Introduktion til Javalin

1. Opret IntelliJ Maven projekt (setup)
2. Navigation i web app (routing)
3. Eksempel (demo)
   - Routing
   - Send data ml server og browser
4. Koble database på web app

Første Javalin projekt

- I kan komme i gang med disse instruktioner

  Setup | Dat 2. semester

- I kan også komme hurtigt i gang ved at tage udgangspunkt i en Github repository template, f.eks. denne:

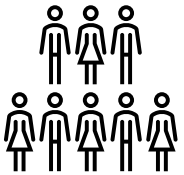
  Tine-m/javalin-template

Maven projekt

I Maven identificeres et projekt med:

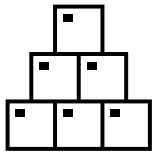
groupId

- identificerer **organisation/projektgruppe**
- skrives typisk som **omvendt domænenavn**
- bruges til at gruppere relaterede projekter



artifactId

- Navnet på **projektet**
- Typisk også navnet på **.jar-fil**, Maven bygger



De dependencies, man behøver, tilføjes i pom.xml

Sidebemærkning

Javalin opfordrer til logging i projektet

(vi koder ikke logging i web app første omgang)

```
###############################################################################
Javalin: It looks like you don't have a logger in your project.
The easiest way to fix this is to add 'slf4j-simple':

pom.xml:
<dependency>
   <groupId>org.slf4j</groupId>
   <artifactId>slf4j-simple</artifactId>
   <version>2.0.17</version>
</dependency>

build.gradle or build.gradle.kts:
implementation("org.slf4j:slf4j-simple:2.0.17")

Visit https://javalin.io/documentation#logging if you need more help
###############################################################################
SLF4J(W): No SLF4J providers were found.
SLF4J(W): Defaulting to no-operation (NOP) logger implementation
SLF4J(W): See https://www.slf4j.org/codes.html#noProviders for further details.
```

Sidebemærkning

Hvorfor er logging vigtigt?

Fejlfinding (debugging). Eks.

ERROR UserController - Failed to create user

java.sql.SQLException: connection refused

Overvågning af requests. Eks.

INFO - GET /posts 200

INFO - POST /login 401

Fejl i produktion

Når appen kører på server (fx Docker / Digital Ocean) er **logs ofte den eneste måde at se problemer på**.

Sidebemærkning

Logging-system vs. System.out.println

Hvad I er vant til

System.out.println | System.err.println

SLF4J

- Standardformat på loglinjer
  [timestamp] [level] [klasse] - [besked]

- Kan slås til og fra

- Kan være på flere niveauer, f.eks.
  ERROR, WARNING, INFO, DEBUG

Sidebemærkning

Log eksempel

2026-03-09 10:14:29 INFO app.controllers.UserController - User 'anna' logged in successfully

2026-03-09 10:14:29 = hvornår det skete

INFO = hvor alvorligt / vigtigt det er

app.controllers.UserController = hvor i koden det kommer fra

User 'anna' logged in successfully = hvad der skete

Sidebemærkning (men relevant nu!)

Vi definerer egen session konfiguration

I main overskriver vi Jetty's standard opsætning:

```
config.jetty.modifyServletContextHandler(handler ->
  handler.setSessionHandler(SessionConfig.sessionConfig()));
```

SessionConfig klassen:

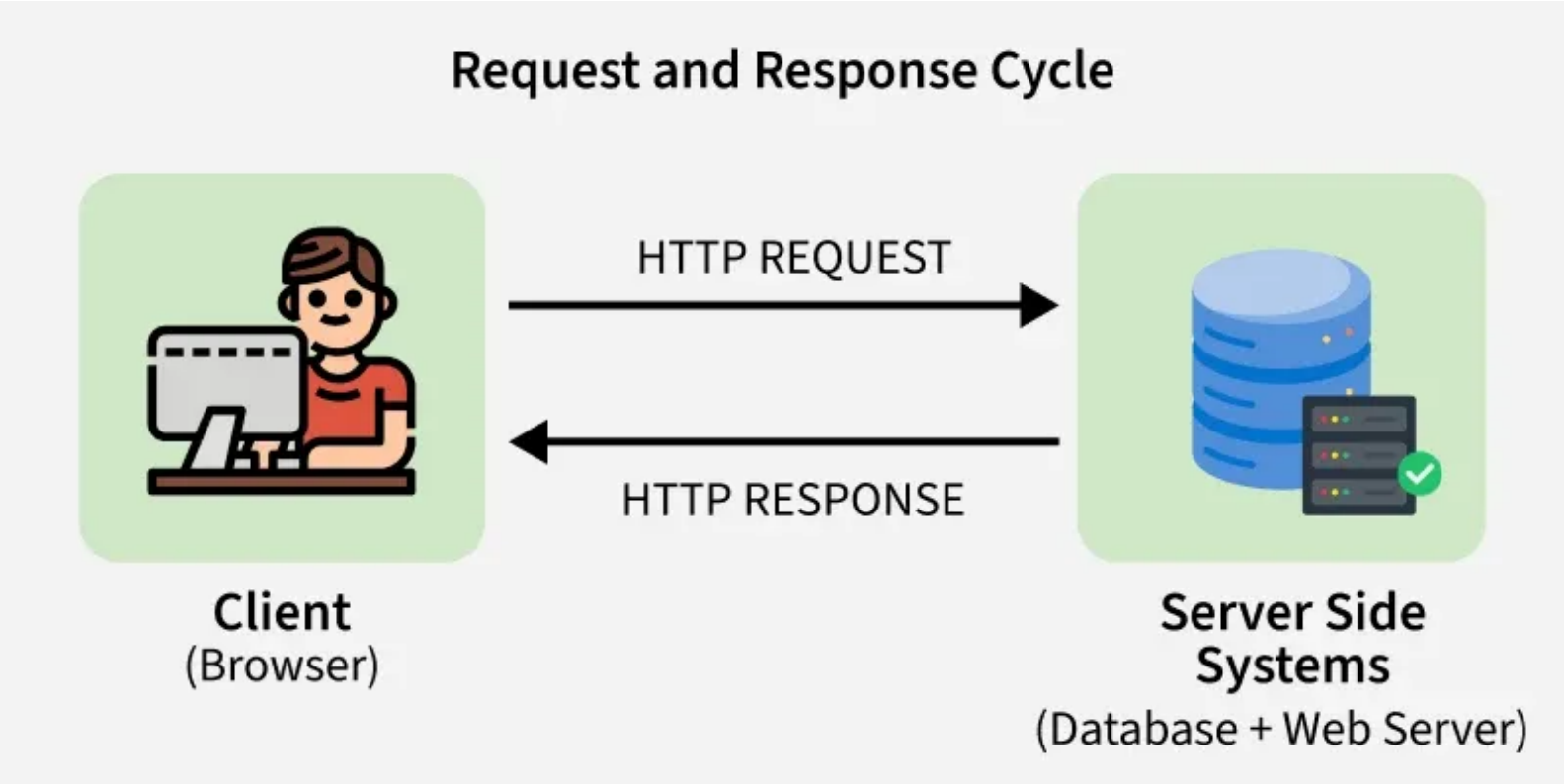
```
sessionHandler.setUsingCookies(true);
sessionHandler.setSessionTrackingModes(EnumSet.of(SessionTrackingMode.COOKIE));
sessionHandler.setHttpOnly(true);
```

Fordele

Sessions spores via cookies
URL-baserede sessions deaktiveres (fx /profile;jsessionid=ABC123)
Session cookie er HttpOnly (kan kun bruges af server, ikke JavaScript)

Hvorfor session-håndtering?

HTTP protokollen er stateless. Vi skal håndtere serverens manglende ”hukommelse” overfor klienten **mellem flere requests** (til login, logout, bruger session)



HTTP er STATELESS

– serveren glemmer ALT, så snart du har spurgt om noget!

