ?month
    ORDER BY ?year
""")
```

Figure 8: SELECT query for retrieving reservation data filtered by year

The following query in Figure 9, also filters reservations of a certain year but retrieves average stay duration per month, calculated by taking the average sum of all week and weekend nights.

```
sparql.setQuery(f"""
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?month (AVG(?total_nights) AS ?media_estancia)
WHERE {{
    ?booking <http://schema.org/arrivalDateMonth> ?month ;
        <http://example.org/weekNights> ?w ;
        <http://example.org/weekendNights> ?we ;
        <http://schema.org/arrivalDateYear> ?year .

    BIND(xsd:integer(?w) + xsd:integer(?we) AS ?total_nights)

    FILTER(?year = "{year}"^^xsd:gYear)
}}
GROUP BY ?month
""")
```

Figure 9: SELECT query for retrieving average stay data filtered by year

And the last query in Figure 10, retrieves the count of reservations per each country, used for displaying most popular tourist destinations.

```
sparql.setQuery("""
PREFIX schema: <http://schema.org/>

SELECT ?addressCountry (COUNT(?addressCountry) AS ?count)
WHERE {
    ?order schema:addressCountry ?addressCountry .
}
GROUP BY ?addressCountry
ORDER BY DESC(?count)
""")
```

Figure 10: SELECT query to retrieve number of reservations per country

There is one additional query, shown in Figure 11, which retrieves all reservation years in order to dynamically display them in the dashboard template file.


