Building the Friends Web Services

Peter van Rijn

www.little-world.nl

Add Section Header in Titlecase

Project Phases

Analysis

Design

Setup

Build

- Iterations

Wired Brain Friends

Wired Brain Coffee is starting a loyalty program called **Wired Brain Friends.** It is a friends database stored on a central server. The server should be accessible via a REST API.

Analysis

The REST API should be able to:

- register a new friend
- find one or more friends
- change a friend
- delete a friend

Design

MVC architecture

Model:

Friend

View:

Postman

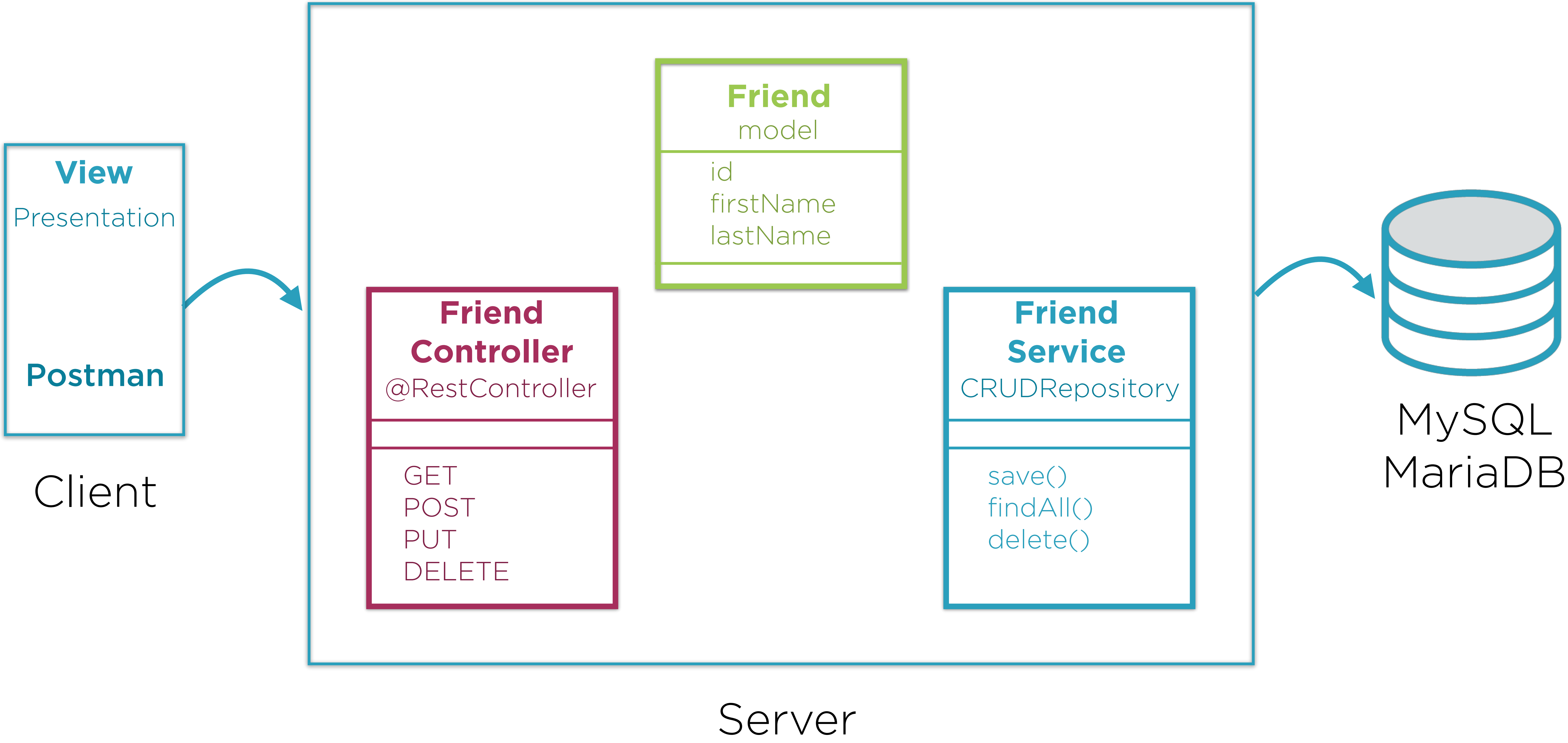
Controller:

FriendController with REST API

Service:

FriendService with CRUD

Wired Brain Friends Architecture



Setup

Install Database

Generate a Project at Spring Initializr

Configure Database

Install Database

MySQL

dev.mysql.com/downloads/mysql

MariaDB

downloads.mariadb.org

```xml
<dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.3.0</version>
</dependency>
```

Repairing the pom.xml

In Java 10 some enterprise libraries are not available anymore.