Thymeleaf opsætning

Hvorfor Thymeleaf? **Dynamisk data** i HTML

```
<div th:if = "${session.user != null}">
   <span th:text = "${session.user.name}"></span>
</div>
```

I main:

```
config.fileRenderer(new
   JavalinThymeleaf(ThymeleafConfig.templateEngine()))
```

HTML sider

God idé at kopiere den medfølgende index.html pga. Thymeleaf opsætning:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Frontpage</title>
    …
</head>

…
</html>
```

Opsætning af endpoint

```
public static void main(String[] args) {
    Javalin app = Javalin.create(…
       ).start(7070);

    // Routing
    app.get("/", ctx ->  ctx.render("index.html"));
}
```

Forklaring

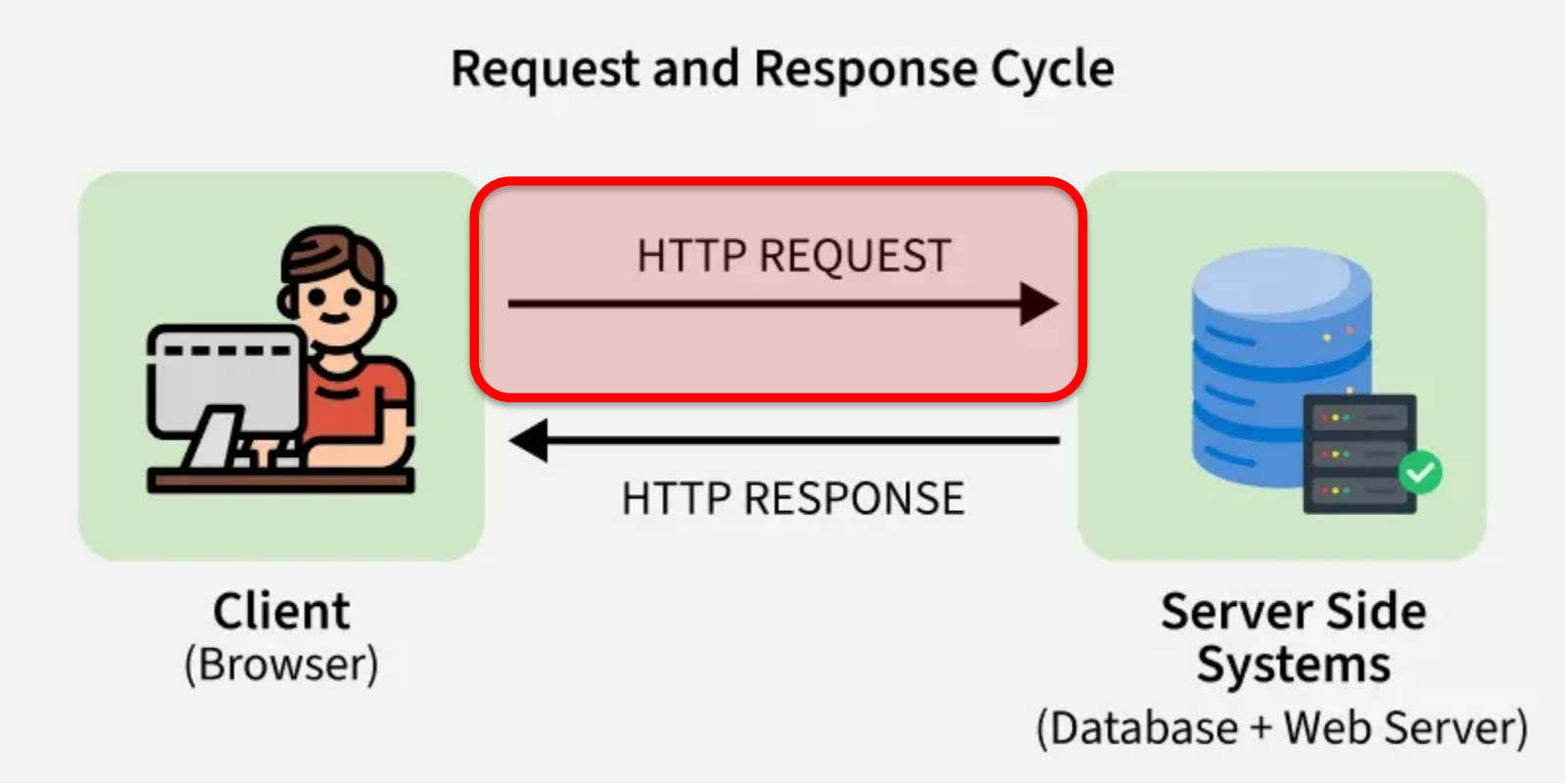
Vi starter server (localhost) på **port 7070**

Vi definerer **endpoint**, så server kan lytte efter request:

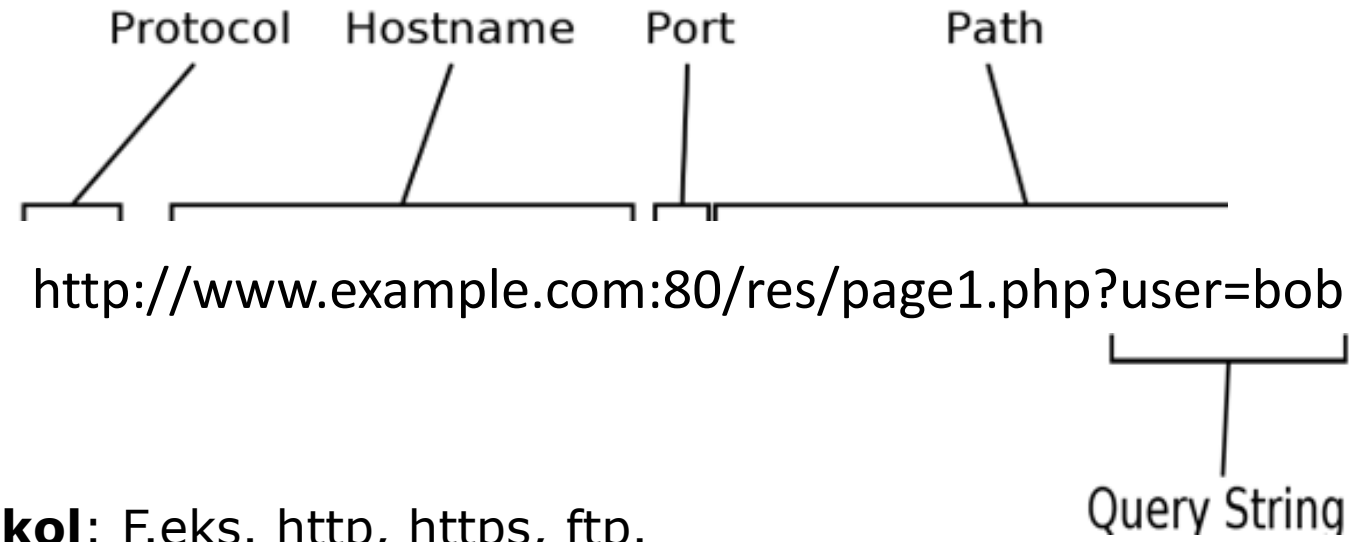
- **/** endpoint = roden af vores web app
- Kan kaldes i browser: **http://localhost:7070/**

Routing

At koble en **URL + HTTP-metode** til noget kode på serveren.



URL = Uniform Resource Locator



Protokol: F.eks. http, https, ftp.
