WILD CODE SCHOOL

Create an API in 15 Minutes

Web API Development with Java and Spring Boot

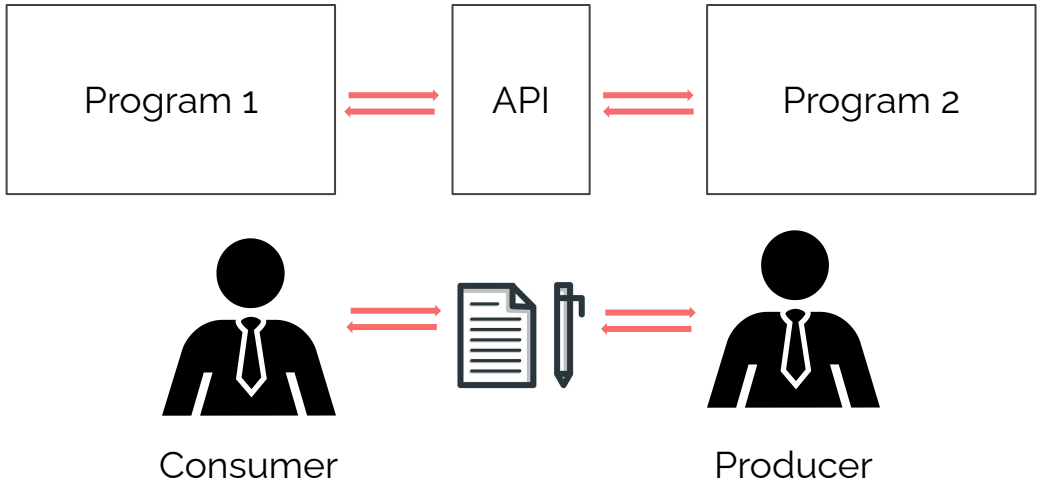
What is an API?

Why are APIs so important?

What is an API?

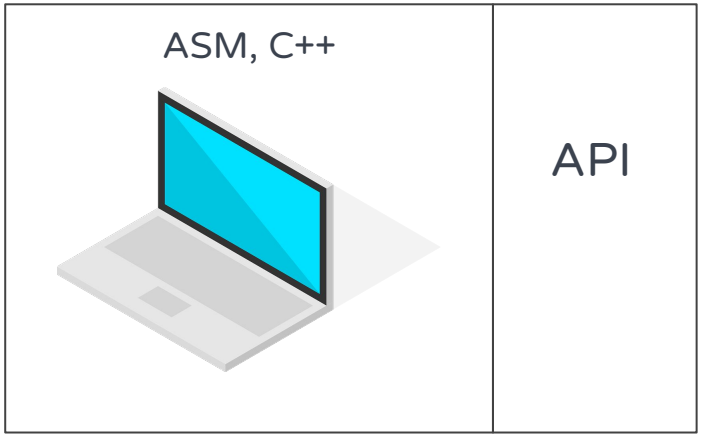
An *Application Programming Interface* or API is **an interface between two computer programs**.

It helps developers write programs that are more maintainable and can easily communicate with other pieces of software.



