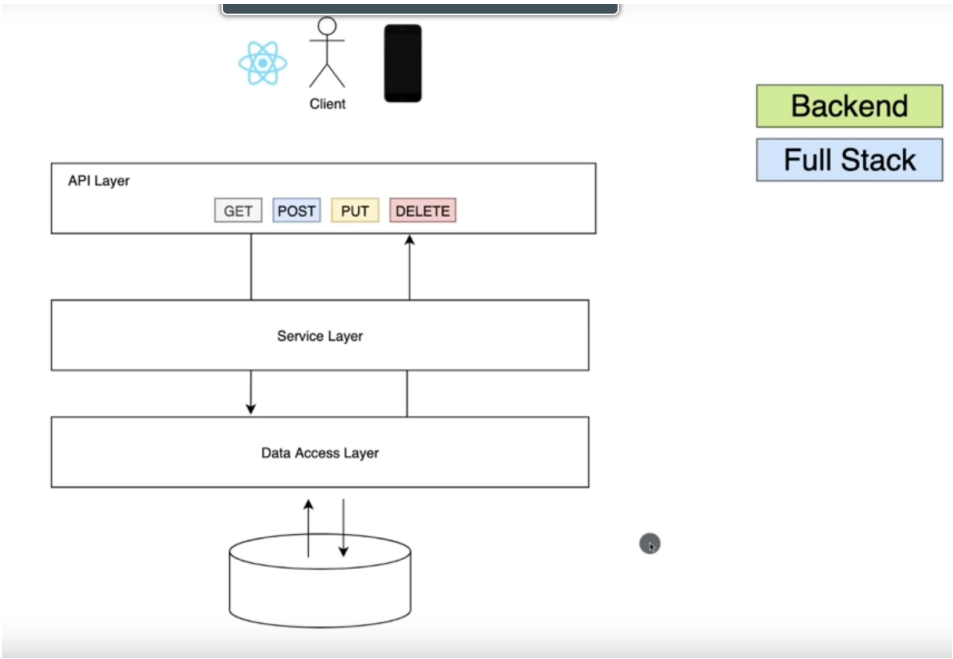
Spring Boot Application



> Learnt from https://www.youtube.com/watch?v=9SGDpanrc8U&t=3139s

Code Set-up

https://github.com/amigoscode/spring-data-jpa-course

- `pom.xml` : dependency
- `src/main/java/com.example.demo/DemoApplication.java` : backbone of code.
- `src/main/resources/application.properties` : configure environment properties
- `src/main/resources/static` : for web developments for example `html/css/js`

Create API

```
@SpringBootApplication
@RestController
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class)
    }

    @GetMapping
    public List<Student> hello() {
        return List.of(
            new Student(
                1L, "Kevin", "abc@gmail.com", LocalDate.of(2000, 12, 30), 21 //
1L since its Long type
            )
        )
    }
}
```

- `RestController` makes the rest code serving RESTful endpoints. Check on `localhost:8080`
- We so far put the routing inside main class. We will put them outside later.

> However, we eventually want to implement entire API Layer. so lets start with creating a class template

1. Create student class as our `model`
    - Create `src/main/java/com.example.demo/student/Student.java` relative to file `DemoApplication.java`

```
package com.exmaple.demo.student;

import java.time.LocalDate;

public class Student {
    private Long id;
    private String name,
    private LocalDate dob;
    private Integer age;

    public Student() { }
    // ctor
    public Student(Long id, String name, String email, LocalDate dob,
Integer age) {
        this.id = id;
        this.name = name;
        this.email = email;
        this.dob = dob;
        this.age = age;
    }
    // ctor without id since db generates it for us
    public Student(String name, String email, LocalDate dob, Integer
age) {
        ...
    }

    // getter + setter
    public Long getId() {
        return studentId;
    }
    public Long setId(Long id) {
        this.id = id;
    }

    ...

    public String toString() {
        return "Student{" +
                "id =" + id +
                ", name='" + name + '\'' +
                ", email='" + email + '\'' +
                ", age=" + age +
                '}';
    }
}
```

2. Implement proper APIs and implement "talks" between API Layer and Service Layer
    - Create `Controller` under `src/main/java/com.example.demo/student/StudentController.java` relative to

`Student class`

- Clear `@RestController` and `GetMapping` function inside the `DemoApplication` main function
- Put code inside `StudentController.java`

```
package com.amigoscode.demo.student;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

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

@RestController
@RequestMapping(path="api/students") //now localhost:8080/api/student
public class StudentController {
    // a reference to student service
    private final StudentService studentService;

    @Autowired
    public StudentController(StudentService studentService) {
        this.studentService = studentService;
        // above line wont work because we dont actually have a student
instance
        // unless you write this.student = new StudentService()
        // autowired means the above class variable studentserviced will
auto-instansiated and get injected into
        // the param of studentService in this function. We will also need
to tell the above class variable studentService
        // will need to be instantiated at some point, to do so add
@Component inside studentService.java
        // or you can add @Service which means we dont just want it to be a
regular component but actually a service
    }

    @GetMapping
    public List<Student> getStudents() {
        // before it returns a new Student object, now we use the service
        return studentService.getAllStudents(); // call function to get data
    }

}
```

