

Case Study Raporu

Projede standart MVC yapısını kurdum. DB işlerini Repository'ye, business logic Service'e, http isteklerini Controller'a bıraktım. Veri tabanı işlemleri için Spring Data JPA kullandım, böylece uzun uzun SQL yazmak yerine findByName gibi methodları kullandım.

Güvenlik ve düzen için dışarıdan gelen verileri direkt Entity sınıflarına almak yerine DTO pattern'ini kullandım. Özellikle bilet alma, buyTicket, kısmında hem bilet oluşturup hem de koltuk sayısını azalttığım için, işlem yarıy kalırsa veriler bozulmasın diye Transactional annotation ekledim, bu sayede bir hata olursa yapılan her şey geri alınıyor.

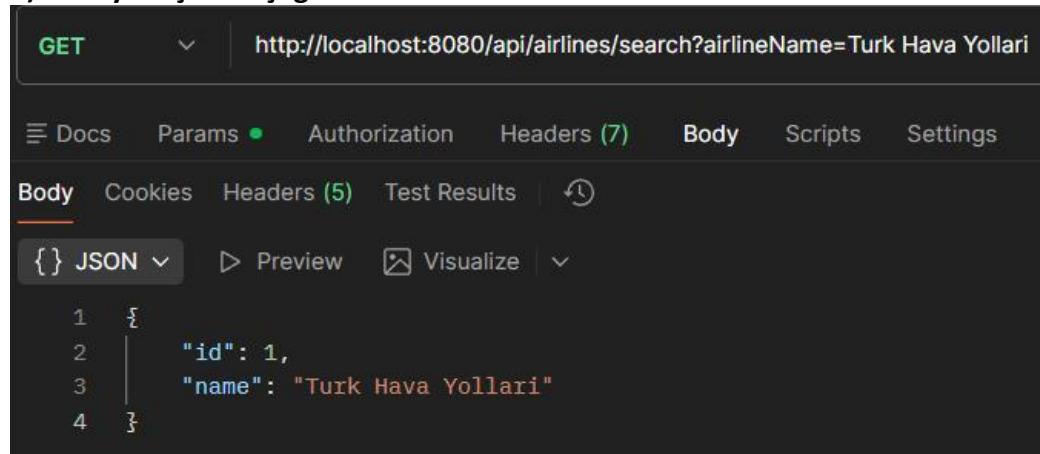
Test tarafında ise işi ikiye ayırdım. Controller testleri için Web MVC Test kullandım çünkü sunucuyu komple kaldırmadan sadece o endpoint'in çalışıp çalışmadığını (200 dönüşme) görmek istedim. Servis katmanında ise Mockito ile Repository'leri taklit ettim (mocklamak). Gerçek veritabanına gitmeden, sadece yazdığım fiyat hesaplama ve maskeleme mantığının doğru çalışıp çalışmadığını test edebildim.

İstenen tüm özelliklerini kapsayacak şekilde backend geliştirmelerini tamamladım. Kodun üzerinden geçmek veya yaklaşımımı detaylandırmak için müsaithisiniz olduğunda görüşme de yapabiliriz. Sorularınız olursa mail yoluyla da iletişime geçebilirsiniz.

Postman İstekleri

GET METHODLARI:

1) Havayolu şirketi çağrıma:



The screenshot shows a Postman request configuration. The method is set to GET, and the URL is `http://localhost:8080/api/airlines/search?airlineName=Turk Hava Yollari`. The Headers section contains 7 items. The Body section is set to JSON and displays the following response:

```
1 {  
2   "id": 1,  
3   "name": "Turk Hava Yollari"  
4 }
```

GET http://localhost:8080/api/airlines/all

Docs Params Authorization Headers (7) Body

Body Cookies Headers (5) Test Results ⏱

{ } JSON ▾ ▷ Preview Visualize ▾

```
1 [  
2 {  
3   "id": 1,  
4   "name": "Türk Hava Yolları"  
5 },  
6 {  
7   "id": 2,  
8   "name": "Pegasus"  
9 }  
10 ]
```