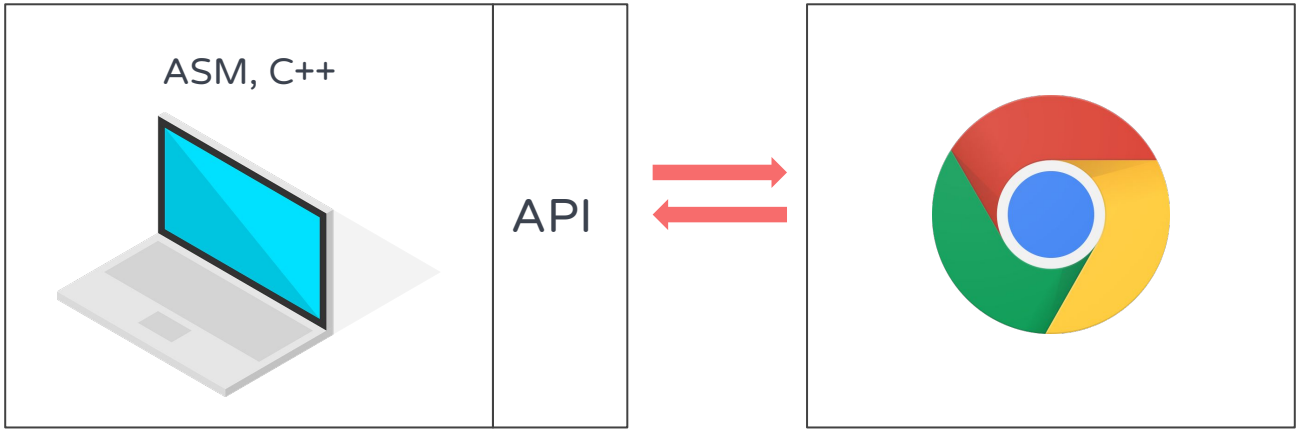
Examples of APIs

An operating system *exposes* several API's that **applications can use** to be able to access your computer's resources.



Examples of APIs

Chrome communicates with many of your operating system's API's to be able to access the internet, display stuff on the screen, get keyboard and mouse inputs, access the camera, the speakers, etc.



Web APIs for multi-client Web applications, M2M and IoT

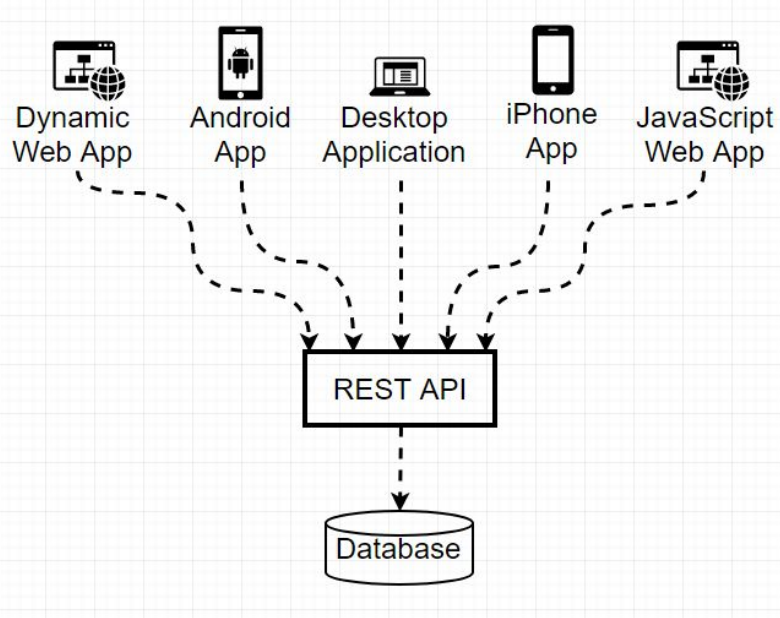
What is a Web API?

A Web API is the interface of a webapp with which an external program can communicate.