Add to the <dependencies> tag

```
spring.datasource.url=jdbc:mysql://localhost/friends
spring.datasource.username=root

spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.jpa.properties.hibernate.dialect=
                              org.hibernate.dialect.MySQL5Dialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

application.properties

The database configuration

Build Architecture

Implement the Architecture

model.Friend
service.FriendService
controller.FriendController

```
@Entity
public class Friend {

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private int id;

  private String firstName;
  private String lastName;
```

Friend Entity Class

The Friend class with an id, firstName and lastName

With getters and setters

```
public interface FriendService
        extends CrudRepository<Friend, Integer> {
}
```

FriendService DAO

The CRUDRepository has all the method we need

save()

findAll()

delete()

```
@RestController
public class FriendController {

  @Autowired
  FriendService friendService;

  // the URL mappings here
}
```

FriendController

The FriendController will contain the URL mappings

And is wired to the FriendService

This is dependency injection managed by the Spring container

Build Iterations

Implement the REST API one by one

POST	create
GET	read
PUT	update
DELETE	delete

```
@PostMapping("/friend")
Friend create(@RequestBody Friend friend) {
  return friendService.save(friend);
}
```

Create

Add a friend to the database.

And echo the friend including with a generated id.

```
@GetMapping("/friend")
Iterable<Friend> read() {
  return friendService.findAll();
}
```

Read

Read all the friends from the database.

And returns them

```
@PutMapping("/friend")
Friend update(@RequestBody Friend friend) {
  return friendService.save(friend);
}
```

Update

Update an existing friend in the database.

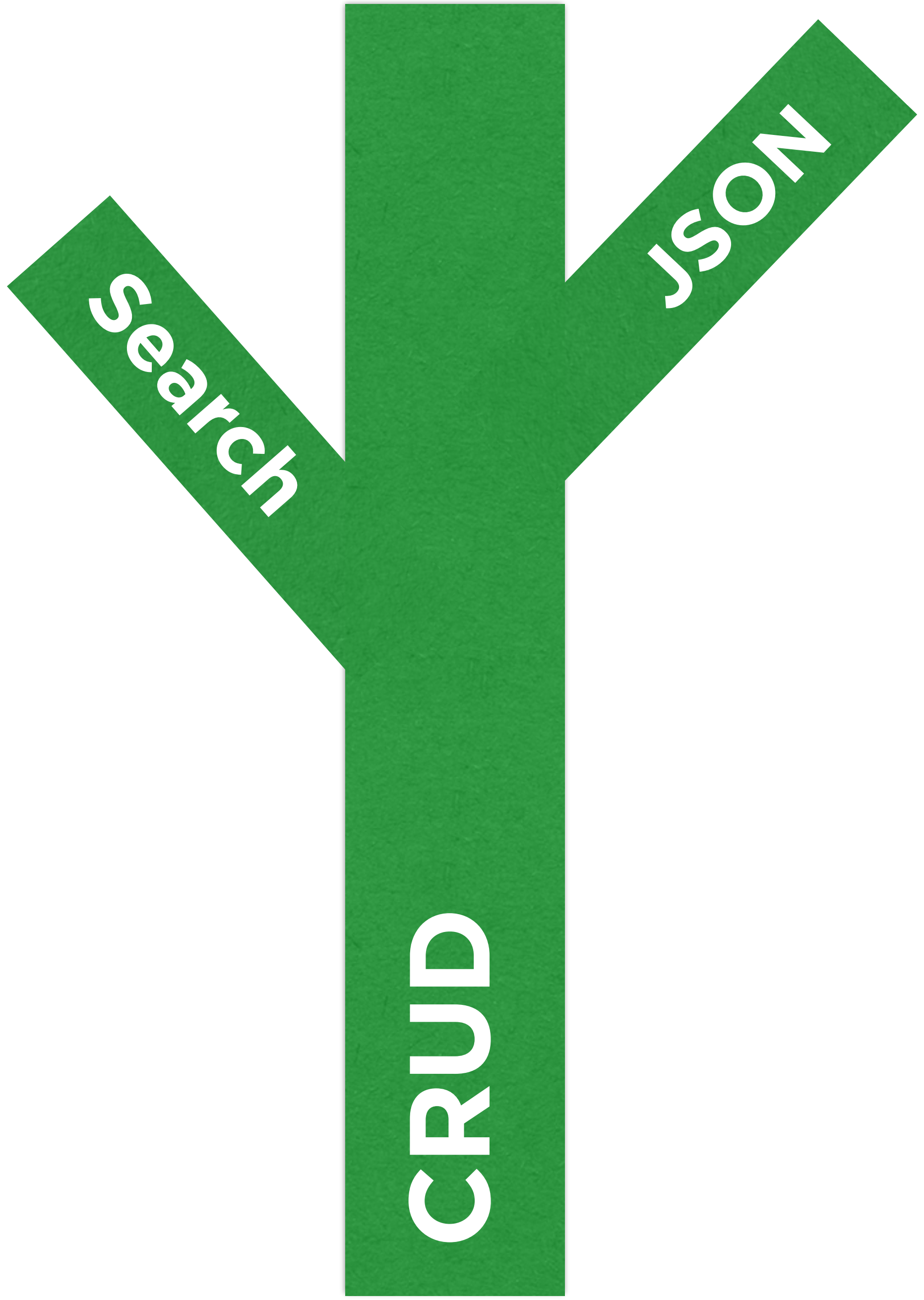
And echo the updated friend.

Save acts as an upsert function

