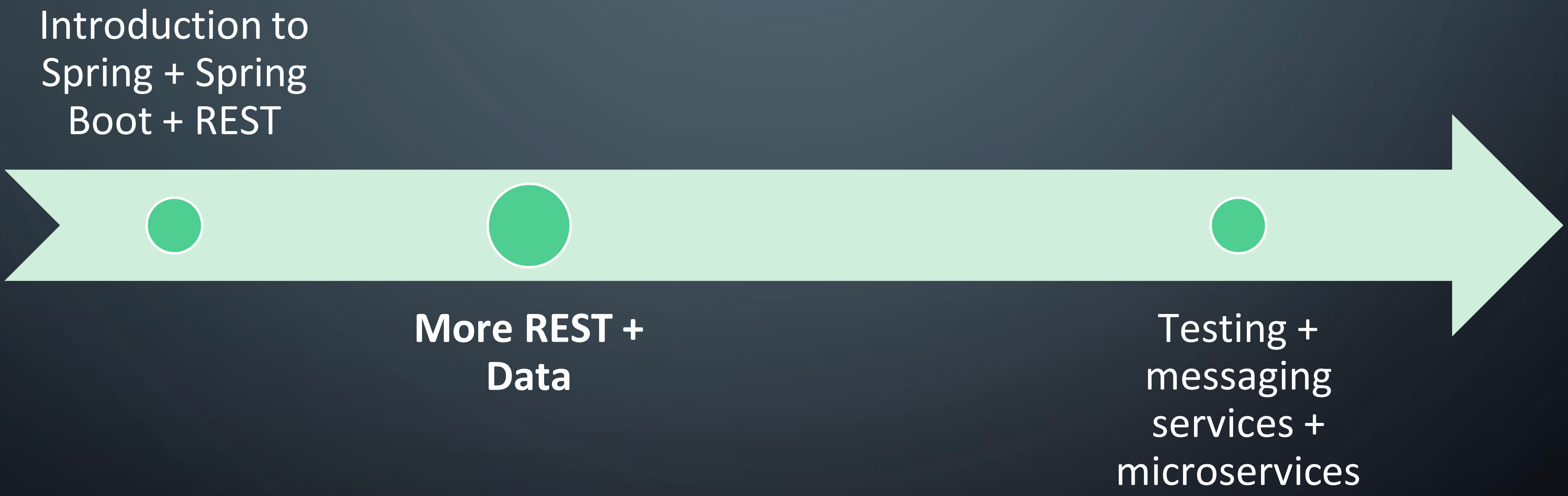
SPRING BOOT FUNDAMENTALS

GOAL OF THE COURSE

- Solid basic understanding of Spring and Spring Boot
- Know different projects and be able to create implementations with them
- Be able to independently modify an existing Spring application

COURSE OVERVIEW


Introduction to Spring + Spring Boot + REST
More REST + Data
Testing + messaging services + microservices

TODAY’S CONTENT

- RESTful services with spring
- Spring MVC
- Requests and responses
- Write your own services
- Introduction transaction management
- Introduction Spring security

WHAT ARE WE DOING TODAY?

Spring

- Make endpoints (Spring Rest), call endpoints (RestTemplates) and automagic databases (JPA)!
- Spring REST
- Controllers
- Coding magic
- Spring Data
- Some more coding magic
- Introduction security
- Introduction transaction management

HOW ARE YOU?

- How are you doing?
- Do you expect any disturbances today?
- What do you hope to learn today?

EXERCISE

Get some REST

- Take 5 minutes to come up with a short explanation of what REST is.

WHAT IS A REST SERVICE?

Representational State Transfer

- **Follows 4 principles:**
- **1. Resource identification through URI**
- e.g. <host>:<port>/application/persons)

- **2. Uniform interface**
- Manipulation of resources through CRUD operations.

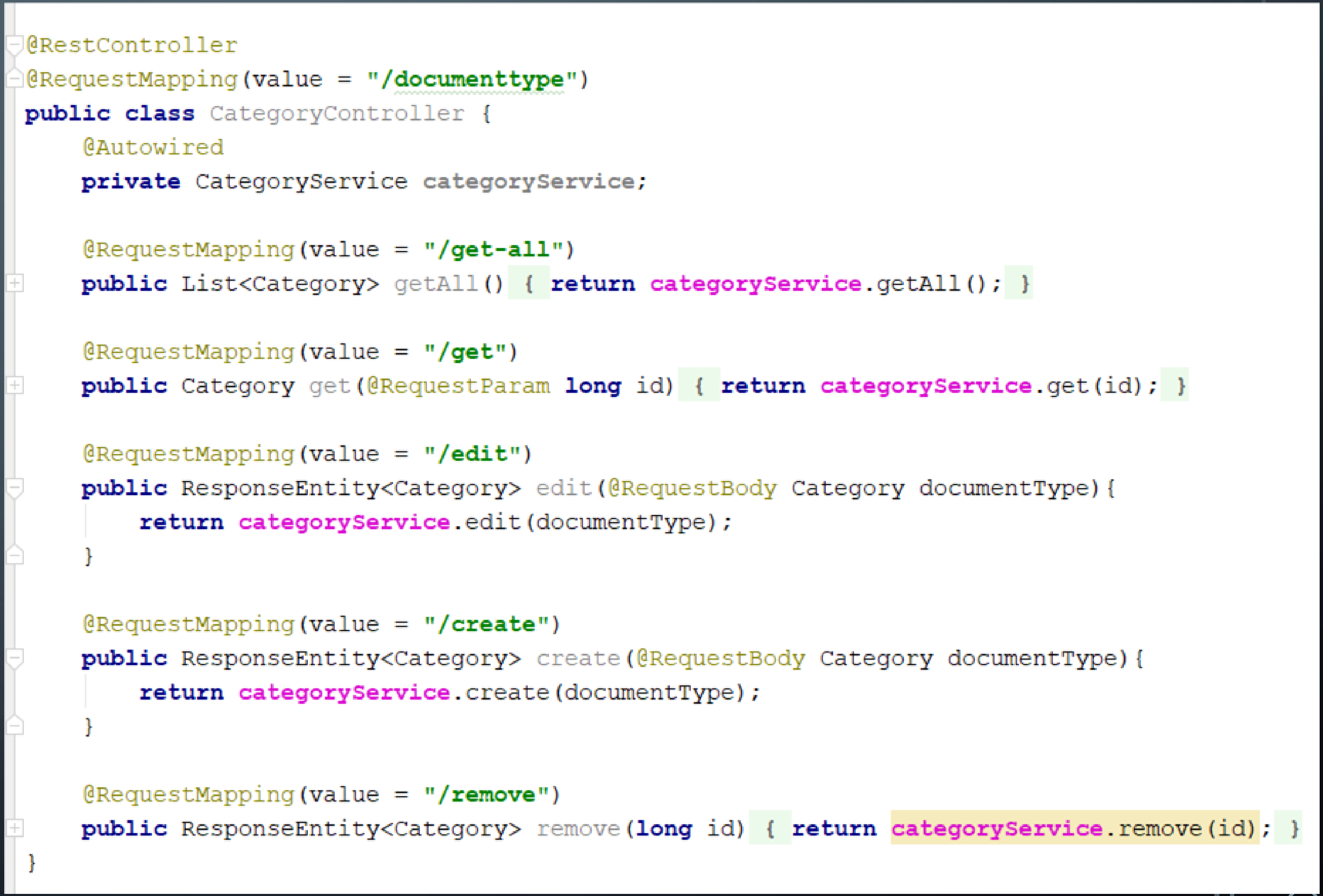
- **3. Self-descriptive messages**
- Resources are decoupled from their representations so the content can be accessed in more than one format (plain text, HTML, XML, JSON etc etc). Metadata in the message is used to do authentication, detect errors, caching, access control etc etc…

- **4. Stateful interactions through hyperlinks**
- Every interaction with a resource is stateless. Stateful interactions can be done through URI manipulation, cookies or hidden form fields.