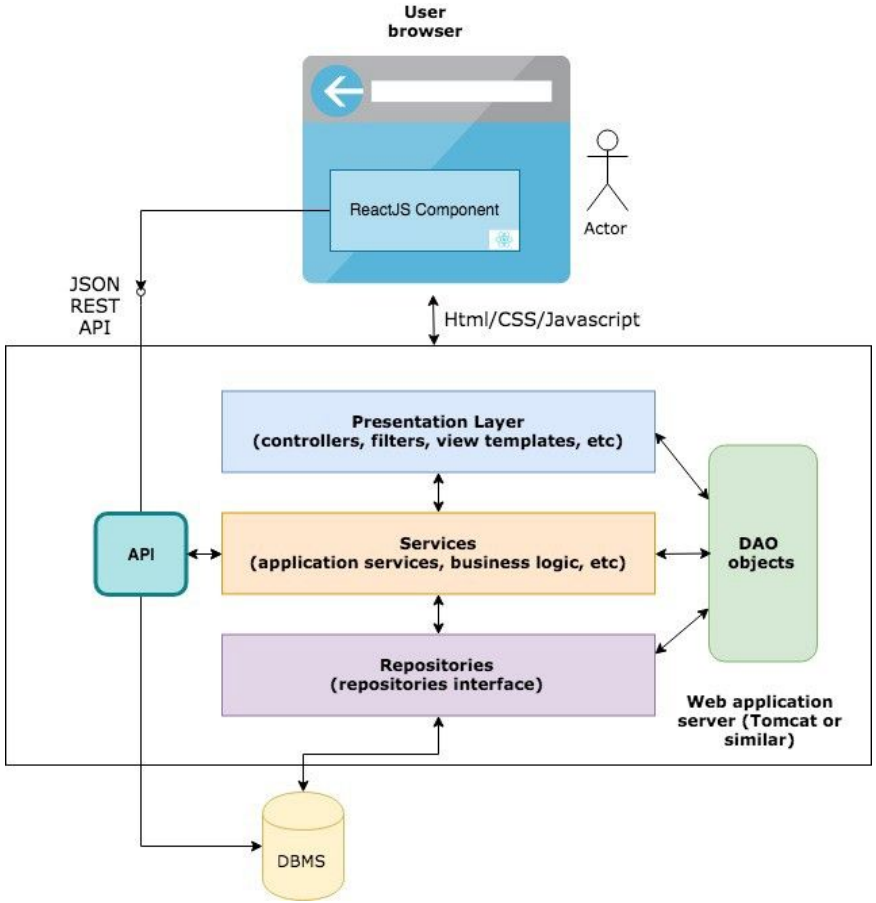
In short, and **most commonly**, **it's a server that accepts HTTP requests and responds with JSON.**

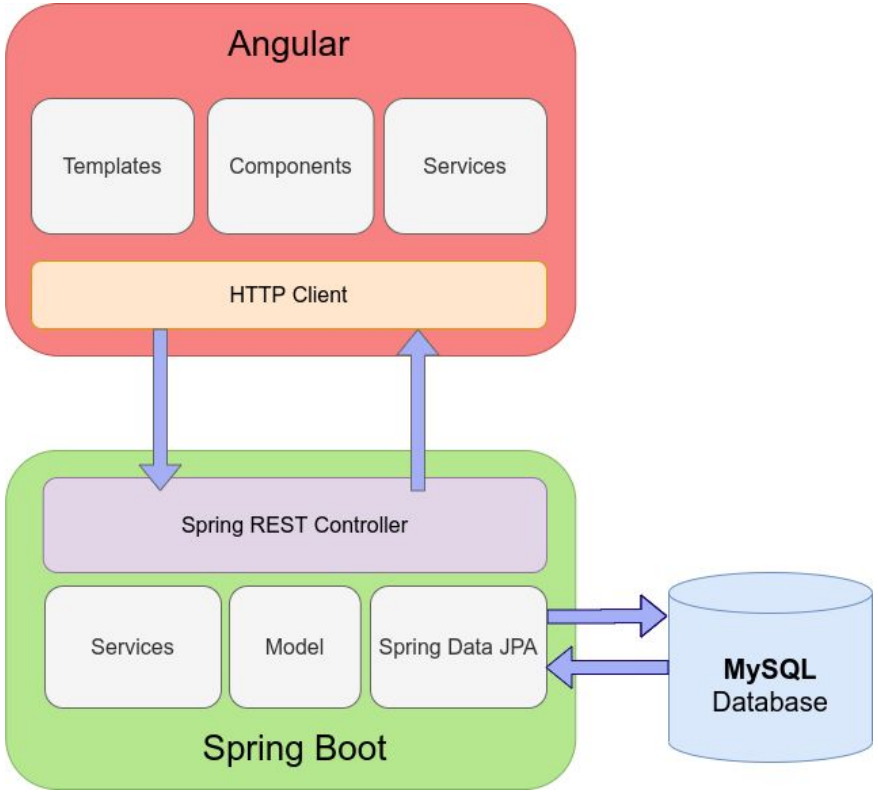
Why create a Web API?

An API makes our app/service **available to many *clients* over the internet**, either ours or 3rd parties'.