2) Havaalanı çağrıma

GET http://localhost:8080/api/airports/search/code?airportCode=ESB

Docs Params Authorization Headers (9) Body Scripts Settings

Body Cookies Headers (5) Test Results ⏱

{ } JSON ▾ ▷ Preview Visualize ▾

```
1 {  
2   "id": 2,  
3   "name": "Ankara Esenboga",  
4   "code": "ESB"  
5 }
```

GET http://localhost:8080/api/airports/search/name?airportName=İstanbul Airport

Docs Params Authorization Headers (9) Body Scripts Settings

Body Cookies Headers (5) Test Results ⏱

{ } JSON ▾ ▷ Preview Visualize ▾

```
1 {  
2   "id": 1,  
3   "name": "İstanbul Airport",  
4   "code": "IST"  
5 }
```

GET <http://localhost:8080/api/airports/all>

Docs Params Authorization Headers (9) Body

Body Cookies Headers (5) Test Results ⏱

{ } JSON ▾ ▶ Preview 🖥 Visualize ▾

```
1 [  
2 {  
3   "id": 1,  
4   "name": "Istanbul Airport",  
5   "code": "IST"  
6 },  
7 {  
8   "id": 2,  
9   "name": "Ankara Esenboga",  
10  "code": "ESB"  
11 }  
12 ]
```

3) Rota Çağırma

GET <http://localhost:8080/api/routes/search/name?dep=Istanbul%20Airport&arr=Ankara%20Esenboga>

Docs Params ● Authorization Headers (9) Body ● Scripts Settings

Body Cookies Headers (5) Test Results ⏱

{ } JSON ▾ ▶ Preview 🖥 Visualize ▾

```
1 [  
2 {  
3   "id": 1,  
4   "departureAirport": {  
5     "id": 1,  
6     "name": "Istanbul Airport",  
7     "code": "IST"  
8   },  
9   "arrivalAirport": {  
10    "id": 2,  
11    "name": "Ankara Esenboga",  
12    "code": "ESB"  
13  }  
14 }  
15 ]
```

The screenshot shows a Postman test result for a GET request to `http://localhost:8080/api/routes/search/code?dep=IST&arr=ESB`. The response body is a JSON array containing one element, representing a route from Istanbul Airport (IST) to Ankara Esenboga (ESB). The JSON structure is as follows:

```
1 [  
2 {  
3   "id": 1,  
4   "departureAirport": {  
5     "id": 1,  
6     "name": "Istanbul Airport",  
7     "code": "IST"  
8   },  
9   "arrivalAirport": {  
10    "id": 2,  
11    "name": "Ankara Esenboga",  
12    "code": "ESB"  
13  }  
14}  
15]
```

4) Uçuş Çağırma

Aynı uçuşu farklı şekillerde çağrılabiliriz

The screenshot shows a Postman interface with two GET requests. The first request is for departure flights from Istanbul Airport (IST), and the second request is for arrival flights to Ankara Esenboga (ESB).

- Request 1: GET `http://localhost:8080/api/flights/search/departure?airportName=Istanbul Airport`
- Request 2: GET `http://localhost:8080/api/flights/search/arrival?airportName=Ankara Esenboga`

GET http://localhost:8080/api/flights/search/airline-name?airlineName=Turk Hava Yollari

Docs Params Authorization Headers (9) Body Scripts Settings

Body Cookies Headers (5) Test Results ⏱

{ } JSON ▾ ▷ Preview ⌂ Visualize ▾

```
1 [  
2   {  
3     "id": 1,  
4     "airline": {  
5       "id": 1,  
6       "name": "Turk Hava Yollari"  
7     },  
8     "route": {  
9       "id": 1,  
10      "departureAirport": {  
11        "id": 1,  
12        "name": "Istanbul Airport",  
13        "code": "IST"  
14      },  
15      "arrivalAirport": {  
16        "id": 2,  
17        "name": "Ankara Esenboga",  
18        "code": "ESB"  
19      },  
20    },  
21    "basePrice": 1000.0,  
22    "seat": 10,  
23    "soldSeats": 3  
24  }  
25 ]
```

