Please Complete The Socrative Pulse Survey!

URL: gosocrative.com
Room Name: JAVAGOLD

Review:

1. What does JSON stand for and what is it?
2. What is a web service?
3. What is Spring? How is it useful to us?
4. What does REST stand for and what is it?
5. If it is successful, what range of HTTP status codes would you expect to see in the response?
6. What is an IP address? What is it used for?
7. What is a client in a client/server relationship?
8. What is a server in a client/server relationship?

Server Side APIs

Part 1

Objectives

- Describe the MVC pattern and why programmers use it
- Describe the differences and responsibilities of backend code versus frontend code
- Describe the purpose and scope of the Spring Boot Core MVC framework
- Create a web application that accepts GET and POST requests and returns JSON
- Run a web application on a local development machine
- Connect a local client application to a locally running API
- Run a web application on a local development machine
- Implement Javadocs on our methods

Model View Controller Definition

Model View Controller (MVC) is a philosophy of application development that divides the application based on responsibilities.

There are many frameworks capable of delivering a MVC application.

Model View Controller Definition

Model

- application state and business logic
- only part of the application that talks to the database
- Models could be classes

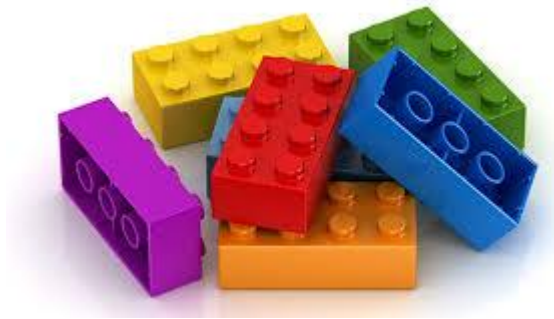
View

- presents data to user
- accepts input from user
- Views might be a desktop display, a mobile display, a file output based on a model

Controller

- Takes input from the view and passes it to the appropriate model objects
- Grabs all necessary building blocks and organizes them for the output
- Takes results from the model and passes it to the appropriate view

Model



View

Controller

View

Life cycle of a *typical* HTTP request:

1) The user/client sends the HTTP request
2) The controller intercepts it
3) The controller calls the appropriate service
4) The service calls the appropriate DAO, which returns some persisted data (for example)
5) The service treats the data, and returns data to the controller
6) The controller stores the data in the appropriate model and calls the appropriate view
7) The view gets *instantiated* with the model's data and gets returned as the HTTP response.

Spring

The Spring Framework is a very popular framework that abstracts various aspects of application development in order to create robust web applications faster.

A strong grasp of basic Java is needed to fully take advantage of Spring!

Spring Boot – an extension of Spring

What is an API?

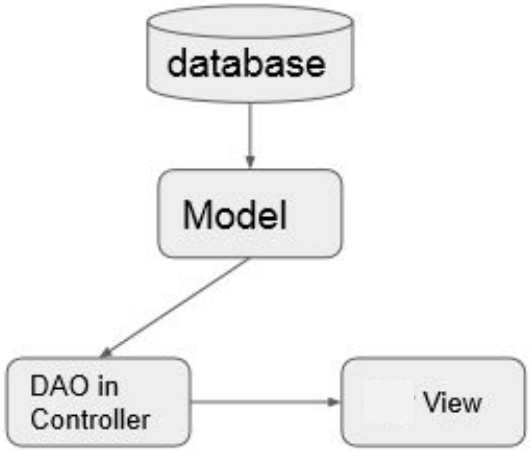
- A set of functions and/or procedures designed to interact with an external system.
- Modern cloud architecture relies heavily on API’s.
- Web API – accessible on the Internet.