SPRING CONTROLLERS

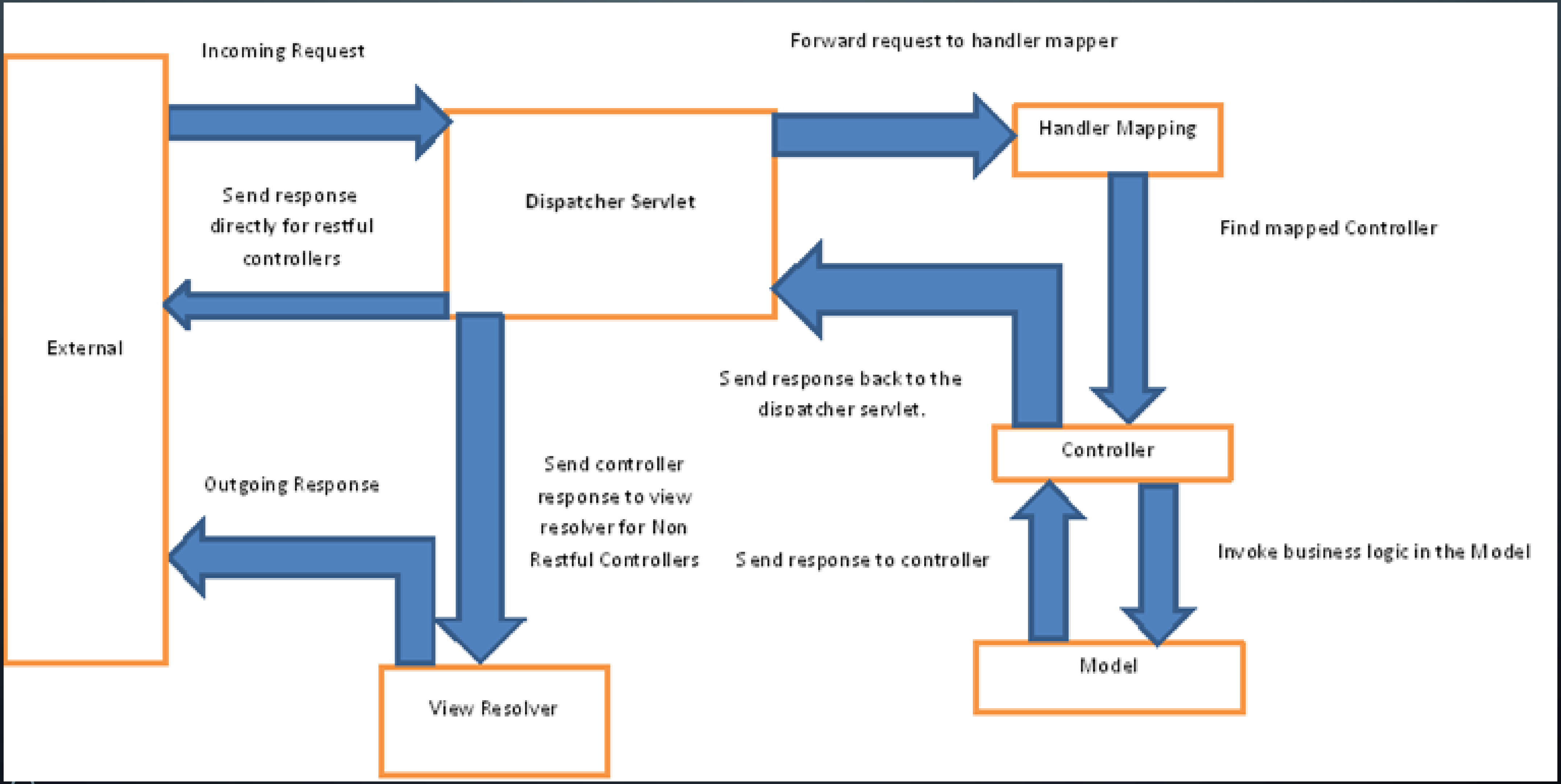
What are they?

- Used as endpoints, routing the calls from external applications or other microservices.



```
@RestController
@RequestMapping(value = "/documenttype")
public class CategoryController {
    @Autowired
    private CategoryService categoryService;

    @RequestMapping(value = "/get-all")
    public List<Category> getAll() { return categoryService.getAll(); }

    @RequestMapping(value = "/get")
    public Category get(@RequestParam long id) { return categoryService.get(id); }

    @RequestMapping(value = "/edit")
    public ResponseEntity<Category> edit(@RequestBody Category documentType){
        return categoryService.edit(documentType);
    }

    @RequestMapping(value = "/create")
    public ResponseEntity<Category> create(@RequestBody Category documentType){
        return categoryService.create(documentType);
    }

    @RequestMapping(value = "/remove")
    public ResponseEntity<Category> remove(long id) { return categoryService.remove(id); }
}
```

SPRING CONTROLLERS


Incoming Request
Forward request to handler mapper
Handler Mapping
Dispatcher Servlet
Send response directly for restful controllers
Find mapped Controller
External
Send response back to the dispatcher servlet.
Controller
Send controller response to view resolver for Non Restful Controllers
Outgoing Response
Send response to controller
Invoke business logic in the Model
Model
View Resolver

CONTENT NEGOTIATION

- The mechanism that determines the presentation of the request and the response, for example XML or JSON
- Most commonly done with @ResponseBody

EXERCISE

Take 5 minutes to find the answer to this question:

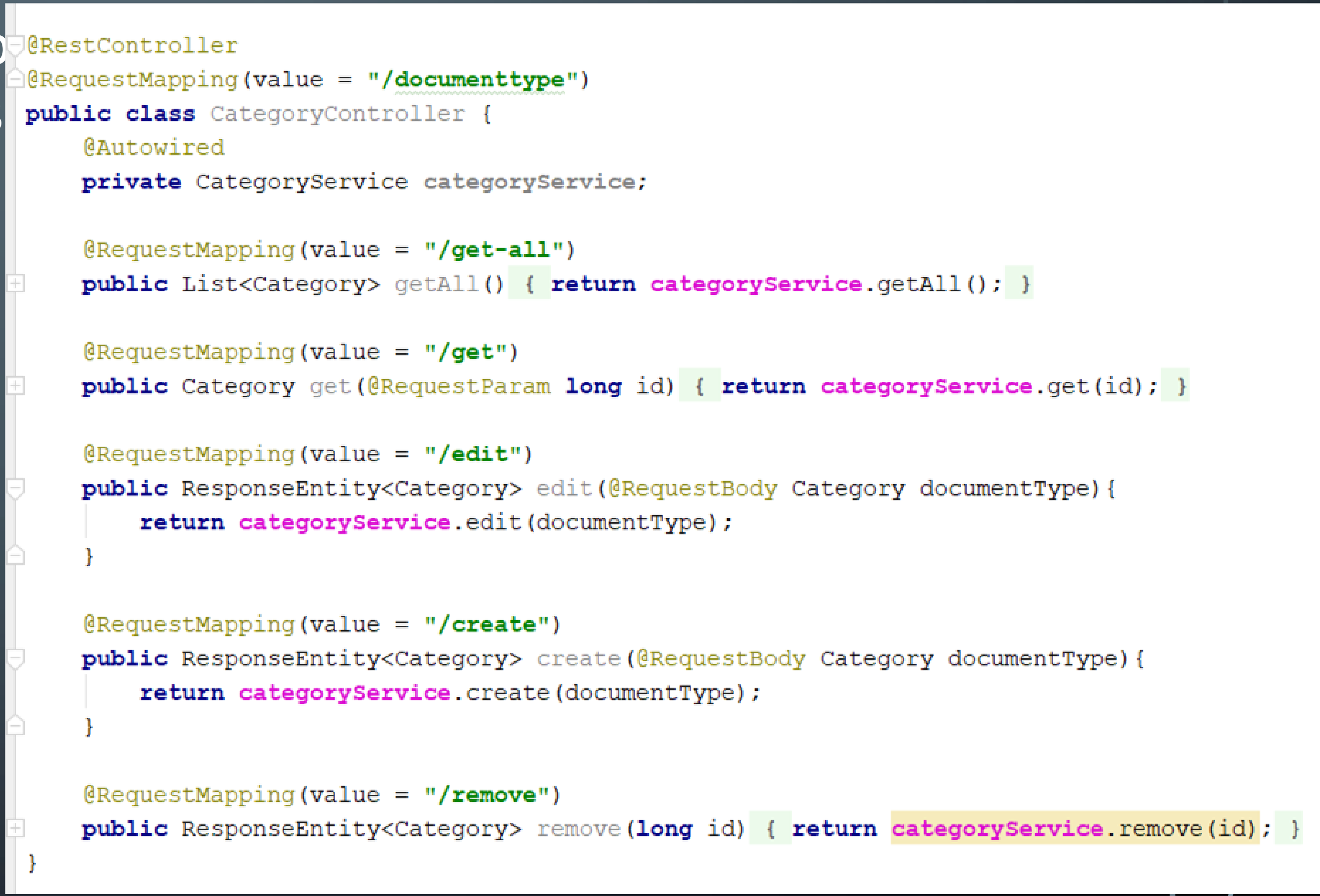
- What is the difference between a Spring Controller and a Spring RestController?

SPRING REST CONTROLLERS

What’s the difference between Sp
Controllers and Spring Controllers

- **They are almost the same!**
- The difference is that for @RestController, Spring will help you (again) to make it less verbose.
- @RestController == @ResponseBody and @Controller and marking the class as a request handler for RESTful web services.

- If you don’t use @RestController, the Controller will try to return paths to the associated view.



```
@RestController
@RequestMapping(value = "/documenttype")
public class CategoryController {
    @Autowired
    private CategoryService categoryService;

    @RequestMapping(value = "/get-all")
    public List<Category> getAll() { return categoryService.getAll(); }

    @RequestMapping(value = "/get")
    public Category get(@RequestParam long id) { return categoryService.get(id); }

    @RequestMapping(value = "/edit")
    public ResponseEntity<Category> edit(@RequestBody Category documentType){
        return categoryService.edit(documentType);
    }

    @RequestMapping(value = "/create")
    public ResponseEntity<Category> create(@RequestBody Category documentType){
        return categoryService.create(documentType);
    }

    @RequestMapping(value = "/remove")
    public ResponseEntity<Category> remove(long id) { return categoryService.remove(id); }
}
```

SENDING PARAMETERS TO SPRING CONTROLLER

- @RequestParam > gets values from URI after the ?
- @PathVariable > gets values directly from specified place in URL . In the @RequestMapping the position of the path variable is specified with {}
- @RequestBody > Automatically maps the body of the http request to the specified object type in the parameter list

CONTROLLER SERVICE REPOSITORY PATTERN

- It is not officially called this as a pattern, but it is extremely common.
- Controller: controls the incoming requests and routs them to the correct service
- Service: this is where the logic is
- Repository: called from service when data needs to be fetched, changed or stored

ASSIGNMENT 1

We need you!

- Our Prime Minister needs your votes to build himself a wall! And he's hoping for the people to give him support. Only problem is, they only have a red pencil system to vote, and sadly, there's only one pencil left. They will need years to vote! So to help our PM out, we need a REST application where we can store the votes in a list. When you vote ('anonymously'), you leave behind your name, age and vote (yes or no).

- **Technical description**
- Write a REST Spring application that is able to perform simple CRUD operations on an internal List of Objects (votes). Just an in memory HashMap is sufficient. The operations are called through any Postman-like application.

- **Bonus**
- Take into account the concurrent access of your 'data'.
- We also need an endpoint for getting the absolute numbers of yesses and no's.
- The key in your HashMap must be a unique set of characteristics of the Votee, used to prevent 'double' voting.

- **Literature**
- Why and when would you rather use a REST service than any other stateful service?

EVALUATING THE ASSIGNMENT

What did you do!?

- You added:
  (Rest)controller(s);
- Service(s);
- Optional: Repository(-y + ies);

- But did you look out for security aspects as who would be able to access the endpoints?

- What if we tried to use our endpoints for malicious inquiries?