5) Bilet Çağırma

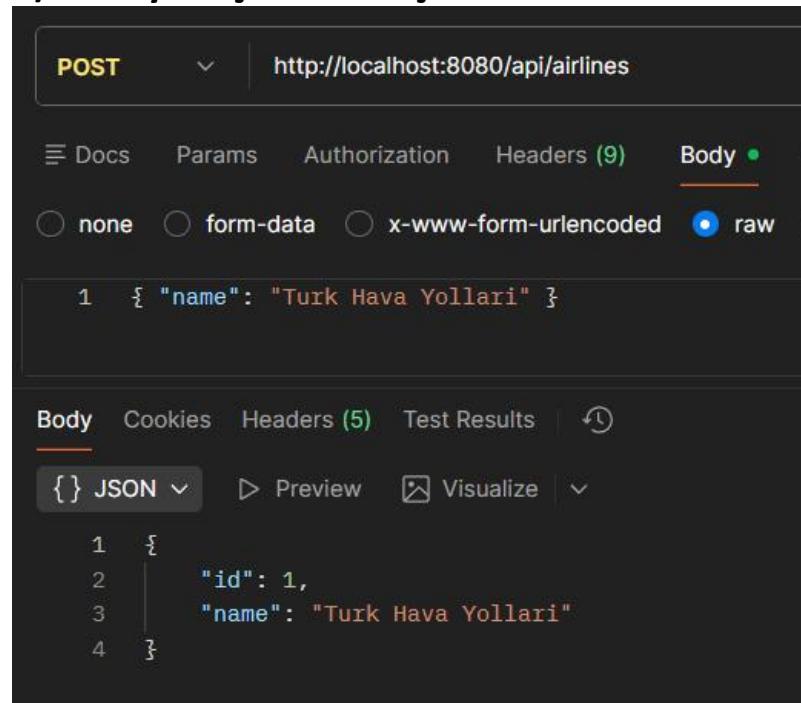
The screenshot shows a Postman interface with the following details:

- Method:** GET
- URL:** <http://localhost:8080/api/tickets/52000acb-d1e3-42fc-b7a2-9ef805e7233b>
- Headers:** (7)
- Body:** (selected)
- Content Type:** none
- Preview:** This request does not have a preview.
- Body Content:** JSON (shown in code block)
- Code Block:** (Numbered lines 8-30 showing a JSON object representing a flight route and ticket details.)

```
8     "name": "Turk Hava Yollari"
9   },
10  "route": {
11    "id": 1,
12    "departureAirport": {
13      "id": 2,
14      "name": "Istanbul Airport",
15      "code": "IST"
16    },
17    "arrivalAirport": {
18      "id": 1,
19      "name": "Ankara Esenboga",
20      "code": "ESB"
21    }
22  },
23  "basePrice": 1000.0,
24  "seat": 10,
25  "soldSeats": 3
26},
27  "creditCardMasked": "123456*****3456",
28  "pricePaid": 1200.0,
29  "canceled": false
30}
```

POST METHODLARI:

1) Havayolu şirketi Oluşturma:



POST http://localhost:8080/api/airlines

Body (9)

raw

```
1 { "name": "Türk Hava Yolları" }
```

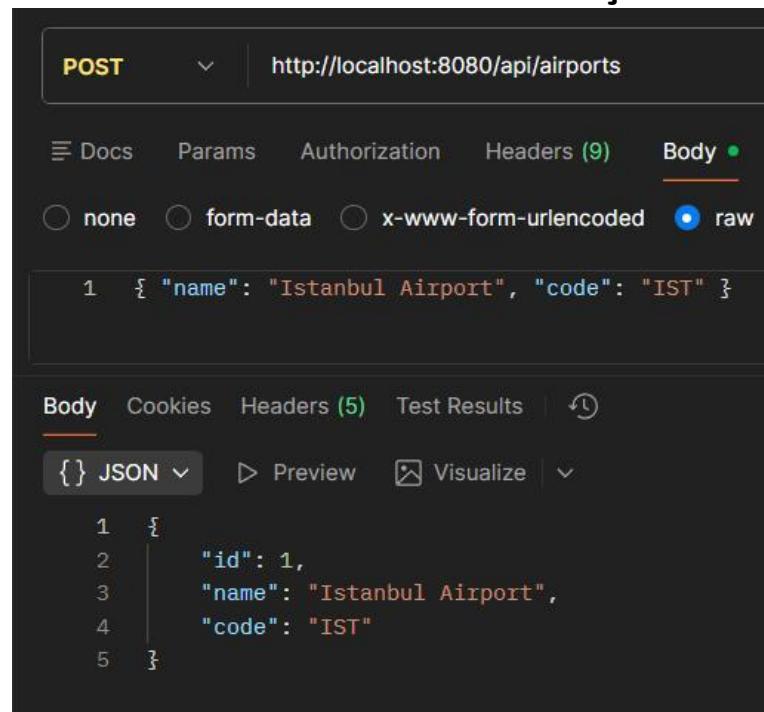
Body Cookies Headers (5) Test Results

{ } JSON ▾ Preview Visualize

```
1 {
2   "id": 1,
3   "name": "Türk Hava Yolları"
4 }
```

2) Havaalanı Oluşturma:

IST ve ESB olmak üzere 2 tane oluşturdum



POST http://localhost:8080/api/airports

Body (9)

raw

```
1 { "name": "Istanbul Airport", "code": "IST" }
```

Body Cookies Headers (5) Test Results

{ } JSON ▾ Preview Visualize

```
1 {
2   "id": 1,
3   "name": "Istanbul Airport",
4   "code": "IST"
5 }
```

3) Rota Oluşturma:

The screenshot shows a POST request to `http://localhost:8080/api/routes`. The request body is set to `raw` and contains the following JSON:

```
1 {  
2   "departureAirportCode": "IST",  
3   "arrivalAirportCode": "ESB"  
4 }
```

The response body shows the created route object:

```
1 {  
2   "id": 1,  
3   "departureAirport": {  
4     "id": 1,  
5     "name": "Istanbul Airport",  
6     "code": "IST"  
7   },  
8   "arrivalAirport": {  
9     "id": 2,  
10    "name": "Ankara Esenboga",  
11    "code": "ESB"  
12  }  
13 }
```

4) Uçuş Tanımlama:

The screenshot shows a POST request to `http://localhost:8080/api/flights`. The request body is a JSON object representing a flight route. The JSON structure is as follows:

```
1 {  
2   "airlineName": "Türk Hava Yolları",  
3   "routeId": 1,  
4   "basePrice": 1000.0,  
5   "seat": 10  
6 }  
7 },  
8   "route": {  
9     "id": 1,  
10    "departureAirport": {  
11      "id": 2,  
12      "name": "Istanbul Airport",  
13      "code": "IST"  
14    },  
15    "arrivalAirport": {  
16      "id": 1,  
17      "name": "Ankara Esenboga",  
18      "code": "ESB"  
19    },  
20    "basePrice": 1000.0,  
21    "seat": 10,  
22    "soldSeats": 0  
23 }  
24 }
```

5) Bilet Satın Alma:

The screenshot shows the Postman application interface. The top bar indicates a **POST** method and the URL **http://localhost:8080/api/tickets/buy?flightId=1&creditCard=1234-5678 9012-3456**. Below the URL, there are tabs for **Docs**, **Params**, **Authorization**, **Headers (9)**, **Body**, **Cookies**, **Headers (5)**, **Test Results**, and **Settings**. The **Body** tab is selected and expanded to show a JSON payload. The JSON content is as follows:

```
3   "ticketNumber": "bc1636d8-0574-4706-b6db-69b3eec4750d",
4   "flight": {
5     "id": 2,
6     "airline": {
7       "id": 1,
8       "name": "Turk Hava Yollari"
9     },
10    "route": {
11      "id": 1,
12      "departureAirport": {
13        "id": 2,
14        "name": "Istanbul Airport",
15        "code": "IST"
16      },
17      "arrivalAirport": {
18        "id": 1,
19        "name": "Ankara Esenboga",
20        "code": "ESB"
21      }
22    },
23    "basePrice": 1000.0,
24    "seat": 10,
25    "soldSeats": 1
26  },
27  "creditCardMasked": "123456*****3456",
28  "pricePaid": 1000.0,
29  "canceled": false
```

Bileti Tekrar satın alırken fiyat artışı:

```
26  },
27  "creditCardMasked": "123456*****3456",
28  "pricePaid": 1100.0,
29  "canceled": false

  soldSeats: 5
},
"creditCardMasked": "123456*****3456",
"pricePaid": 1200.0,
"canceled": false
}
```

Ticket Delete Methodu:

```
DELETE      ▾ http://localhost:8080/api/tickets/52000acb-d1e3-42fc-b7a2-9ef805e7233b

25   "soldSeats": 2
26 },
27 "creditCardMasked": "123456*****3456",
28 "pricePaid": 1200.0,
29 "canceled": true
30 }
```

H2 DB:

The screenshot shows the H2 Database interface with three separate query panes. The left sidebar lists tables: AIRLINE, AIRPORT, FLIGHT, ROUTE, TICKET, INFORMATION_SCHEMA, and Users. The version is H2 2.3.232 (2024-08-11).

Query 1: SELECT * FROM AIRLINE;

ID	NAME
1	Turk Hava Yollari

(1 row, 2 ms)

Query 2: SELECT * FROM AIRPORT;

Query 3: SELECT * FROM ROUTE;

jdbc:h2:mem:testdb

- ALINE
- AIRPORT
- FLIGHT
- ROUTE
- TICKET
- INFORMATION_SCHEMA
- Users

H2 2.3.232 (2024-08-11)

Run Run Selected Auto complete Clear SQL statement:

```
SELECT *
FROM FLIGHT
```

ID	BASE_PRICE	SEAT	SOLD_SEATS	AIRLINE_ID	ROUTE_ID
1	1000.0	10	3	1	1

(1 row, 2 ms)

Edit

jdbc:h2:mem:testdb

- ALINE
- AIRPORT
- FLIGHT
- ROUTE
- TICKET
- INFORMATION_SCHEMA
- Users

H2 2.3.232 (2024-08-11)

Run Run Selected Auto complete Clear SQL statement:

```
SELECT *
FROM ROUTE
```

ID	ARRIVAL_AIRPORT_ID	DEPARTURE_AIRPORT_ID
1	2	1

(1 row, 0 ms)

Edit

jdbc:h2:mem:testdb

- ALINE
- AIRPORT
- FLIGHT
- ROUTE
- TICKET
- INFORMATION_SCHEMA
- Users

H2 2.3.232 (2024-08-11)

Run Run Selected Auto complete Clear SQL statement:

```
SELECT *
FROM TICKET
```

ID	CREDIT_CARD_MASKED	IS_CANCELED	PRICE_PAID	TICKET_NUMBER	FLIGHT_ID
1	123456*****3456	FALSE	1000.0	7daf832-128d-42f6-9bd8-dfd76b26e401	1
2	123456*****3456	FALSE	1100.0	84743294-676c-44c4-83d8-b2686c03540a	1
3	123456*****3456	FALSE	1200.0	5559edb7-993e-42f3-91b5-dc2737f02557	1

(3 rows, 1 ms)

Edit