```
sparql.setQuery("""
    PREFIX schema: <http://schema.org/>

    SELECT ?year
    WHERE {
        ?booking schema:arrivalDateYear ?year .
    }
    GROUP BY ?year
""")
```

Figure 11: SELECT query for retrieving all booking years in database

4. Application functionalities (UI)

The application provides a comprehensive hotel management system with an intuitive user interface (UI) divided into interactive dashboard and reservation management modules. The system features a responsive user interface, with Django handling backend operations, HTML providing structure, and CSS ensuring styling. Frontend interactivity is implemented through JavaScript, while Chart.js powers data visualization. SPARQL queries retrieve RDF data from the database, which Django processes in its view functions. The processed data is then dynamically rendered on the client side and displayed as various graphs on the dashboard.

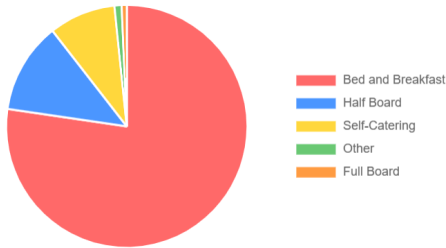
4.1. Dashboard

Upon entering the web application, users are greeted with a dashboard as the home page. This dashboard presents key hotel statistics, allowing managers and directors to gain valuable insights into the hotels' performance and areas for improvement. They can track trends and progress over the years. The dashboard maintains a clean, structured layout with a modern design, ensuring easy readability and usability.

The data is displayed in a series of cards, arranged in a 2x2 grid. On the left (Figure 12), a pie chart illustrates meal plan preferences, categorizing them into five options: Bed and Breakfast (BB), Half Board (HB), Full Board (FB), Self-Catering (SC), and Other. Users can hover over each section to see the exact number of visitors for each meal plan. Next to the pie chart, a bar chart visualizes monthly reservations for each year. A year selector dropdown enables users to filter the data by selecting a specific year. When a selection is made, the system sends a request to the backend, where a SPARQL query processes the data using filter operations. The results are then converted to JSON format and used to update the chart dynamically.

Hotel Statistics Dashboard

Meal Plans



Reservations per Month

Select Year: **2016**

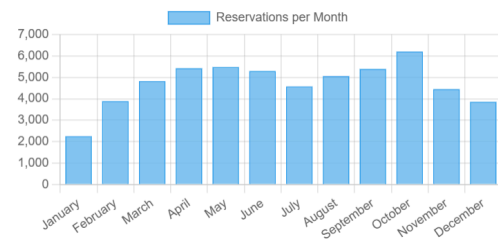


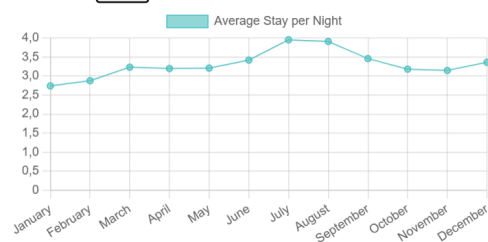
Figure 12: Dashboard - first part

Additionally, the dashboard features data on average stay duration and the top visitor countries (Figure 13). On the left, a line chart visualizes the average number of nights guests stay at the hotel. A dropdown menu allows users to select a specific year, dynamically updating the chart. The plotted data points show fluctuations in guest stay duration, with noticeable peaks during certain months.

On the right, a panel titled "Top Countries" lists the countries with the number of reservations, sorted in descending order. A search bar at the top enables users to filter the list by entering a country name. Portugal ranks first with the highest number of reservations, followed by the United Kingdom and France. The list provides valuable insights into the primary sources of guests. Below the charts, a "Manage Reservations" button is prominently displayed, accessible only to administrators for managing reservations.

Average Stay per Month

Select Year: **2016**



Top Countries

Search for a country...

1. Portugal: 48590 Reservations
2. United Kingdom: 12129 Reservations
3. France: 10416 Reservations
4. Spain: 8568 Reservations
5. Germany: 7288 Reservations
6. Italy: 3766 Reservations
7. Ireland: 3375 Reservations
8. Belgium: 2342 Reservations
9. Brazil: 2224 Reservations
10. Netherlands: 2104 Reservations

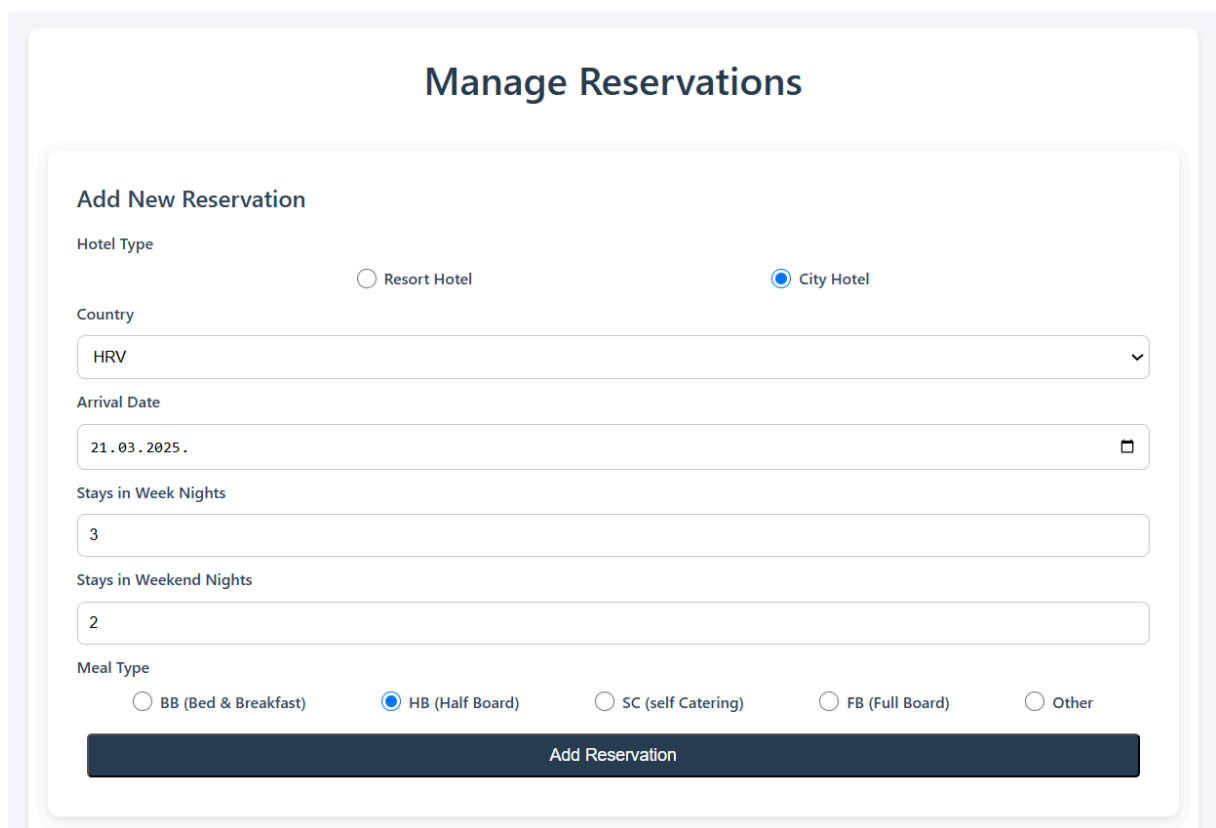
Manage Reservations

Figure 13: Dashboard - second part

If a user is an admin, they can access the Manage Reservations section. This section allows administrators to add, modify, or delete reservations. Additionally, an option to get reservation enables admins to check exact saved reservations in the dataset.

4.2. Manage Reservations

When adding a new reservation, the admin must provide details such as the hotel type (Resort Hotel or City Hotel), guest's country, arrival date, average stay duration (separately for weekday and weekend nights), and the meal type. Upon submitting the reservation, the system validates the input. If all required fields are completed, a confirmation message appears, indicating that the reservation was successfully added along with the generated reservation ID.



The screenshot displays a web interface titled "Manage Reservations". Within this interface, there is a section titled "Add New Reservation". This section contains several form fields and radio buttons for user input. The "Hotel Type" field has two radio buttons: "Resort Hotel" and "City Hotel", with "City Hotel" selected. The "Country" field is a dropdown menu showing "HRV". The "Arrival Date" field shows "21. 03. 2025." with a calendar icon. The "Stays in Week Nights" field contains the number "3". The "Stays in Weekend Nights" field contains the number "2". The "Meal Type" field has five radio buttons: "BB (Bed & Breakfast)", "HB (Half Board)" (selected), "SC (self Catering)", "FB (Full Board)", and "Other". At the bottom of the form is a dark blue button labeled "Add Reservation".