REST with Single Page Applications (SPA) vs. SSR






Web API
Development
with REST

What is REST?

REST is a **standard for designing web APIs** - RESTful APIs.

Alternatives to REST are SOAP and GraphQL.

It is mainly **a set of conventions and best practices** for building web interfaces.

REST describes the following (among other things):

- how to build **URIs for each resource**
- what **HTTP verbs to use** for each operation
- **how data is received and sent** in each resource

How to build the URI

Right
~~Wrong~~

Get a users list:
http://mywebsite.com**/users**
~~http://mywebsite.com/user~~
~~http://mywebsite.com/getAllUsers~~

Get all reviews of an user:
http://mywebsite.com**/users/321/reviews**
~~http://mywebsite.com/users/reviews/321~~

Get user info:
http://mywebsite.com**/users/56**
~~http://mywebsite.com/users/show/56~~

Get a user review:
http://mywebsite.com**/users/321/reviews/14**
~~http://mywebsite.com/users/321/review/14~~

How to build the URI

Filtered and **sorted list** of users:

http://mywebsite.com**/users?filter=client&sort=asc**

~~http://mywebsite.com/user/filter/client/sort/asc~~

Deconstructing the URI

HTTP methods

The 4 most common actions we perform using a web api are **create, read, update, delete - aka CRUD.**

In REST the corresponding HTTP methods are the following:

- **POST** - create
- **GET** - read
- **PUT** - update
- **DELETE** - delete

These 4 methods are **by far the most common** and they will usually be enough to build your API

HTTP methods

Create a user:
POST http://mywebsite.com/users
~~GET http://mywebsite.com/users/create~~

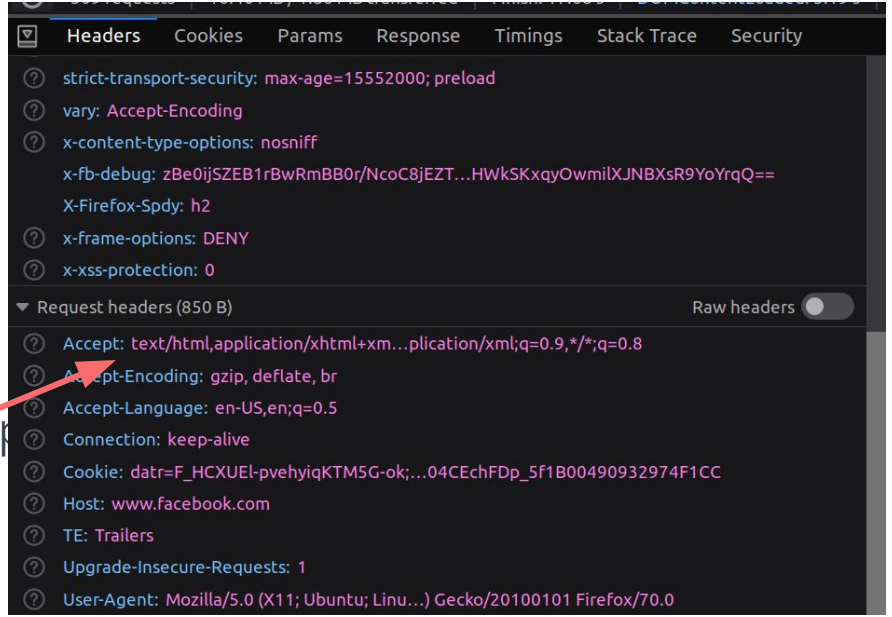
Edit a user:
PUT http://mywebsite.com/users/14
~~POST http://mywebsite.com/users/14/edit~~

Delete a user:
DELETE http://mywebsite.com/users/14
~~POST http://mywebsite.com/user/14/delete~~

HTTP Response Formats

The response **can come in many formats: HTML, XML, JSON, JSON-LD, CSV, etc.**

It is up to the client to define the format it accepts via the Accept header (it can accep multiple formats)



HTTP Response Codes

An **HTTP response always has an associated status code**. It tells the client if the request was successful, if there was an error, etc.