CONSUMING REST SERVICES

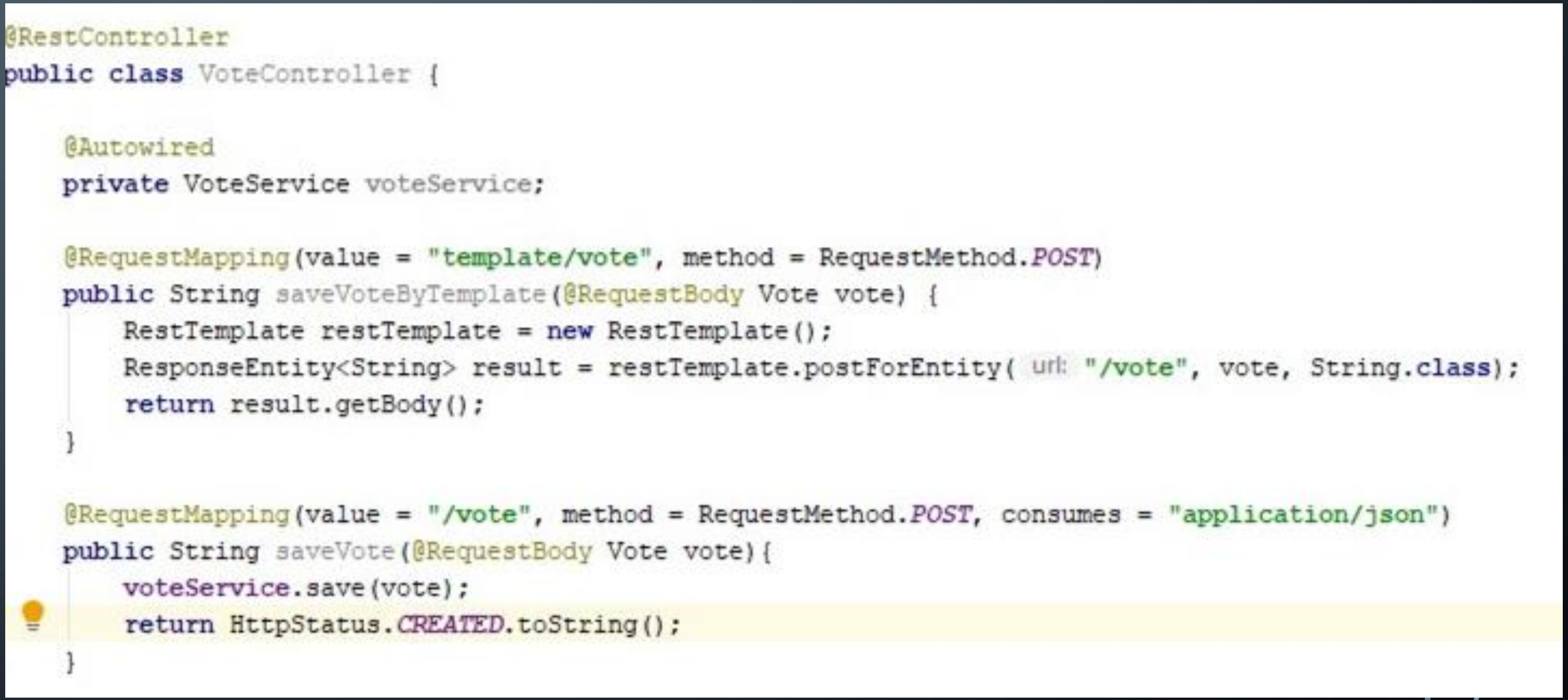
CONSUMING APIS: SPRING RESTTEMPLATE

What is a RestTemplate?

- Spring class that can be used to consume another REST api's/applications/interfaces. Using this RestTemplate results in formatted input/output, handles security issues and exceptions/errors.

- You can fully customize this RestTemplate to do custom security and errorhandling.

- RestTemplate is just the thing for executing synchronous HTTP requests in Spring



```java
@RestController
public class VoteController {

    @Autowired
    private VoteService voteService;

    @RequestMapping(value = "template/vote", method = RequestMethod.POST)
    public String saveVoteByTemplate(@RequestBody Vote vote) {
        RestTemplate restTemplate = new RestTemplate();
        ResponseEntity<String> result = restTemplate.postForEntity( url: "/vote", vote, String.class);
        return result.getBody();
    }

    @RequestMapping(value = "/vote", method = RequestMethod.POST, consumes = "application/json")
    public String saveVote(@RequestBody Vote vote){
        voteService.save(vote);
        return HttpStatus.CREATED.toString();
    }
```

HTTPCLIENT

- Java 11 upgrade for the *HttpURLConnection*

Three core classes:

- *HttpRequest*
- *HttpClient*
- *HttpResponse*

```
HttpRequest request = HttpRequest.newBuilder()
 .uri(new URI("https://postman-echo.com/post"))
 .headers("Content-Type", "text/plain;charset=UTF-8")
 .POST(HttpRequest.BodyPublishers.ofString("Sample request body"))
 .build();
```

EXERCISE

Using a RestTemplate for rest endpoint consumption

- Now consume one endpoint from this api: http://dummy.restapiexample.com/

SPRING DATA

TWO COMMON APPROACHES

SPRING DATA JPA

- Read, create, update, and delete data in a relational database using Java objects (ORM)
- Database agnostic
- Unchecked exceptions
- “Secretly”, JPA is based on JDBC under the hood

SPRING DATA JDBC

- Communicating with the database using SQL
- Database dependent
- Checked exceptions (SQLException for example)

JPA

Java Persistence API

- Remember the definition of Java EE? “Java EE is a very extensive guideline of best practices (protocols) to build and structure your application.”

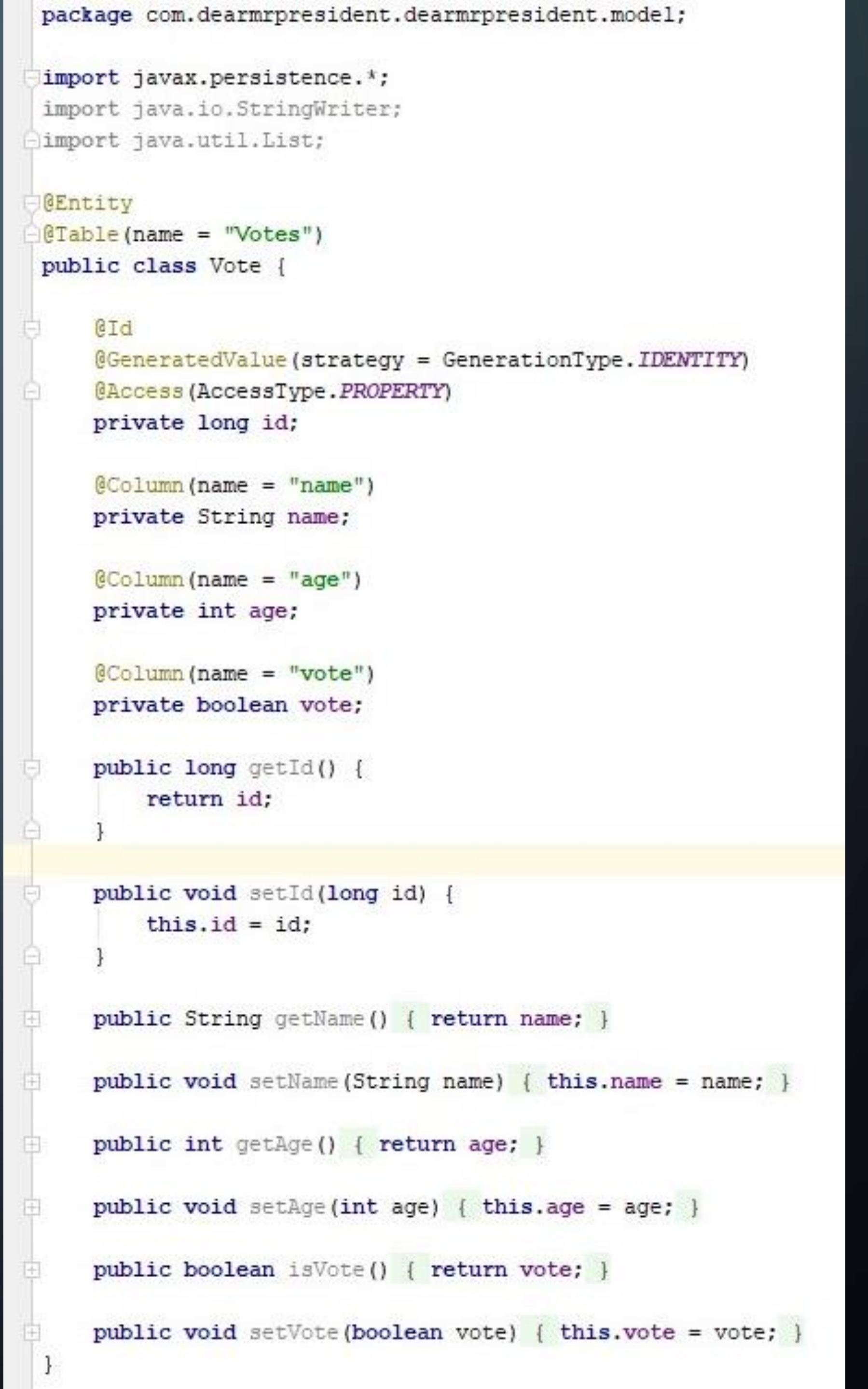
- Definition for JPA: “The Java Persistence API provides a specification for persisting, reading, and managing data from your Java object to relational tables in the database.”

- It’s a specification on interaction between code and the database. Examples of implementations are Hibernate, Eclipse Link, Top Link etc…

HIBERNATE

Making your life easier

- Hibernate is a JPA provider/ implementation.
- Use it to have your entities (models) mapped directly with your database and let the care of persistency, creation of queries etc be handled by your framework!
- How to do it? Annotations! Of course!