- Create service layer by creating files at `src/main/java/com.example.demo/student/StudentService.java`

> We would like to put the

```
package com.amigoscode.demo.student;

import com.amigoscode.demo.EmailValidator;
import com.amigoscode.demo.exception.ApiRequestException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.util.StringUtils;
```

```
@Service
public class StudentService {

    public List<Student> getStudents() {
        return List.of(
            new Student(
                1L, "Kevin", "abc@gmail.com", LocalDate.of(2000, 12, 30), 21
// 1L since its Long type
            )
        )
    }
}
```

Connect to Databases - Data Access Layer

1. Add into `application.properties`

```
spring.datasource.url=jdbc:postgresql://localhost:5432/student
spring.datasource.username=
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=create-drop
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.properties.hibernate.format_sql=true
```

- username and password for local PostgreSql is empty

```
CREATE DATABASE student;
\du
GRANT ALL PRIVILEGES ON DATABASE "student" TO gabriel_role
\l
\c student
\d
```

2. **Map Student Class to JPA entity**

```
@Entity  // for hibernate
@Table   // for tables in db
public class Student {
    @Id
    @SequenceGenerator(
        name = "student_sequence",
        sequenceName = "student_sequence",
        allocationSize = 1
    )
    @GeneratedValue(
        strategy = GenerationType.SEQUENCE,
        generator = "student_sequence"
    )
    private Long id;
    private String name,
    ...
}
```

- If we run our application, we will see `Hibernate: create table student (...)` meaning table is created.

3. Implement Data Access Layer to access data (Repository is to access data)
   - Create class `StudentRepository` for working with JPA to access data inside database. `com.example.demo/student/StudentRepository.java`
   - Create the above file as `interface` (Controllers and Service are `class`)

```
package com.example.demo.student;
import org.springframework.data.jpa.repository.JpaRepository
// to use JpaRepository, put the type T and the type for primary key inside
<>
@Repository
public interface StudentRepository extends JpaRepository<Student, Long> {


}
```

   - Change `studentService` to use the repository instead of manually return a list of object

```
@Service
public class StudentService {

    private final studentRepository studentRepository; // reference

    @Autowired
    public StudentService(studentRepository studentRepository) {
        this.studentRepository = studentRepository;
    }

    List<Student> getAllStudents() {
        return studentRepository.findAll(); // available to us in the
JpaRepository interface
    }
}
```

4. To add some data into our table, create a new config file
   - Create `StudentConfig.java` as a class under `/student`

```
@Configuration
public class StudentConfig {
    @Bean
    CommandLineRunner commandLineRunner(StudentRepository repository) {
        return args -> {
           Student mariam = new Student( // no need to pass id
                ...
            );
            Student alex = new Student(
                ...
            );

            repository.saveAll(mariam, alex); // save to table
        }
    }
```

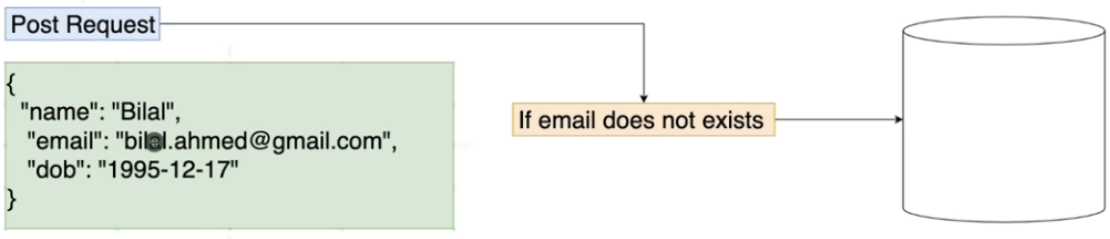
```
}
```

Implement 4 RESTful API requests

Before everything, I would not like to save `age` in table but would like to retrieve it by calculation

```
//inside student.java class
package com.example.demo.student;
import javax.persistence.* // if we change hibernete to sth else, ensure
everything still work
import java.time.LocalDate;

public class Student {
    ...
    private Long id;
    private String name,
    private LocalDate dob;

    @Transient // means no need for age to be a column in database, it will
calculated for us
    private Integer age;

    ...   // remove age from all params in ctor as well

    public Integer getAge() {
        return Period.between(this.dob, LocalDate.now()).getYears();
    }
}
```

1. Post Request to add students



   - Implement POST method inside `StudentController.java`

```
// new endpoint to post
@PostMapping
public void registerNewStudent(@RequestBody Student student) { // take
request body and map to student param
    studentService.addNewStudent(student); // call this method from service
}
```

   - Implement `addNewStudent` in `StudentService;java`

```
public void addNewStudent(Student student) {
    Optional<Student> studentOptional =
studentRepository.findStudentByEmail(student.getEmail());
    // save if student email not taken
    if (studentOptional.isPresent()) {
        throw new IllegalStateExeception("email token")
    }
    studentRepository.save(student);
}
```