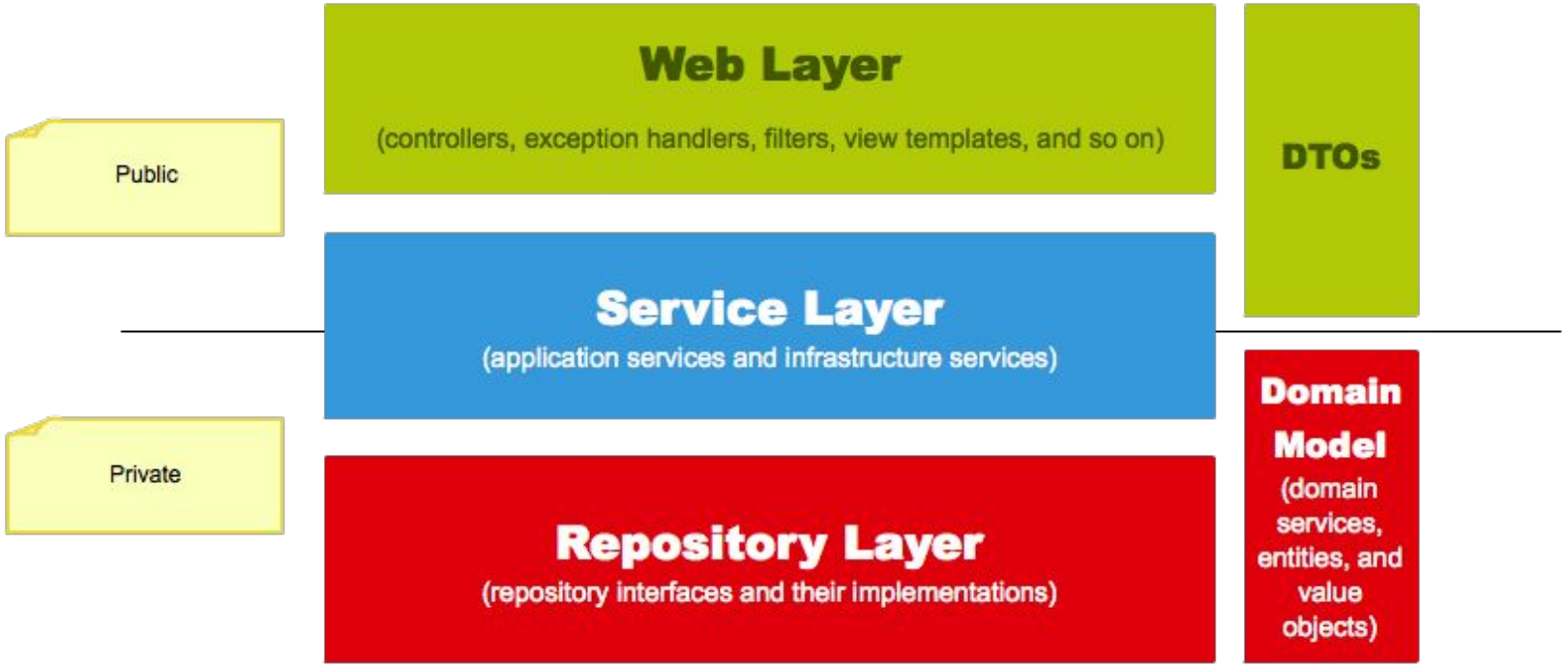
Status codes are **classified into families:**

- **1xx** - information
- **2xx** - success (e.g.: 200 OK, 201 created)
- **3xx** - redirect
- **4xx** - client side error (e.g.: 400 Bad request, 401 Unauthorized)
- **5xx** - server side error (e.g.: 500 internal server error)

HTTP Response Codes

When developing an API consult the list of http status codes and **make sure to always send relevant status codes** for each request:
https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

Layered Architecture for Web APIs



JPA

Standards & Specifications

JPA (Java Persistence API)

Set of Standards

Entities

Purpose: correspondence between the Java object and the SQL table

```java
@Entity
public class Question {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;

    public Long getId() {
        return id;
    }

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

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

}
```

eclipselink

OpenJPA

Hibernate

Repository (DAO)

Purpose: methods for CRUD operations

save()
findById()
findAll()
delete()

...

