

Spring Boot Training

BCA 2080 4th Semester

Nepathya College

Web Server

- Facilitates remote function calls
- Any device anywhere in the network can call functions of a particular device.
- Each function call is executed in that particular device, called the server; and the function returns some output back to the caller.
- In this way, data communication and operation on the data happens seamlessly without the requirement of architecture compatibility.

Each web request is a function call

- `getImage("cat.jpg")`
- `getVideo("cat.mpeg", 55, 480)`
- `storeMyInfo({ 'name': 'Acer', 'price': 55000 })`
- `/image/cat.jpg`
- `/profile/save?name=Acer&price=55000`
- `<item name="Acer" "price"="55000"></item>`

Function ↔ Route

```
@GetMapping("/hello")  
public String greet() {  
    return "Hello World!";  
}
```

This greet() function exists in the server. But we can execute it via browser using route `"/hello"`

URL routes

- /user/123 /user/{id}
- /blog/tilottama-mun /blog/{title}
- /image/cat.jpg /image/{filename}
- ```
@GetMapping("/user/{placeholder}")
public String getData(@PathVariable("placeholder") int
userID) {
 // method body
}
```

# HTML Form

```
@GetMapping("/register")
public String getRegisterPage() {
 return ""
 <form action="/save" method="get">
 <input type="text" name="fullname" />
 <input type="text" name="address" />
 <input type="submit" value="SUBMIT" />
 </form>"";
}
```

Create a **getRegisterPage()** method to return this form as string

URL: /save?fullname=Prakash&address=Tilottama

```
@GetMapping("/save")
public String saveRegisterPage(
 @RequestParam("fullname") String fullname,
 @RequestParam("address") String address) {
 System.out.println(fullname); //Prakash
 System.out.println(address); //Tilottama
}
```

# Personalization

- Currently the server is storing only a single shopping list.
- How to store shopping list for multiple users?
- Alice's shopping list must only be seen and modified by Alice. But Bob also has his own shopping list in the same server. Every user has their own personal shopping list.

# Personalization

- HTTP server doesn't remember previous requests.
- So each HTTP request must be uniquely identified by the server based on request data.
- HTTP client (user) should provide identification information in every request.
- Identification information should be unique to avoid misidentification.



# Improvements

- How to make sure User A cannot impersonate as User B?
  - Using un-guessable name

# HTTP Headers

- Contains metadata about message
- Contains info about server (HTTP response)
- Contains info about client (HTTP request)
- Format:

key: value

key: value

key: value

Set-Cookie: **cookie**name=value (sent by server)

Cookie: **cookie**name=value (sent by client)

# User Identification using Cookie

1. Server checks if **token** is present in cookie
2. If yes, go to 7.
3. Server generates a new **token**
4. Server stores the **token** in its database
5. Server sets that **token** to client as cookie
6. Go to 9.
7. Checks given **token** against its database.
8. If present, user is identified! (User is auth-ed)
9. Perform the identified user's request

# Task

- Write a method to generate a string of `n` length containing alphanumeric characters (A-Za-z0-9)
  - `getRandomString(20)` returns a random string of 20 characters

# Page Walkthrough

- When user lands on homepage
  - Check if user has a valid token
    - If yes; show his personal info
    - If no; redirect to **login** page
- When user lands on login page
  - Generate a new token and store it along with username in server database
  - Send the token as cookie

# Authentication

If username exists in db; then

If password matches; then

Allow user login

Else

Disallow user login

Else

Create new user with given info

# Chat Message System

- A user can send chat message to any other user in the system by username
- A user can view received chat messages
- A user can open chat box to chat with any user
- A user can broadcast a chat message so that every user can receive it
- PRO: user can send HTML formatted chats

# Database

Chat:

- fromUsername (String)
- toUsername (String)
- message (String)
- time (Datetime)
- seen (Boolean)

Chat 1

Chat 2

Chat 3

Chat 4



# POST method

| GET                                             | POST                                                    |
|-------------------------------------------------|---------------------------------------------------------|
| Data can be transferred only via URL parameters | Data can be transferred in HTML data body               |
| Used to transfer public facing data             | Used to transfer sensitive data                         |
| Supports only small string data                 | Supports large data such as complex forms, images, etc. |
| Used to request data from server                | Used to send data to the server                         |

# POST method

## HTTP Request Format

```
GET /profile/save?name=nepathya&address=Tilottama HTTP/1.1
```

```
Accept:
```

```
Connection:
```

```
Cookie:
```

```
POST /profile?id=12345 HTTP/1.1
```

```
Accept:
```

```
Connection:
```

```
Cookie:
```

```
<binary-data-body>
```

# POST method

## HTTP Request Pattern

GET /login HTTP/1.1

Accept:

Connection:

Cookie:

To get HTML page for login

POST /login HTTP/1.1

Accept:

Connection:

Cookie:

To send Login form data to server

username=nepathya&password=nepal123

# Controller Types

@RestController	@Controller
The mapping can return any kinds of data.	The mapping returns only a string (i.e. name of a html page in templates directory)
Used for API based servers	Used for server-side rendering
Javascript based frameworks can be used to interact with the server.	Server side rendering engines like Thymeleaf can be used.



# Thymeleaf

- Add the last line to `build.gradle` as follows:

```
dependencies {
 implementation 'org.springframework.boot:spring-boot-starter-web'
 testImplementation 'org.springframework.boot:spring-boot-starter-test'
 testRuntimeOnly 'org.junit.platform:junit-platform-launcher'
 implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
}
```

- Then **Gradle > Refresh Gradle Project**
- Now you can use Thymeleaf rendering in your spring boot project