Hostnavn: Navn på server, hvor ressourcen er placeret. Et domænenavn eller en IP-adresse (f.eks. 192.168.0.1)

Port: Port på serveren, der skal bruges til at hente ressourcen. Standardporten for http er 80
Path: Angiver stien til ressourcen på serveren
Request parameter (query string): Angiver eventuelle parametre placeret efter et spørgsmålstegn (?) og kan indeholde flere nøgle-værdi-par adskilt af ampersand (&)

HTTP metoder

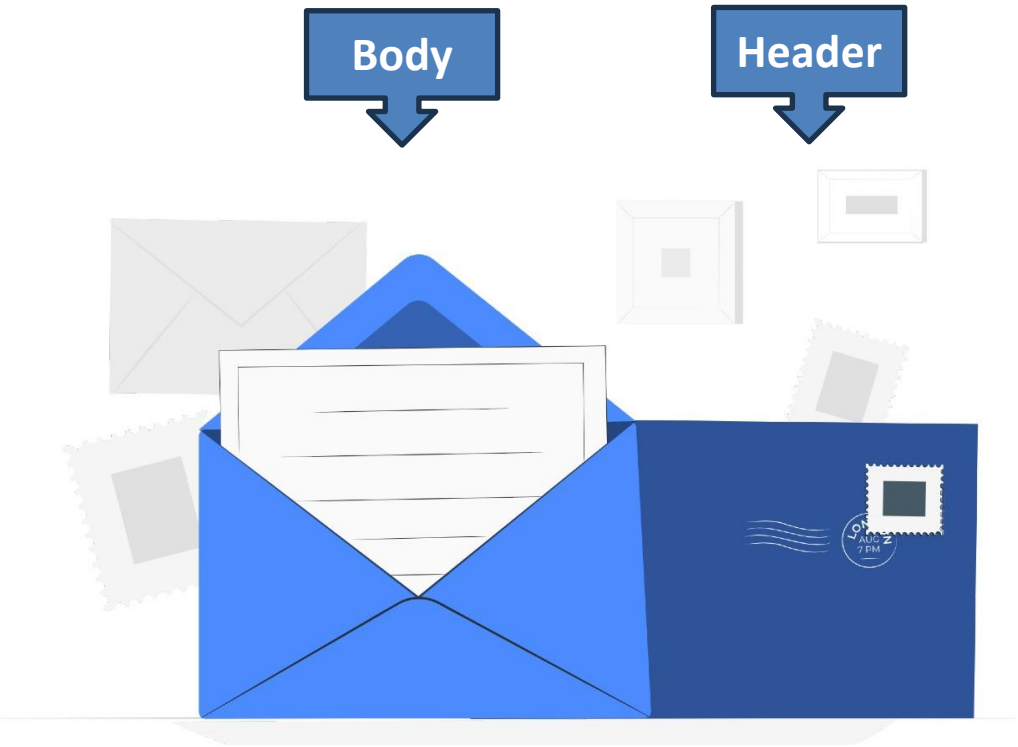
De mest almindelige

Beskriver **handling** klienten ønsker at udføre:

Metode	Formål
GET	Hente/læse
POST	Oprette ny