API as a source of data

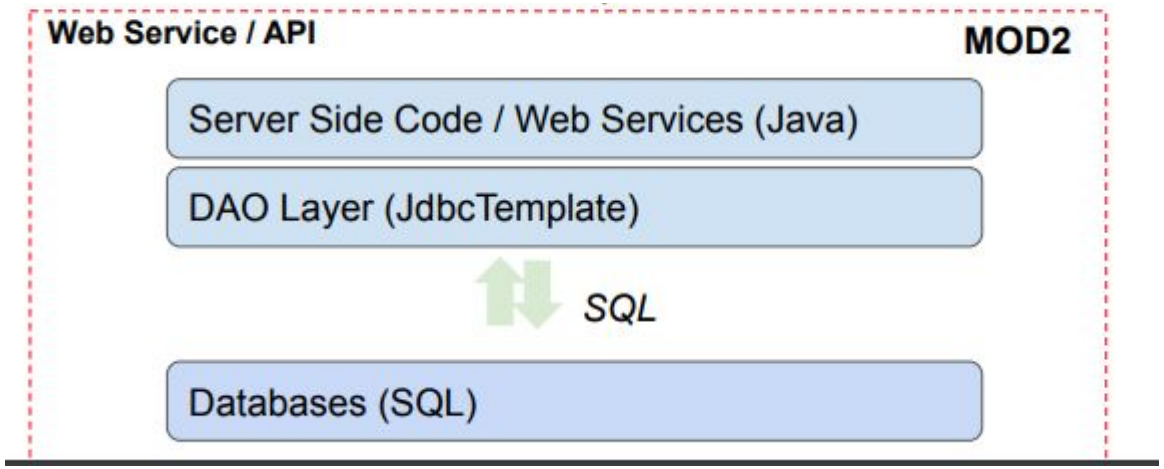
- We have explored various ways of obtaining data, starting from having Java read a text file, to building a sophisticated relational database like PostgreSQL.
- APIs could potentially be yet another source of data for other applications to consume.

API as a source of data

Consider what happens after a controller receives a GET request:

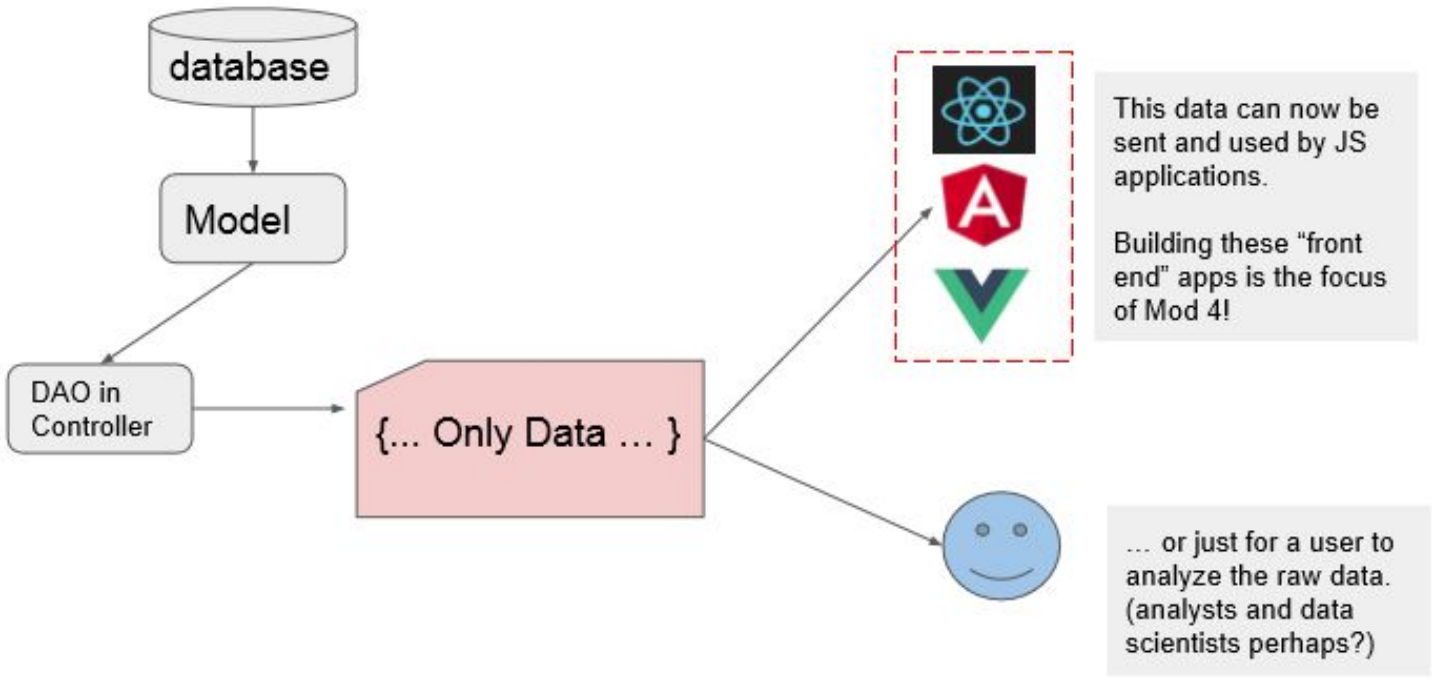


This is the direction data flows in a MVC application

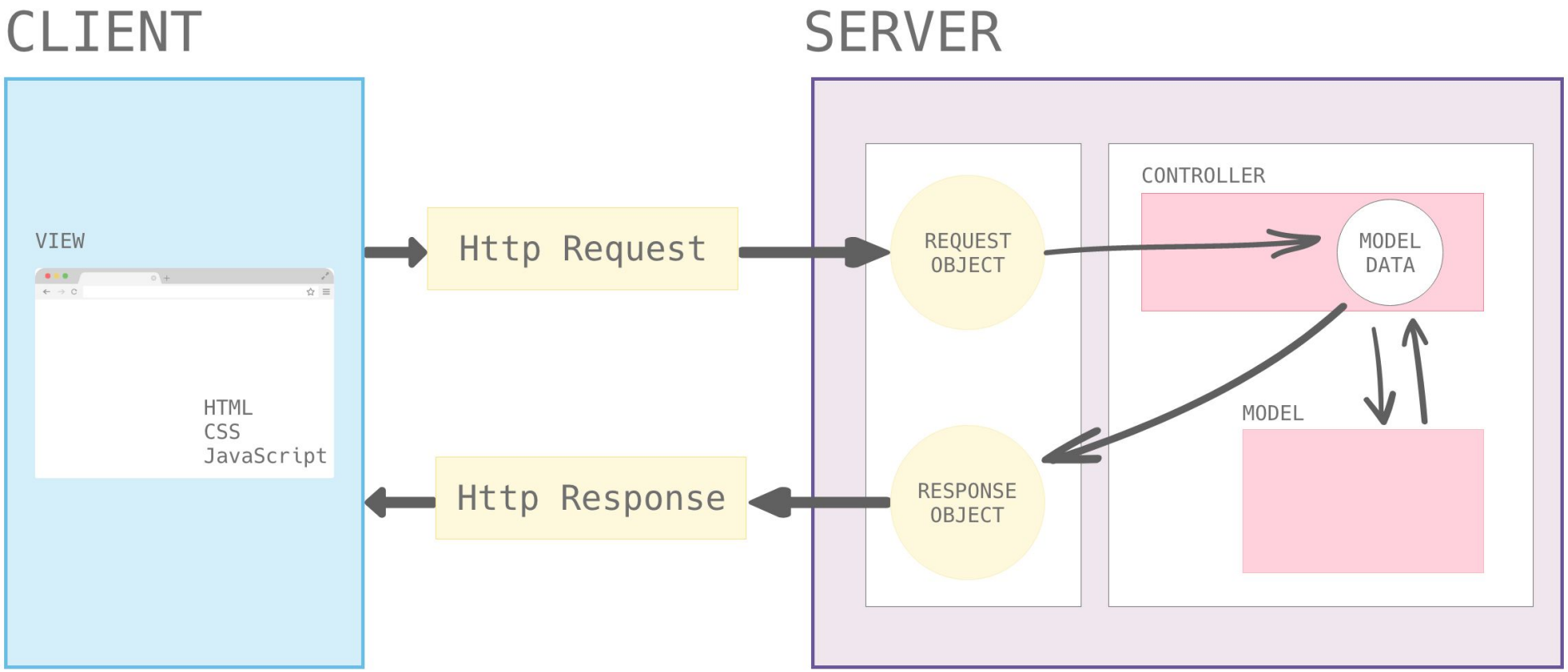


API as a source of data

- Can have controller return just the data.