```
@DeleteMapping("/friend/{id}")
void delete(@PathVariable Integer id) {
 friendService.deleteById(id);
}
```

Delete

Delete a friend from the database using an id.



CRUD

- The Basic Functionality

Search

- findBy ??

JSON

- Data Types
- Java Mapping

Search

Find by Id

Find by FirstName AND LastName

Find by FirstName OR LastName

```
@GetMapping("/friend/{id}")
Optional<Friend> findById(@PathVariable Integer id) {
  return friendService.findById(id);
}
```

findById

Uses a GET with a path variable

Returns zero or one friend using the id

```
public interface FriendService
              extends  CrudRepository<Friend, Integer> {

  Iterable<Friend> findByFirstNameAndLastName(
               String firstName,
               String lastName
   );
}
```

findByFirstNameAndLastName

To FriendService add method findByFirstNameAndLastName

There is no method body

That is generated by Spring Data

```
@GetMapping("/friend/search")
Iterable<Friend> findByQuery(
        @RequestParam("first") String firstName, @RequestParam("last")
        String lastName)
{ }
```

findByQuery

The URL Query contains the request parameters ‘first’ and ‘last’.

They are mapped to the firstName and lastName arguments.

JSON Mapping

JSON Types

Mapping JSON to Java

JSON annotations

Mapping Relations

- Embedded
- OneToMany

JSON Types

JavaScript Object Notation

String	“aa” or ‘aa’
Number	1 or 3.1
Boolean	true or false
List	[1, 2, 3]
Object	{“a”: 1, “b”:”yes”}
Null	null

JSON to Java Mapping

JSON	Java
"name": “John",	String name;
"age": 34,	int age;
"weight": 78.4,	double weight;
"married": true,	boolean married;
"address": { "street": "Park Lane 3”, "city": "Little Town" },	Address address;
"children": ["Mary", "Elisa"],	List<String> children;
"unused": null	Object unused = null;

JSON to Java Mapping

Annotation	Description
@JsonProperty("first")	property name
@JsonIgnore	ignore this property
@JsonIgnoreProperties	ignore these properties
@JsonInclude(JsonInclude.Include. NON_NULL)	exclude values: null, empty, default
@JsonManagedReference	parent-child relation
@JsonBackReference	child-parent relation

Demo

Rename Properties

Add Properties

Add Relation

- Address
  - One is @Embedded
  - More is @OneToMany

```
@JsonProperty("first-name")
private String firstName;
@JsonProperty("last-name")
private String lastName;
int age;
@JsonIgnore
boolean married;
```

Java Types and Annotations

Java types

- **int and boolean**

Annotations

- **@JsonProperty**
- **@JsonIgnore**

```
//in Friend
@JsonManagedReference
@OneToMany(mappedBy = "friend", cascade = CascadeType.ALL)
List<Address> addresses;

//in Address
@JsonBackReference
@ManyToOne
Friend friend;
```

@OneToMany with @ManyToOne

Using a backReference with a Foreign Key in the database

Add Json..Reference annotation otherwise a infinite loop

In Postman everything stays the same

Summary

Project

- Create, Read, Update and Delete
- Finders
    - FindBy ??
- JSON Mapping
    - Relations