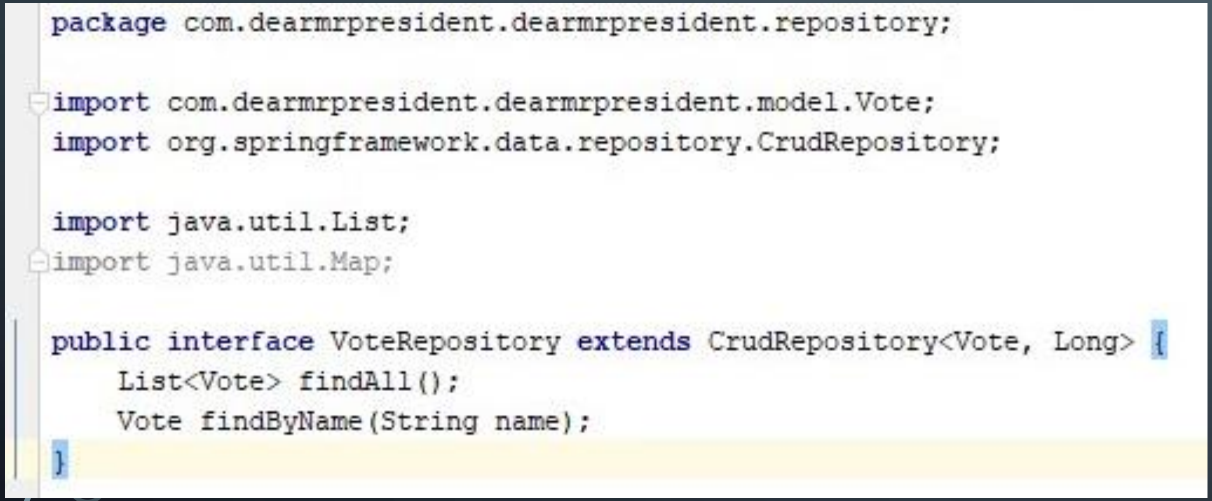


```java
package com.dearmrpresident.dearmrpresident.model;

import javax.persistence.*;
import java.io.StringWriter;
import java.util.List;

@Entity
@Table(name = "Votes")
public class Vote {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Access(AccessType.PROPERTY)
    private long id;

    @Column(name = "name")
    private String name;

    @Column(name = "age")
    private int age;

    @Column(name = "vote")
    private boolean vote;

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public int getAge() { return age; }

    public void setAge(int age) { this.age = age; }

    public boolean isVote() { return vote; }

    public void setVote(boolean vote) { this.vote = vote; }
}
```

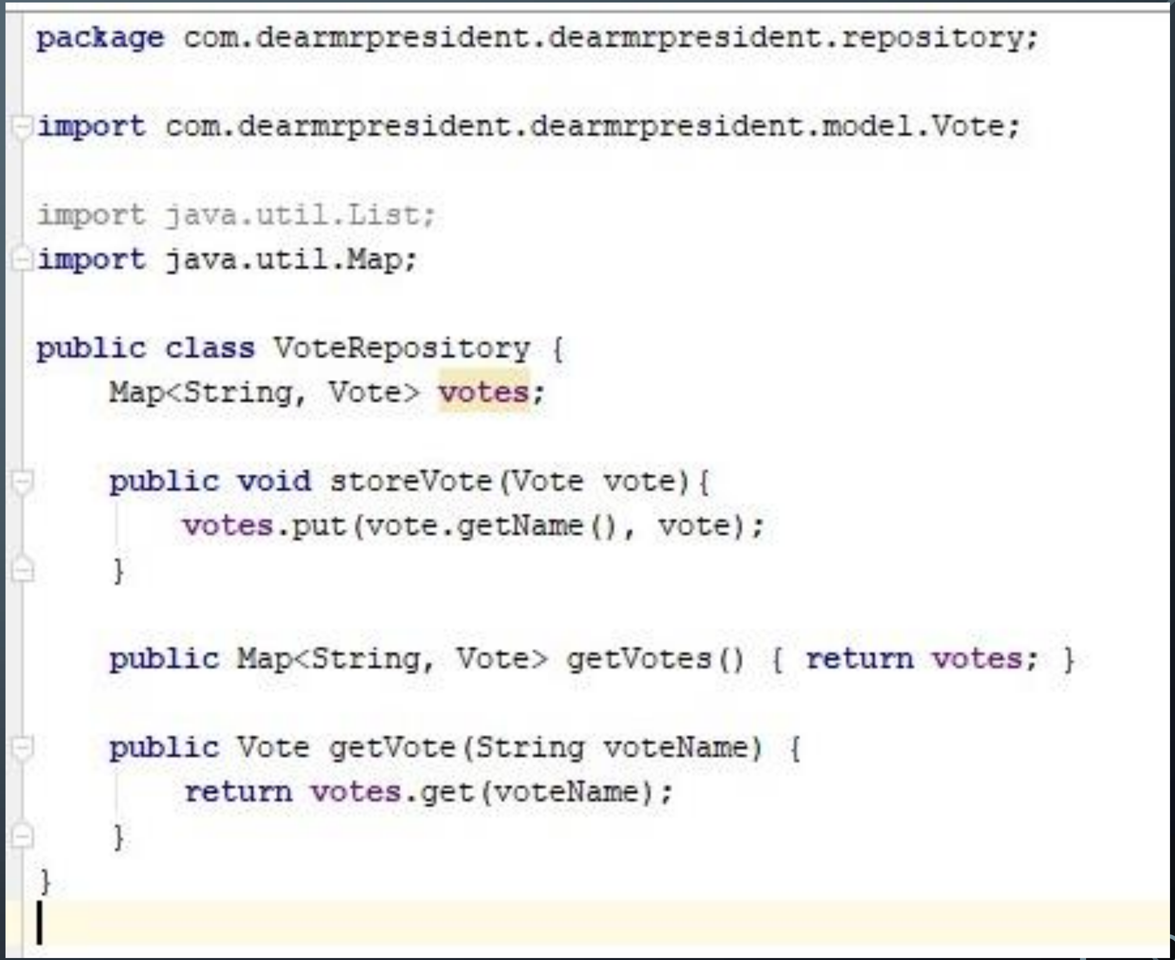
SPRING DATA JPA

Then came along Spring Data

- Makes your repositories more awesome. Helps with implementing JPA repositories to easier access JPA data sources



```java
package com.dearmrpresident.dearmrpresident.repository;

import com.dearmrpresident.dearmrpresident.model.Vote;
import org.springframework.data.repository.CrudRepository;

import java.util.List;
import java.util.Map;

public interface VoteRepository extends CrudRepository<Vote, Long> {
    List<Vote> findAll();
    Vote findByName(String name);
}
```