- Create a function `findStudentByEmail` inside `repository` to be called in `service`

```
@Repository
public interface StudentRepository extends JpaRepository<Student, Long> {
    @Query("SELECT s FROM Student s WHERE s.email = ?1")
    Optional<Student> findStudentByEmail(String email);
}
```

- Add below line inside `application.properties` to see the error message you throw

  `server.error.include-message=always`

2. DELETE method to delete student by Id
   - Implement DELETE method inside `StudentController.java`

```
// new endpoint to delete
@DeleteMapping(path="{studentId}") // delete by putting id on path
public void deleteStudent(@PathVariable("studentID") Long id)) { // take id
from path variable
    studentService.deleteStudent(id); // call this method from service
}
```

   - Implement `deleteStudent` inside `service`

```
public void deleteStudent(Long studentId) {
    boolean exists = studentRepository.existsById(studentId);
    if (!exists) {
        throw new IllegalStateException(
            "id does not exist"
        )
    }
    studentRepository.deleteById(studentId)
}
```

3. PUT method to change student `name` and `email`

> We this time will use `@Transactional` to implement PUT method which makes us no need to write JPQL query. It allows us to use the setters to update the entity when its possible

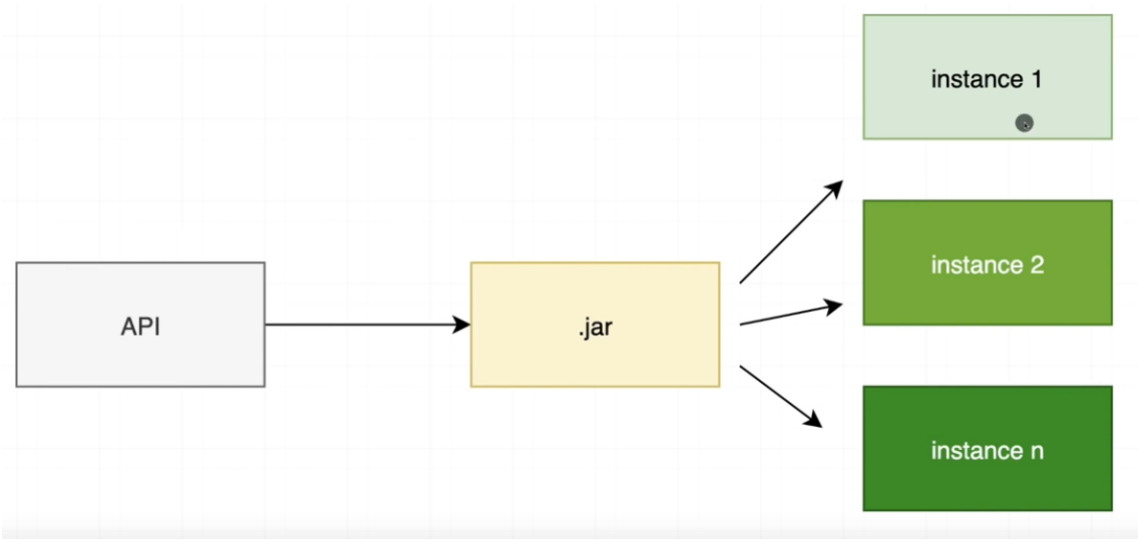
   - Implement PUT method inside `StudentController.java`

```
@PutMapping(path-"{studentId}")
public void updateStudent(
    @PathVariable("studentID") Long studentId),
    @RequestParam(required = false) string email),
    @RequestParam(required = false) string name)
    ) {
    studentService.updateStudent(studentId, name, email);
}
```

- Use `@Transactional` to implement `Service`

```
@Transactional // the entity goes into a managed state
public void updateStudent(Long studentId, String email, String name) {
    Student student = studentRepository.findById(sutdentId).orElseThrow(
        () -> new IllegalStateException("id does not exist");
    )
    if (name != null && name.length() > 0 &&
       !Object.equals(student.getName(), name)) {
       student.setName(name);
    }
    if (email != null && email.length() > 0 &&
       !Object.equals(student.getEmail(), email)) {
       // also email needs to be unique in db
       Optional<Student> studentOptional =
studentRepository.findStudentByEmail(email);
       if (studentOptional.present()) {
           throw new IllegalStateException("email taken");
       }
       student.setEmail(email);
    }
}
```

Packaging Application



- We would like to take application, package it as jar file and run on multiple instances

1. Clean `target` folder which gets generated after run. Clean it by clicking on `maven` then `clean`
2. Click on `maven` then `install` to get jar file

   We will then see `target/test-classes/demo-0.0.1-SNAPSHOT.jar`

Use Command line to go to target and then run:

`cd target`

`java -jar demo-0.0.1-SNAPSHOT.jar`

Now the application is up and running. You can test on localhost.

At this point we have run on one instance. To run on multiple, need to specify port.

Open a new terminal, and run `java -jar demo-0.0.1-SNAPSHOT.jar --server.port=8081`