Spring Data JPA

Spring Data JPA

One repository to rule them all

Interfaces

CrudRepository

↓

PagingAndSortingRepository

↓

JpaRepository

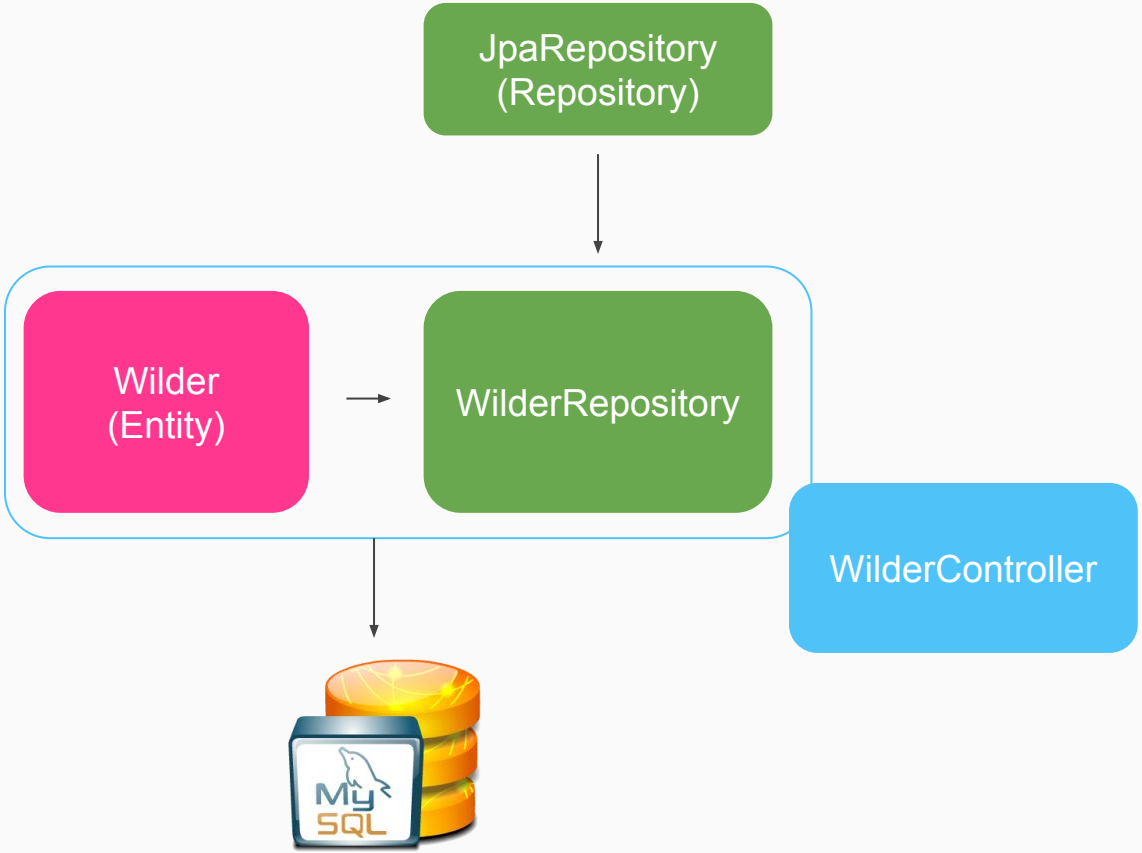
```java
public interface CrudRepository<T, ID extends Serializable>
  extends Repository<T, ID> {

  <S extends T> S save(S entity);      // (1)

  Optional<T> findById(ID primaryKey); // (2)

  Iterable<T> findAll();               // (3)

  long count();                        // (4)

  void delete(T entity);               // (5)

  boolean existsById(ID primaryKey);   // (6)

  // … more functionality omitted.
}
```

1. Saves the given entity.
2. Returns the entity identified by the given ID.
3. Returns all entities.
4. Returns the number of entities.
5. Deletes the given entity.
6. Indicates whether an entity with the given ID exists.