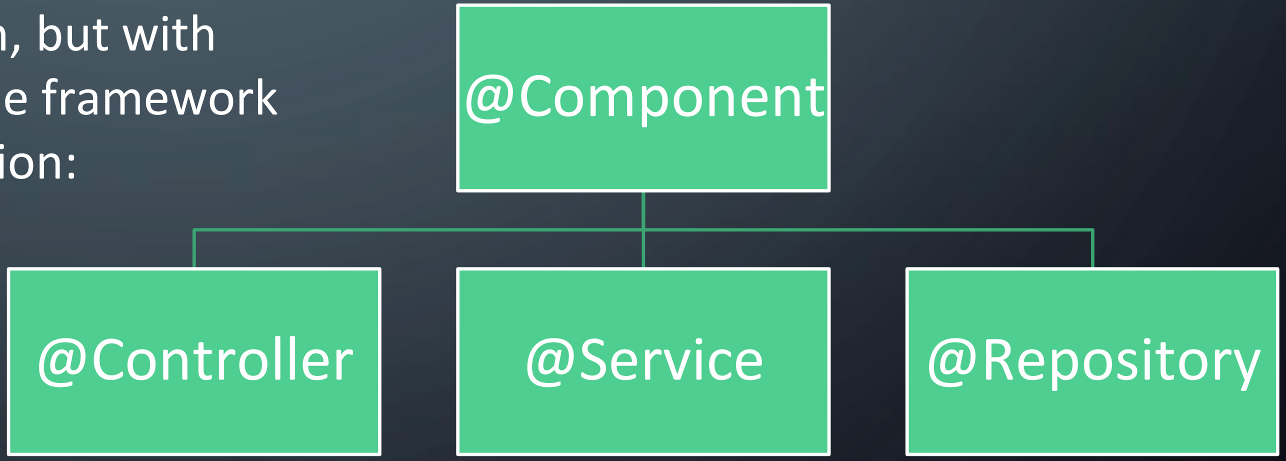


```java
package com.dearmrpresident.dearmrpresident.repository;

import com.dearmrpresident.dearmrpresident.model.Vote;

import java.util.List;
import java.util.Map;

public class VoteRepository {
    Map<String, Vote> votes;

    public void storeVote(Vote vote){
        votes.put(vote.getName(), vote);
    }

    public Map<String, Vote> getVotes() { return votes; }

    public Vote getVote(String voteName) {
        return votes.get(voteName);
    }
}
```

@REPOSITORY, @SERVICE, @CONTROLLER, @RESTCONTROLLER, @COMPONENT

- @Component: registers class as Spring bean, which makes it possibe for them to be injected as a dependency
- These annotations are doing the same as @Component during bean creation, but with some extra special treatment by the framework later based on the specific annotation:

- @Service
- @Controller
- @Repository



CRUDREPOSITORY VS JPAREPOSITORY

CRUDREPOSITORY

- Crud operations
- Doesn’t depend on JPA

JPAREPOSITORY

- Extends CrudRepository and PagingAndSortingRepository
- JPA specific functionality (e.g. flushing and deleting in batch)

SPRING DATA JPA - CACHEABLE

@Cacheable

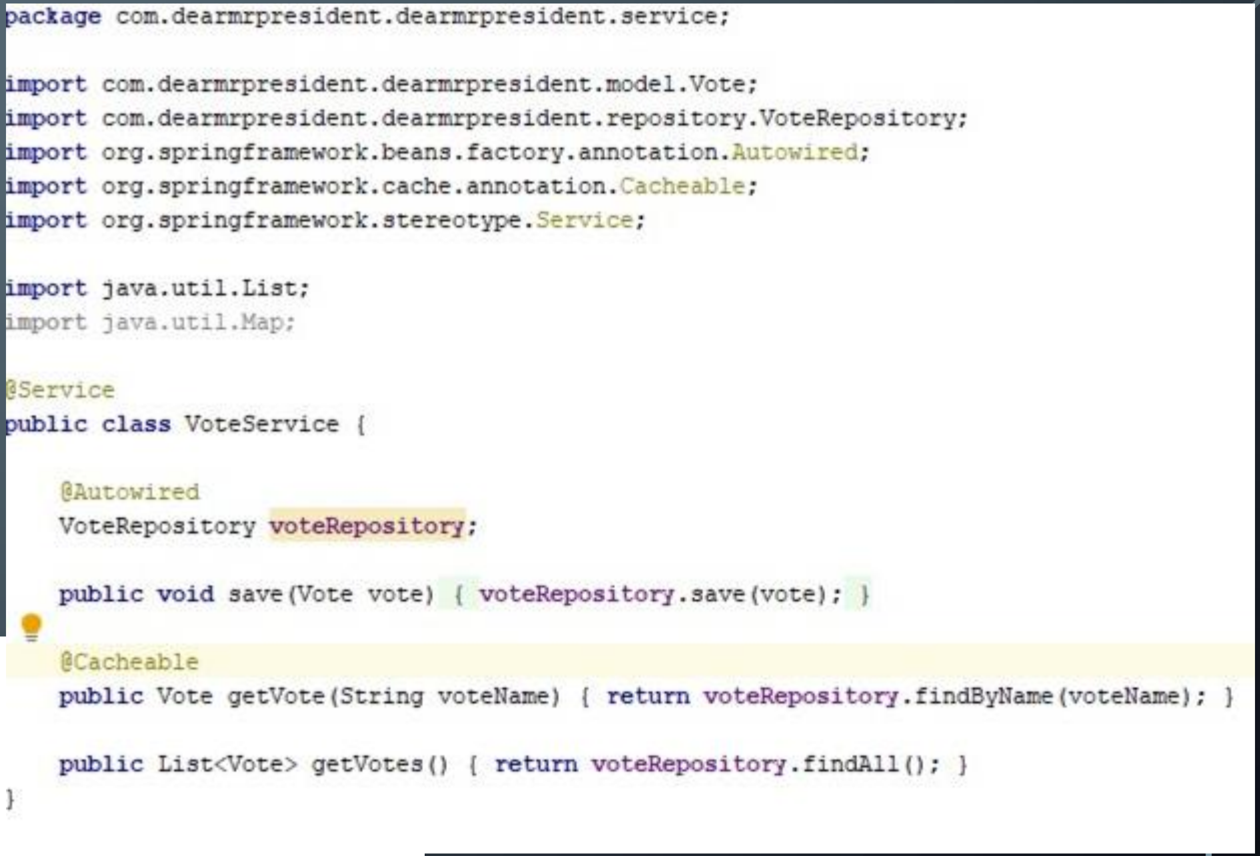
- What would be the advantages of using a cache in your data layer?

- When would you use a cache?
- Would you use it for data that changes often, or data that is more or less static, but often used?