Request and Response life cycle



REST Controllers

- In order to send back data instead of a view, we set our controllers to REST controllers.
  - REST is short for **Re**presentational **S**tate **T**ransfer. It is an architectural style.
  - REST client apps use HTTP GET/POST methods to invoke RESTful web services.
  - Spring Boot make it easy for us to mark a class as a REST controller using a special annotation (@RestController)

REST Controllers

- In theory, Spring makes this easy. We have at our disposal an annotation called @RestController.

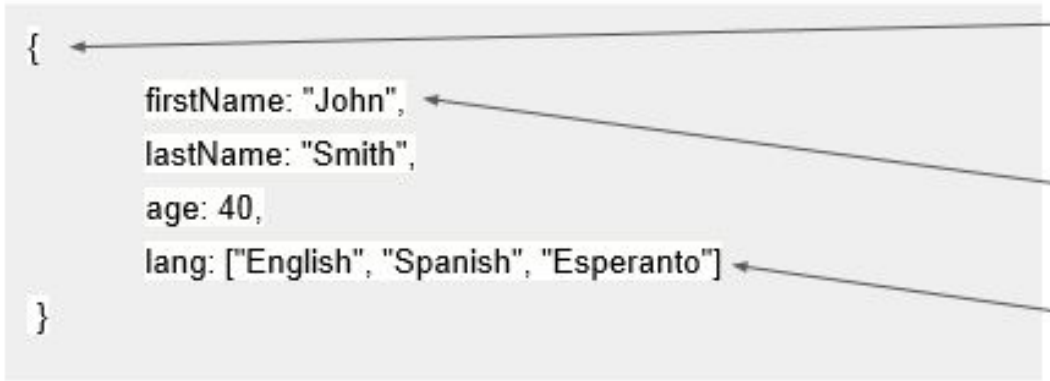
```
@RestController
public class ProductReviewsController  {
   …
}
```

JSON

- The data generated by the controllers are in a format called JSON.
- JSON is simple! Only three rules:
  - Objects in JSON's are delimited by curly braces { … }
  - Arrays in JSON's are delimited by brackets [...]
  - Data is listed in key-value pairs (key : value)

JSON example

Here is an example of JSON data:



- The object itself is enclosed with a set of curly braces.
- Each property of the object is listed as a key value pair.
- An array is enclosed with square brackets.

Handler Methods

- Annotated with @RequestMapping
  - Handle a request from web
  - path= argument
    - Maps to the request path
  - method= RequestMethod.XXX
    - Specifies what type of request will cause the method to execute

Structuring an endpoint (GET)

This is a request mapping that takes care of GET requests, that respond to the /hotels.

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

This is a request mapping that takes care of GET requests, that respond to the /hotels with a path variable. An actual call to this endpoint would look something like “/hotels/5”. The variable id would thus take on the value of 5

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

Model

```
package com.techelevator.reservations.models;

public class Hotel {

    private int id;
    private String name;
    private Address address;
    private int stars;
    private int roomsAvailable;
    private double costPerNight;
    private String coverImage;

    public Hotel() {
    }

    public Hotel(int id, String name, Address address, int stars,
                         int roomsAvailable, double costPerNight) {
        this.id = id;
        this.name = name;
        this.address = address;
        this.stars = stars;
        this.roomsAvailable = roomsAvailable;
        this.costPerNight = costPerNight;
        this.coverImage = "default-cover-image.png";
    }
…
```