Spring Data JPA

Create an “API” in 15 minutes

Project Architecture



Spring Data JPA

Create an “API” in 15 minutes

2. Create Wilder Entity

```java
package com.wildcodeschool.jpademo.entities;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "wilders")
public class Wilder {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String email;
    private String category;

    public Wilder() {
    }

    public Wilder(String name, String email, String category) {
        this.name = name;
        this.email = email;
        this.category = category;
    }


    // Getters et setters...
```

Optional

Delegate generation of id to MySQL

The attributes will be automatically converted to MySQL table columns

Spring Data JPA

Create an “API” in 15 minutes

3. Create Repository

```java
package com.wildcodeschool.jpademo.repositories;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.wildcodeschool.jpademo.entities.Wilder;

@Repository
public interface WilderRepository extends JpaRepository<Wilder, Long> {

}
```

Which entity is managed by this repository?

What is the type of id (the primary key)?

Spring Data JPA

Create an “API” in 15 minutes

4. Create (REST-)Controller

```java
package com.wildcodeschool.jpademo.controllers;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RestController;
import com.wildcodeschool.jpademo.repositories.WilderRepository;

@RestController
public class WilderController {

    @Autowired
    WilderRepository wilderRepo;


}
```

We inject the repository we just created

Create RequestMapping on Controller for **create**

```
@RestController
public class WilderController {

      @Autowired private WilderRepository wilderRepository;

      @PostMapping("/wilders")
      @ResponseStatus(HttpStatus.CREATED)
      public Wilder createWilder(@RequestBody Wilder wilder)
      {
          return wilderRepository.save(wilder);
      }
}
```

Responds to a **POST** request like **http://localhost:8080/wilders**

Create RequestMappings on Controller for **read**

```
@GetMapping(value = "/wilders/{id}")
public Wilder getWilder(@PathVariable(“id”) Long id)
{
    return  wilderRepository.findById(id);
}
```

Responds to a **GET** request like **http://localhost:8080/wilders/1**

```
@GetMapping(value = "/wilders")
public List<Wilder> readAllWilders()
{
    return  wilderRepository.findAll();
}
```

Responds to a **GET** request like **http://localhost:8080/wilders**

Create RequestMapping on Controller for **update**

```
@PutMapping(value = "/wilders/{id}")
public Wilder updateWilder(@PathVariable(“id”) Long id, @RequestBody Wilder wilder) {
      Wilder storedWilder = wilderRepository.findById(id).get();
      // Update storedWilder
      return wilderRepository.save(storedWilder);
}
```

Responds to a **PUT** request like **http://localhost:8080/wilders/1**

Create RequestMapping on Controller for **delete**

```
@DeleteMapping("/wilders/{id}")
public Employee deleteWilder(@PathVariable(“id”) Long id)
{
    return  employeeRepository.deleteById(id);
}
```

Responds to a **DELETE** request like **http://localhost:8080/wilders/1**

Spring Data JPA

Create an “API” in 15 minutes

5. Bonus: Create custom methods

```
@Repository
public interface WilderRepository extends JpaRepository<Wilder, Long> {

    public Wilder findByName(String name);
    List<Wilder> findByEmailContaining(String email);

}
```

```
@RequestMapping("/getByName")
public Wilder getWilderByName(String name) {
    return wilderRepo.findByName(name);
}

@RequestMapping("/getByEmail")
public List<Wilder> getWilderByEmail(String email) {
    return wilderRepo.findByEmailContaining(email);
}
```

https://docs.spring.io/spring-data/data-jpa/docs/current/reference/html/#jpa.query-methods.query-creation

Spring Data JPA

Create an “API” in 15 minutes

5.1 **Testing** in the browser

5.2 **Verify** in Database

Testing an API

Test all your requests **using GUI programs** like:

- ➔ Postman - the most used
- ➔ Swagger
- ➔ Insomnia - postman's main competitor (lighter).

Or Text-based alternatives:

- ➔ REST Client - a VSCode extension to test routes.
- ➔ IntelliJ HTTP Client

They help you define the headers, body, access tokens, etc, **without having to write code.** You can even create documentation for your API automatically after you've tested the routes.