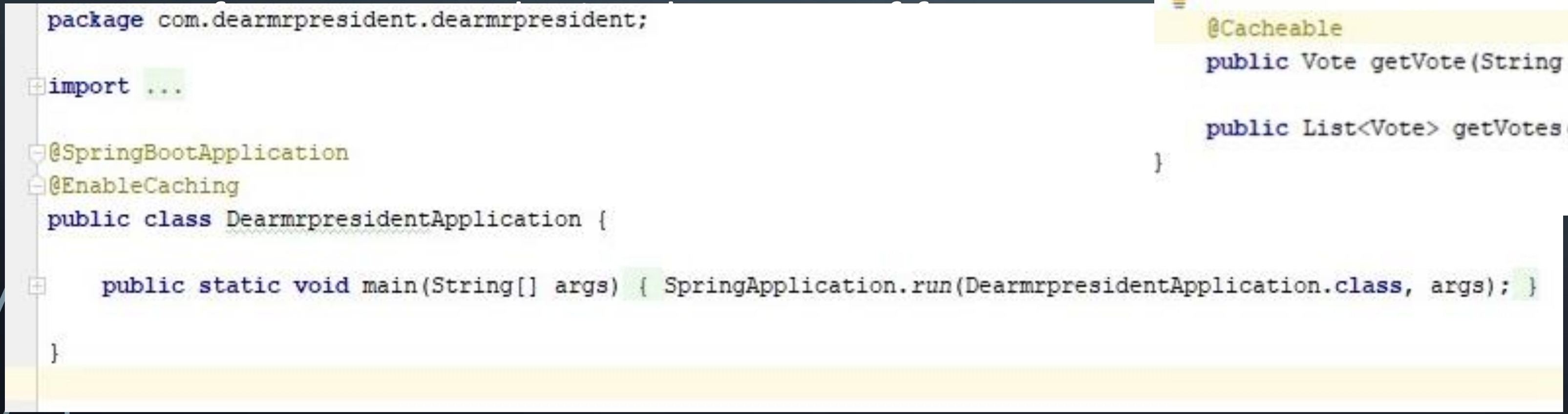
SPRING DATA JPA

Using @Cacheable in your code

- Enable caching in your application class (@EnableCaching).
- Mark the retrieval of data as @Cacheable("name").



```
package com.dearmrpresident.dearmrpresident.service;

import com.dearmrpresident.dearmrpresident.model.Vote;
import com.dearmrpresident.dearmrpresident.repository.VoteRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.cache.annotation.Cacheable;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Map;

@Service
public class VoteService {

    @Autowired
    VoteRepository voteRepository;

    public void save(Vote vote) { voteRepository.save(vote); }

    @Cacheable
    public Vote getVote(String voteName) { return voteRepository.findByName(voteName); }

    public List<Vote> getVotes() { return voteRepository.findAll(); }
}
```



```
package com.dearmrpresident.dearmrpresident;

import ...

@SpringBootApplication
@EnableCaching
public class DearmrpresidentApplication {

    public static void main(String[] args) { SpringApplication.run(DearmrpresidentApplication.class, args); }

}
```

ADDING PROJECTS TO SPRING

- Add the correct dependency to your POM.xml
- And you’re good to go!

- Using project? Usually annotations, sometimes a bit more work as we’ll see when we get to Security

ADDING THE H2 DATABASE

- H2 is typically for testing purposes, in memory database
- Flushed upon restart of the application

- Use the following tutorial to add it in the next exercise: https://www.baeldung.com/spring-boot-h2-database

ASSIGNMENT 2

New insights, new application?

- You can continue on what you started this morning... And change it to fullfill our next assignment!

- Add a database to the application. >> Google H2
- Add the repositories to use the database and make your operations.
- Maybe you need to change something on your models?

- **Tip: You can also generate a new project from the Initializr with the new projects added and continue from there.**

ASSIGNMENT 2B

Hibernate!

- Expand your Vote object
- Create a new Object Person, and give Vote a Person
- Give person an object address. A person can have multiple addresses and persons can share addresses as well.

SPRING DATA REST

SPRING DATA REST

- Quickstart for a REST project
- Built on top of Spring Data project
- Easy to configure and automatically builds APIs
- Tutorial: https://www.baeldung.com/spring-data-rest-intro

INTRODUCTION TO TRANSACTION MANAGEMENT

TRANSACTIONS

What is a Transaction?

- A Transaction is a set of statements that is considered as one. It is only possible for all the statements to take effect. If one of them fails, all the others are rolled back and non of it seems to have happened.

- The transaction can be considered one single action.

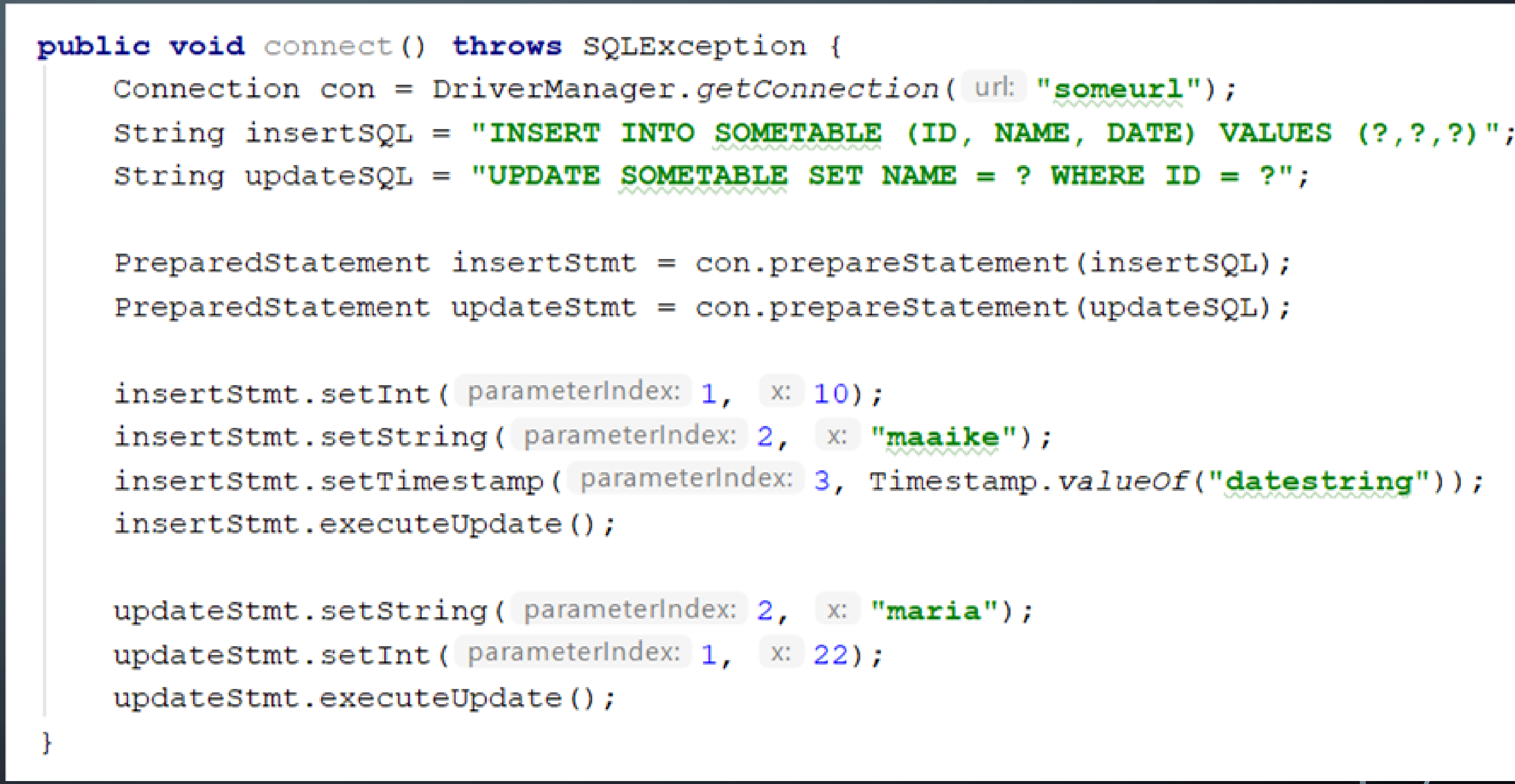
- Important when data integrity needs to be secured.

- Transactions should adhere to ACID: Atomicity, Consistency, Isolation and Durability

TRANSACTIONS

What does a Transaction look like?

- Perform various deleted, update or insert operations using SQL queries.
- If all the operation are successful then perform *commit* otherwise *rollback* all the operations.
- What happens in this example?
- What happens if you change “maria” with a super lengthy String that exceeds the reserved space for a String in the DB?