Controller

```
package com.techelevator.reservations.controllers;

import com.techelevator.reservations.dao.HotelDAO;
import com.techelevator.reservations.dao.MemoryHotelDAO;
import com.techelevator.reservations.dao.MemoryReservationDAO;
import com.techelevator.reservations.dao.ReservationDAO;
import com.techelevator.reservations.exception.HotelNotFoundException;
import com.techelevator.reservations.exception.ReservationNotFoundException;
import com.techelevator.reservations.models.Hotel;
import com.techelevator.reservations.models.Reservation;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;
import java.util.ArrayList;
import java.util.List;

@RestController
public class HotelController {

    private HotelDAO hotelDAO;
    private ReservationDAO reservationDAO;

    public HotelController() {
        this.hotelDAO = new MemoryHotelDAO();
        this.reservationDAO = new MemoryReservationDAO(hotelDAO);
    }
```

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

Inside the method, having made the above call, the String state will take on the value of Ohio, and the String city will take on the value of Cleveland.

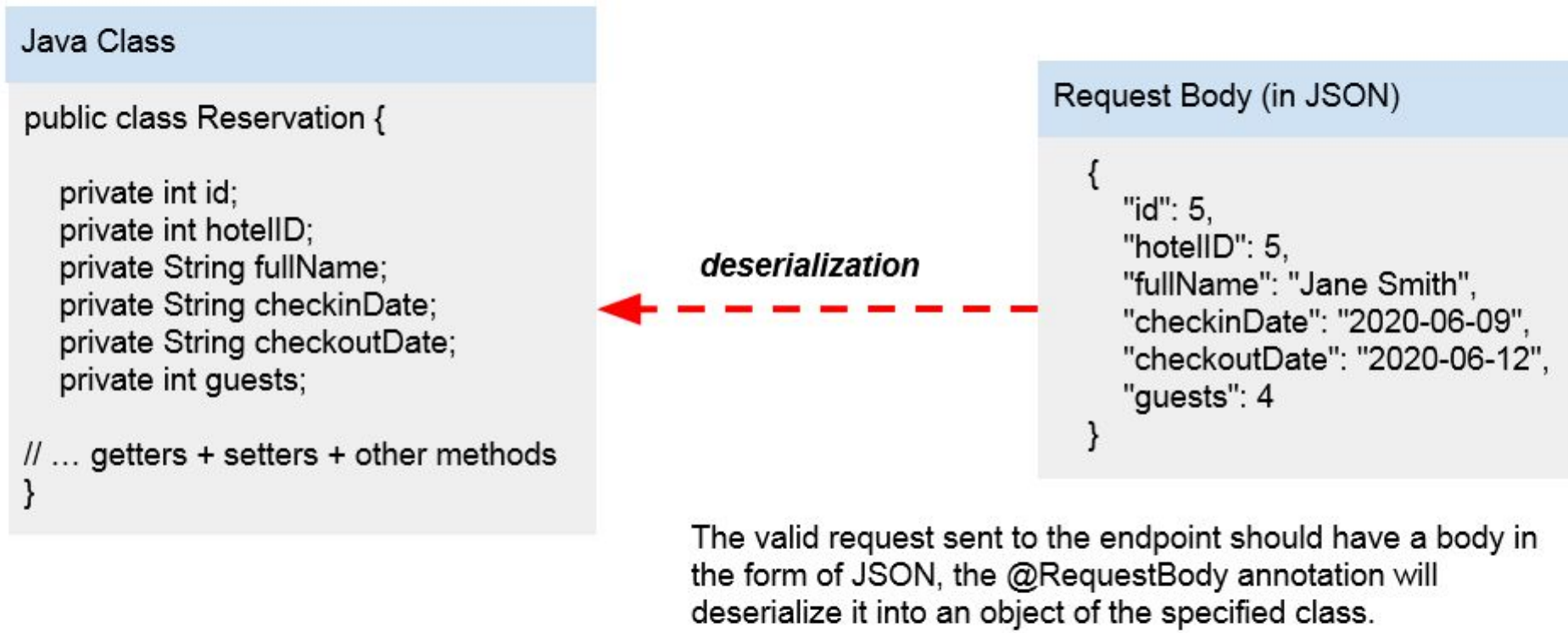
Structuring an endpoint (POST)

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)



The 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.