Figure 14: Manage reservations - add new reservation

The get function is used to view property values of specific booking by providing a valid ID. If ID exists, the values are retrieved, otherwise a message shows, informing the admin that the provided ID does not exist.

The screenshot displays a web interface for retrieving reservation information. At the top, the title 'Get Reservation' is shown. Below it, a label 'Reservation ID' is positioned above a text input field containing the value '126'. A dark blue button labeled 'Get Reservation' is located below the input field. Underneath the button, the section 'Reservation Details' is displayed, listing the following information: Reservation ID: 126, Hotel Type: Resort Hotel, Country: PRT, Arrival Date: 2015-July-4, Stays in Week Nights: 1, Stays in Weekend Nights: 0, Meal Type: FB, and Cancelation status: 0.

Reservation Details	
Reservation ID:	126
Hotel Type:	Resort Hotel
Country:	PRT
Arrival Date:	2015-July-4
Stays in Week Nights:	1
Stays in Weekend Nights:	0
Meal Type:	FB
Cancelation status:	0

Figure 15: Manage reservations - get reservation

Additionally, the modify function allows the admin to update an existing reservation by entering the correct reservation ID. If the ID is verified, the admin can modify details such as the arrival date, length of stay (weekday and weekend nights), meal plan, and whether the reservation has been canceled or not.

The delete function requires only the reservation ID. Once the admin enters a valid ID, the system removes the corresponding reservation from the dataset. Get function lists all of the details about reservation, as displayed in image.

Modify Reservation

Reservation ID

Arrival Date

Stays in Week Nights

Stays in Weekend Nights

Is the reservation canceled?

☐ Yes ☐ No

Meal Type

☐ BB (Bed & Breakfast) ☐ HB (Half Board) ☐ SC (self Catering) ☐ FB (Full Board) ☐ Other

Modify Reservation

Delete Reservation

Reservation ID

Delete Reservation

Back to Dashboard

Figure 16: Manage reservations - modify and delete reservation

5. Conclusions

HotelStats is a strong foundation for analyzing hotel reservation data, offering a detailed and interactive user interface for visualizing key statistics. While the current version provides an excellent starting point with useful features such as reservation management and month-based histograms, there is room for further enhancement. The addition of more advanced graphing capabilities and interactive data visualizations would significantly improve the insights provided.

From a technical standpoint, the use of RDF (Resource Description Framework) and GraphDB as the triplestore solution proves to be an effective approach for structuring and querying complex data. RDF allows for flexible, semantic data modeling that makes it easy to integrate data from various sources. GraphDB's powerful support for SPARQL queries enables dynamic and efficient data retrieval, allowing for real-time analytics.

However, there are challenges that come with this setup. RDF's flexibility can also lead to difficulties in maintaining consistency and integrity of the data. Despite these challenges, the potential for growth and optimization remains vast, and with continued development, HotelStats can evolve into a more powerful tool for comprehensive hotel data analysis.

6. Configuration required to run the application

Requirements: Python 3.10 or higher, pip, GraphDB, git

First step: Downloading git repository

Git repository is available on: <https://github.com/KrsticevicM/hotel-stats>.

User needs to clone the repository locally: `git clone`
<https://github.com/KrsticevicM/hotel-stats.git>

Second step: Installing dependencies

It is advisable for user to create a separate anaconda environment. The project dependencies are provided in `environment.yml` file. User can position inside `hotel-stats` folder and create project environment by running the command:

```
conda env create -f environment.yml
```

Third step: Dataset

User needs to position inside `hotel-stats/hotel_system/hotel_database_script` and run script for converting `.csv` dataset file to `.nt` dataset file:

```
python convert_hotels_csv_to_rdf.py
```

Fourth step: Database

Free version of GraphDB by ontotext, provided at the following link, was used: <https://graphdb.ontotext.com/>. After the user downloads and starts the database, it is running on <http://localhost:7200/>. User needs to create a repository named `hotels` and import the previously created `hotel_bookings.nt` file inside of it.

Fifth step: Run the application

User should position inside the `hotel-stats/hotel_systems` folder.

If wanting to create admin, user should run:

```
python manage.py createsuperuser
```

On the first time of application running, it is advisable to run:

```
python manage.py migrate
```

The application starts by running command:

```
python manage.py runserver
```

and can be accessed in a web browser on <http://localhost:8000/>.