Server Side APIs

Part 1

What is an API?

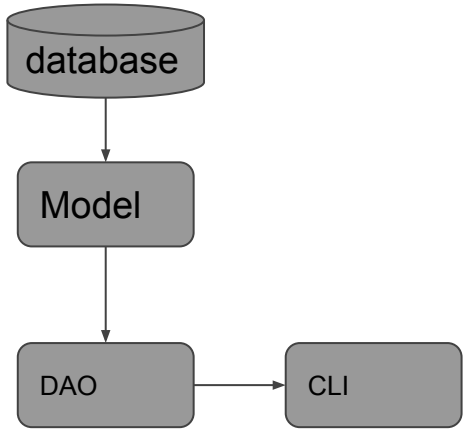
- A set of functions and/or procedures designed to interact with an external system.
- Modern cloud architecture relies heavily on API’s.

API's as a source of data

- We have explored various ways of obtaining data, starting from having Java read a text file, to building a sophisticated relational database like PostgreSQL.
- API's could potentially be yet another source of data for other applications to consume.

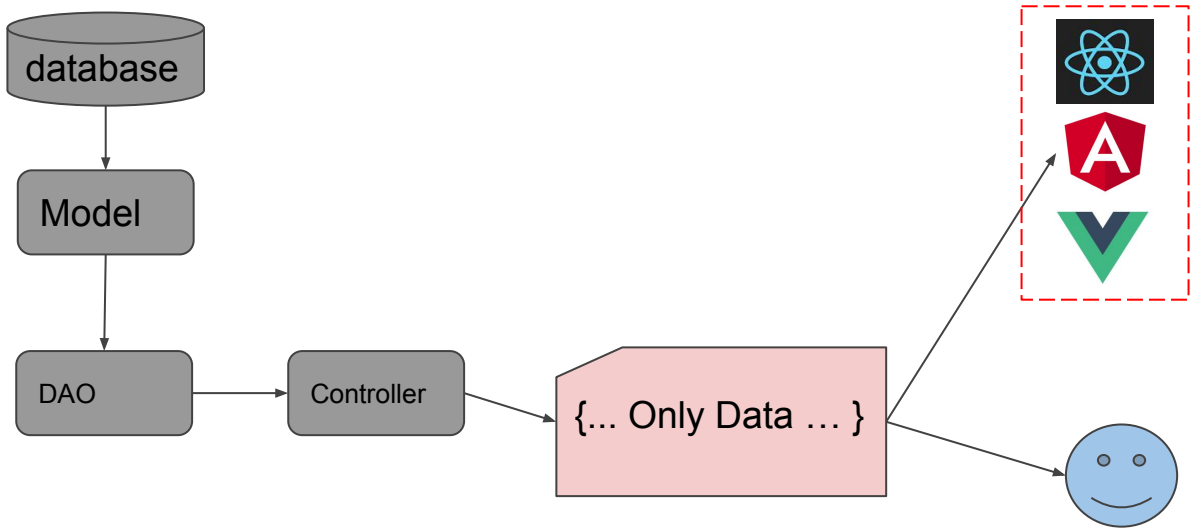
API's as a source of data

Consider the JDBC styled applications we build in Module 2



API's as a source of data

Now consider what happens if instead of a CLI we only send and receive data:



This data can now be sent and used by JS applications.

Building these "front end" apps is the focus of Mod 4!

… or just for a user to analyze the raw data. (analysts and data scientists perhaps?)

REST Controllers

- In order to send and receive data we will be building REST controllers.
  - REST is short for **Re**presentational **S**tate **T**ransfer.
- The Spring framework makes this easy. We have at our disposal a class level annotation called **@RestController**.

```
@RestController

public class ProductReviewsController {
… }
```

Let’s try this out!

Structuring an endpoint (GET)

This is a request mapping that takes care of GET requests, that respond to the /hotels. Note the method level annotation @RequestMapping.

```
@RequestMapping(path = "/hotels", method = RequestMethod.GET)
public List<Hotel> list() {
   return hotelDAO.list();
}
```

Here we have another GET that respond to /hotels with a path variable. An actual call to this endpoint would look something like “/hotels/5”.

```
@RequestMapping(path = "/hotels/{id}", method = RequestMethod.GET)
public Hotel get(@PathVariable int id) {
   return hotelDAO.get(id);
}
```

Structuring an endpoint (GET) with parameters

We can specify that our GET endpoints take parameters, consider the following:

```
@RequestMapping(path = "/hotels/filter", method = RequestMethod.GET)
 public List<Hotel> filterByStateAndCity(@RequestParam String state, @RequestParam(required = false) String city) {
      ...
}
```

The above mapping specifies that the get request can take two parameters, one for state and the other for city, with the latter being optional. An acceptable call to this endpoint could be:

/hotels/filter?state=Ohio&city=Cleveland

Structuring an endpoint (POST)

POST requests require a body which contains the data we are sending to the endpoint.

```
@RequestMapping( path = "/hotels/{id}/reservations", method = RequestMethod.POST)
public Reservation addReservation(@RequestBody Reservation reservation, @PathVariable("id") int hotelID) {
    return reservationDAO.create(reservation, hotelID);
}
```

The above mapping handles a POST request which takes a body that can be deserialized into a Reservation object.

Structuring an endpoint (POST)

Java Class

```
public class Reservation {

    private int id;
    private int hotelID;
    private String fullName;
    private String checkinDate;
    private String checkoutDate;
    private int guests;

// … getters + setters + other methods
}
```

Request Body (in JSON)

```
{
    "id": 5,
    "hotelID": 5,
    "fullName": "Jane Smith",
    "checkinDate": "2020-06-09",
    "checkoutDate": "2020-06-12",
    "guests": 4
}
```

A valid request sent to the endpoint should have a body in the form of JSON, the @RequestBody annotation will deserialize it into an object of the specified class.

Let’s implement some GETs and POSTs

Consuming GETs and POSTs

- In the first two lectures we talked about how to consume an API from a Java Application using a RestTemplate object.
  - At that time, we had to use a “simulated API” courtesy of JSON Server.
- We now know how to build the real thing, there is no need for JSON Server!

Consuming a GET

This is the GET endpoint with a path variable that we defined today:

```
@RequestMapping(path = "/hotels/{id}", method = RequestMethod.GET)
public Hotel get(@PathVariable int id) {
    return hotelDAO.get(id);
}
```

… and this is how a different Java application can consume it:

```
hotel = restTemplate.getForObject(BASE_URL + "hotels/" +  id, Hotel.class);
```

Consuming a POST

This is the GET endpoint with a path variable that we defined today:

```
@RequestMapping( path = "/hotels/{id}/reservations", method = RequestMethod.POST)
public Reservation addReservation(@RequestBody Reservation reservation, @PathVariable("id") int hotelID) {
    return reservationDAO.create(reservation, hotelID);
}
```

… and this is how a different Java application can consume it:

```
HttpHeaders headers = new HttpHeaders();
headers.setContentType(MediaType.APPLICATION_JSON);
HttpEntity<Reservation> entity = new HttpEntity<>(reservation, headers);
 reservation = restTemplate.postForObject(BASE_URL + "hotels/" + reservation.getHotelID() + "/reservations",  entity, Reservation.class);
```