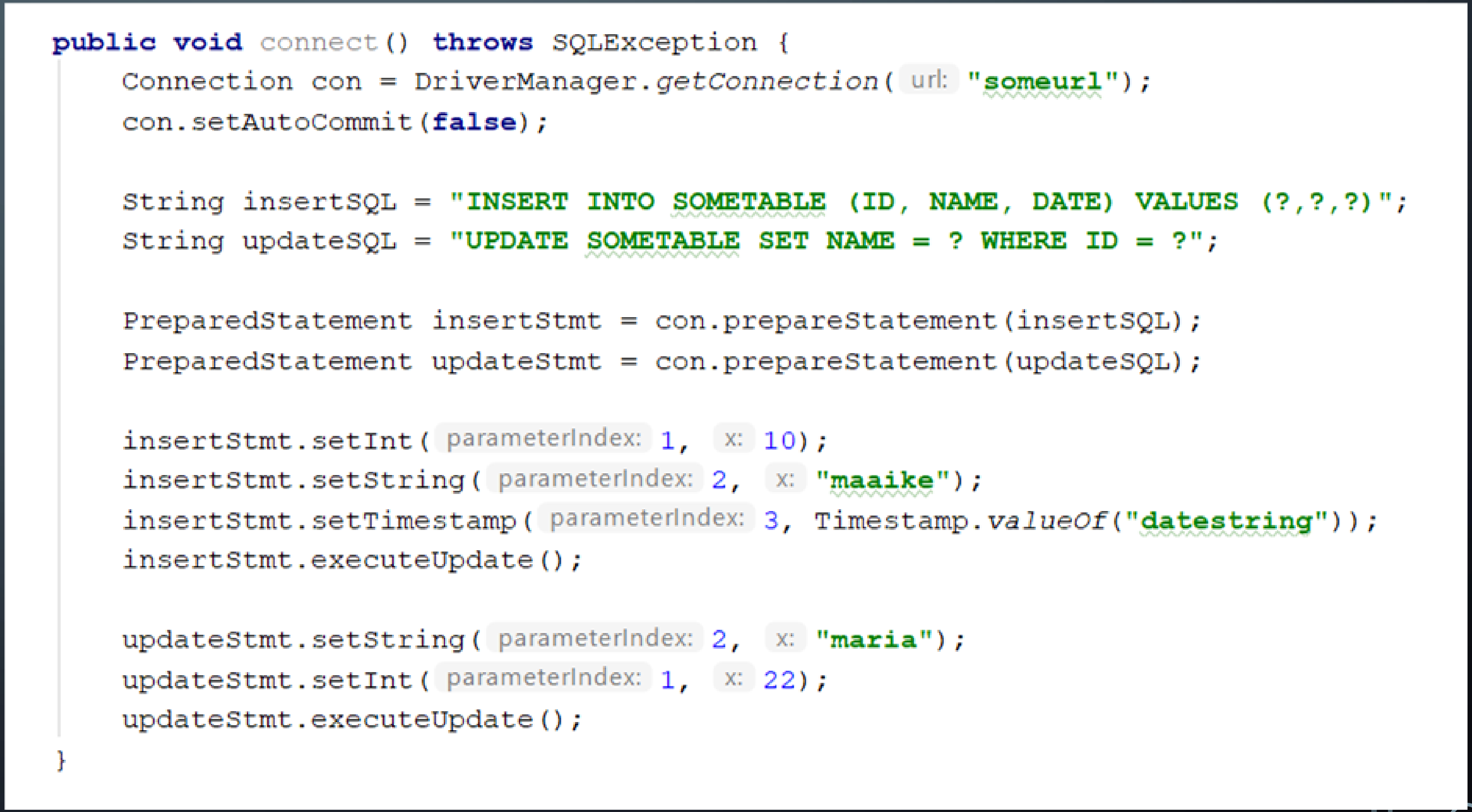


```
public void connect() throws SQLException {
    Connection con = DriverManager.getConnection( url: "someurl");
    String insertSQL = "INSERT INTO SOMETABLE (ID, NAME, DATE) VALUES (?,?,?)";
    String updateSQL = "UPDATE SOMETABLE SET NAME = ? WHERE ID = ?";

    PreparedStatement insertStmt = con.prepareStatement(insertSQL);
    PreparedStatement updateStmt = con.prepareStatement(updateSQL);

    insertStmt.setInt( parameterIndex: 1,  x: 10);
    insertStmt.setString( parameterIndex: 2,  x: "maaike");
    insertStmt.setTimestamp( parameterIndex: 3, Timestamp.valueOf("datestring"));
    insertStmt.executeUpdate();

    updateStmt.setString( parameterIndex: 2,  x: "maria");
    updateStmt.setInt( parameterIndex: 1,  x: 22);
    updateStmt.executeUpdate();
}
```

JDBC TRANSACTIONS

And with transaction handling

- And what will be the end state of the DB in this example?



```
public void connect() throws SQLException {
    Connection con = DriverManager.getConnection( url: "someurl");
    con.setAutoCommit(false);

    String insertSQL = "INSERT INTO SOMETABLE (ID, NAME, DATE) VALUES (?,?,?)";
    String updateSQL = "UPDATE SOMETABLE SET NAME = ? WHERE ID = ?";

    PreparedStatement insertStmt = con.prepareStatement(insertSQL);
    PreparedStatement updateStmt = con.prepareStatement(updateSQL);

    insertStmt.setInt( parameterIndex: 1,  x: 10);
    insertStmt.setString( parameterIndex: 2,  x: "maaike");
    insertStmt.setTimestamp( parameterIndex: 3, Timestamp.valueOf("datestring"));
    insertStmt.executeUpdate();

    updateStmt.setString( parameterIndex: 2,  x: "maria");
    updateStmt.setInt( parameterIndex: 1,  x: 22);
    updateStmt.executeUpdate();
}
```

TRANSACTIONS

Local and Global Transactions

- Local transactions: a transaction that takes place on one transactional resource like a JDBC connection or one message queue. Easy to implement transaction management, can be useful in a centralized system, where all components and resources reside in one place.

- Global transactions: a transaction that uses multiple transactional resources (e.g. multiple relational databases, message queues etc).

- Global transactions are harder for implementing transaction management than local transaction, but they are needed for a distributed system across multiple systems. Transaction management needs to be done across multiple systems so both local and global systems need to be managed.

SPRING TRANSACTIONS

Spring transaction abstraction

- Spring Boot detects spring-jdbc and h2 on the classpath.
- It will create a DataSource and a JdbcTemplate for you ready to use.
- Also a DataSourceTransactionManager will be created for you: this is the component that intercepts the @Transactional annotated method.

@TRANSACTIONAL

It has some flavours!

- @Transactional can be put on

- An interface (all methods within become transactional);
- a method of an interface;
- a class definition (all public methods within become transactional);
- a public method of a class. (You can safely put this on private methods, but it will simply be ignored)

- @Transactional(readOnly = true)
- This is not necessary, but does some optimizations under the hood for get calls (default is false).

- @Transactional(norollbackFor/rollbackFor Y)
- By default, a transaction will be rolling back on any RuntimeException and Errors but not on checked exceptions (business exceptions). This is how you can rollback on checked exceptions.

INTRODUCTION TO SPRING SECURITY

WHAT DID WE DO TODAY?

- RESTful services with spring
- Spring MVC
- Requests and responses
- Write your own services
- Introduction transaction management

POP QUIZ!

1. What is the core functionality of RESTControllers?

1. What is the difference between JPA, Hibernate and Spring Data JPA?

1. What is the difference between @Component and @Service?

1. When would you use @Cacheable?

QUESTIONS?

- Maaike.vanputten@brightboost.nl
- Or Whatsapp: +31683982426
- Don’t hesitate to contact me, I love to help!