Controller

- a collection of request handlers
- a handler behaves like a function
- handler gets input and produces output after some processing

URL components

localhost:8000/user/134?name=ram&age=25

localhost: domain name

8000: port

/user/134: route

?name=ram&age=25: query parameters; key-value pairs containing request specific data

Path variables

```
@GetMapping("/sqr/{x}")
public String(@PathVariable("x") int x) {
    return …
}
```

HTML Rendering

- Create a about page that can be accessed from home page.
- Create a /time controller that displays time in **h1** style
- Create a /greeting/Raju controller that displays
  - Hello Raju (in red font)

HTML Rendering

- Create a web page to display a list of usernames = [“Raju”, “Sujal”, “Acer”] in the following format:

1. Raju
2. Sujal
3. Acer

```
<ol>
    <li th:each=“e, itr : ${names}” th:text=“${e}”></li>
</ol>
```

We can use `${itr.count}` for current iteration value.

HTML Rendering

- Create a web page to display a table of product details including product name, price and stock quantity.

SN	Name	Price	Stock qty
1	Orange	100	5
2	Rice	80	50

```
<ol>
    <li th:each="e, itr : ${products}" th:text="${e.name}"></li>
</ol>
```

count, index, even, odd, first, last

Please download Postgres database server, pgAdmin

Gather data from form

```
<form action="/todo/new">
  <input type="text" name="userinput" />
  <input type="submit" value="SAVE" />
</form>
```

```
@GetMapping("/todo/new")
public String
  getFormData(@RequestParam("userinput")
  String input) {
     System.out.println(input);
     return "view";
  }
```

Delete a todo

```
@GetMapping("/todo/delete/{id}")
public String deleteTodo(int id, Model ..
  list.removeIf(todo -> todo.getId() == id);
  …
```

Dynamic href; how to?

```
<a th:href="@{/todo/delete/{id}(id=${todo.id})}"
  />Delete</a>
```

Database configuration

1. Add following dependencies:
   - Spring Data JPA
   - PostgreSQL Driver
2. Create a database and an user for your project
3. Add following to application.properties

```
spring.datasource.url=jdbc:postgresql://localhost:5432/todoapp
spring.datasource.username=todoapp
spring.datasource.password=todoapp
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.hibernate.ddl-auto=update
```

Entity = Table

1. Create a POJO class that represents a table
2. POJO = Plain Old Java Object
3. Annotate that class with @Entity
4. Annotate one field with @Id for PKey
5. Create EntityRepository interface

```
@Repository
public interface TodoRepository extends JpaRepository<Todo, PK> {
}
```

CRUD operations

Create:

1. Initialize the entity object
2. Set the field's values using corresponding setters.
3. Call `entityRepository.save()` to actually persist the entity to database.

Read:

1. Use `repository.findAll(), repository.findBy…()` methods

Update:

1. Get the entity to be updated using `repository.find…()` method
2. Set new values to desired field
3. Call `repository.save()`

Delete:

1. Get the entity to be updated using `repository.find…()` method
2. Call `repository.delete()`

GET vs POST

GET	POST
Data is encoded in the URL	Payload (large data) is located in the request body
Primarily used to get data/resources from the server	Primarily used to upload/save data to the server
No privacy of request data	High privacy of request data