PUT	Erstatte eksisterende
DELETE	Slette

GET vs. POST

GET	POST
Værdier synlige i URL'en.	Værdier ikke synlige i URL'en.
Begrænsning på længden af værdierne, generelt 255 tegn.	Ingen begrænsning på længden af værdierne, da de sendes via HTTP body.
Understøtter kun strengdatatyper.	Understøtter forskellige datatyper, såsom streng, numerisk, binær osv.
Resultater kan bogmærkes.	Resultater kan ikke bogmærkes.
Parametre forbliver i webbrowserhistorikken.	Parametre gemmes ikke i webbrowserhistorikken.



Request header + Request body: ~ argument til metode (Java)
Response header + Response body: ~ returværdi fra metode (Java)

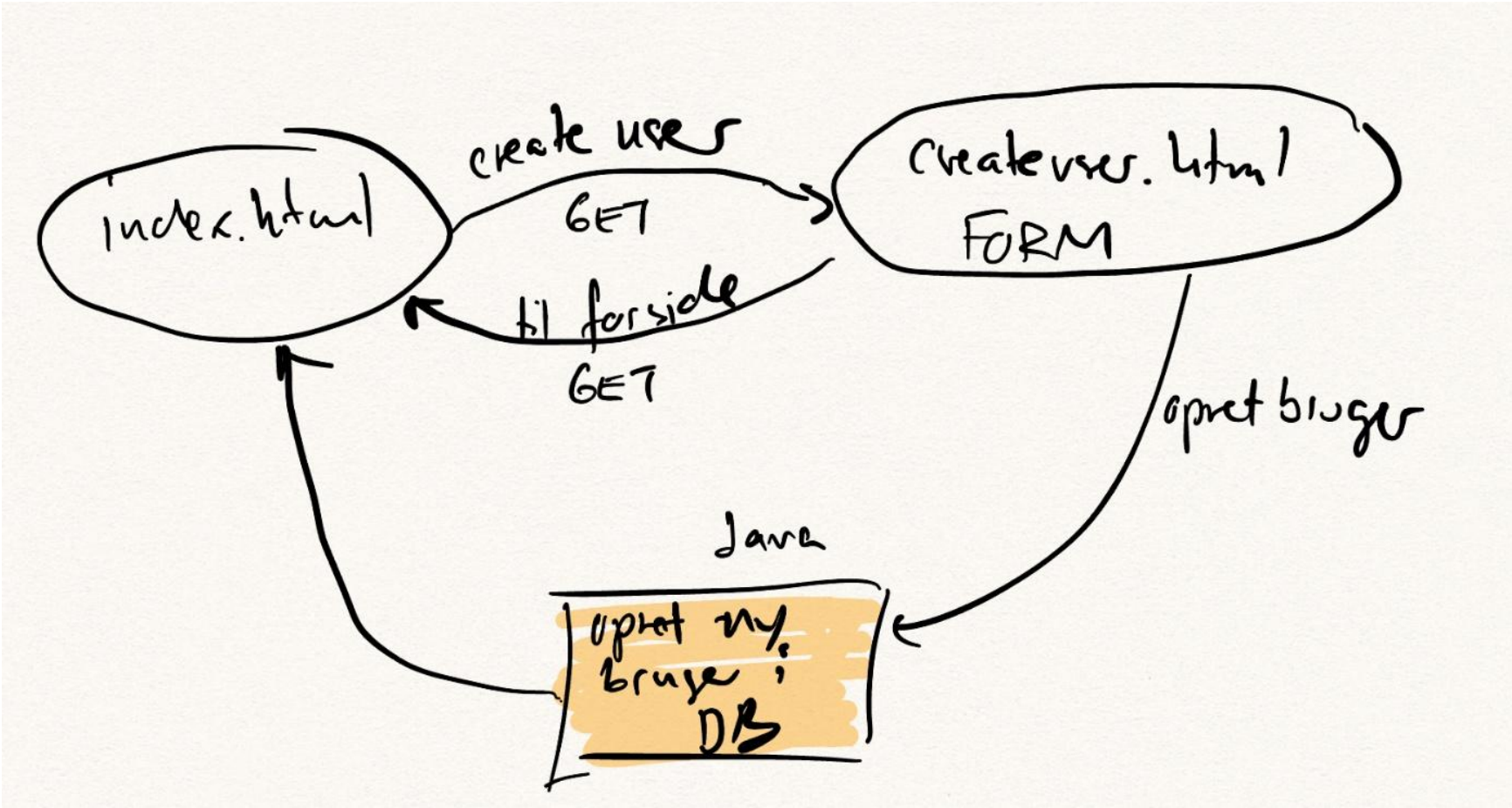
Routing

Vi skal sørge for to ting i vores backend kode:

1. Definere **route** (hvordan skal vi kalde på server)
   - Skal det være GET eller POST?
   - Hvad skal endpoint (path) hedde?

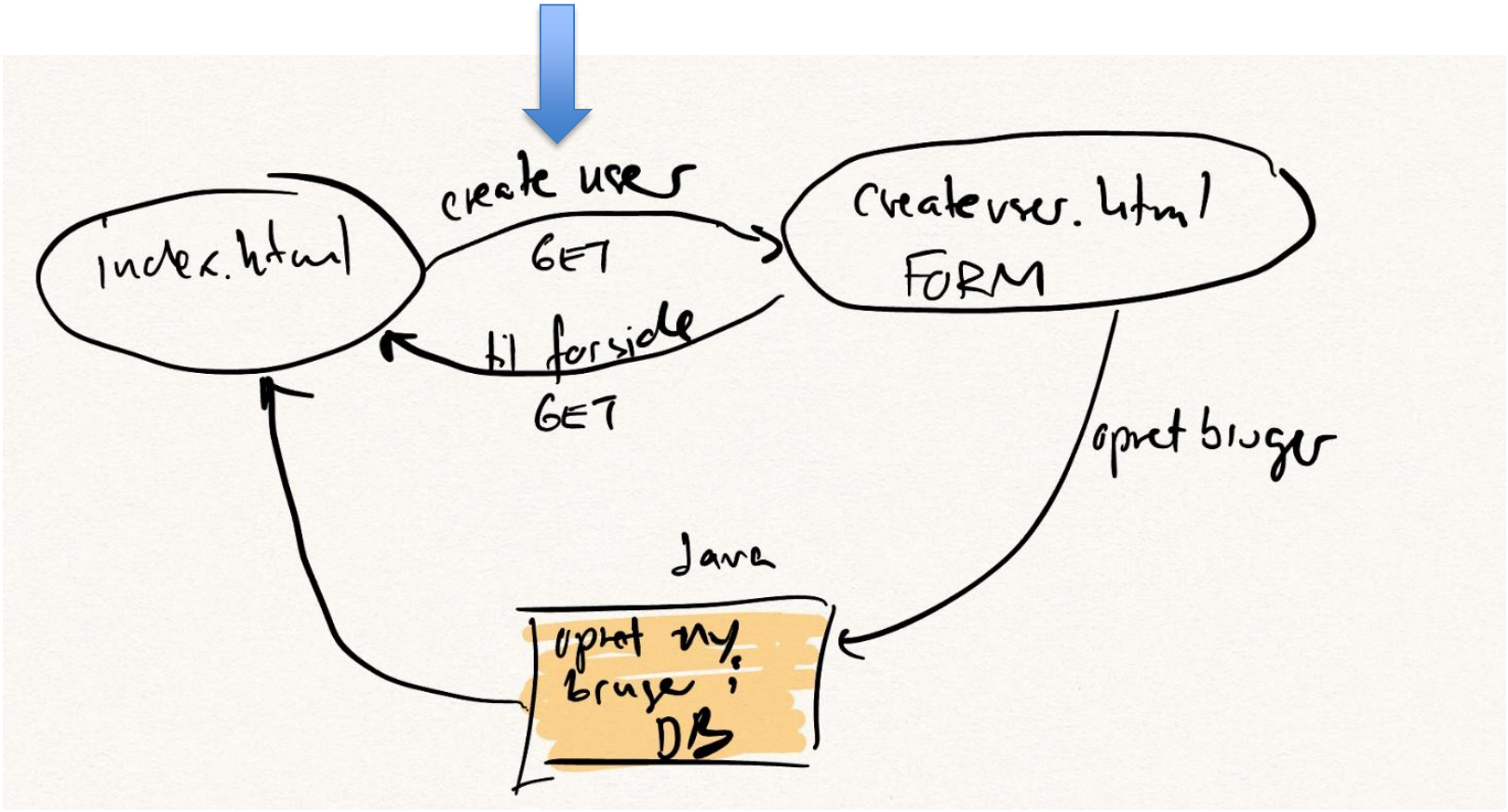
2. Opsætte **route handler** (hvad skal ske på server)
   - Hvad skal der kodemæssigt ske, når vi rammer et bestemt endpoint?

Eksempel
(UML state machine diagram)



Definere en route

#1: Vi skal fra index.html → createuser.html



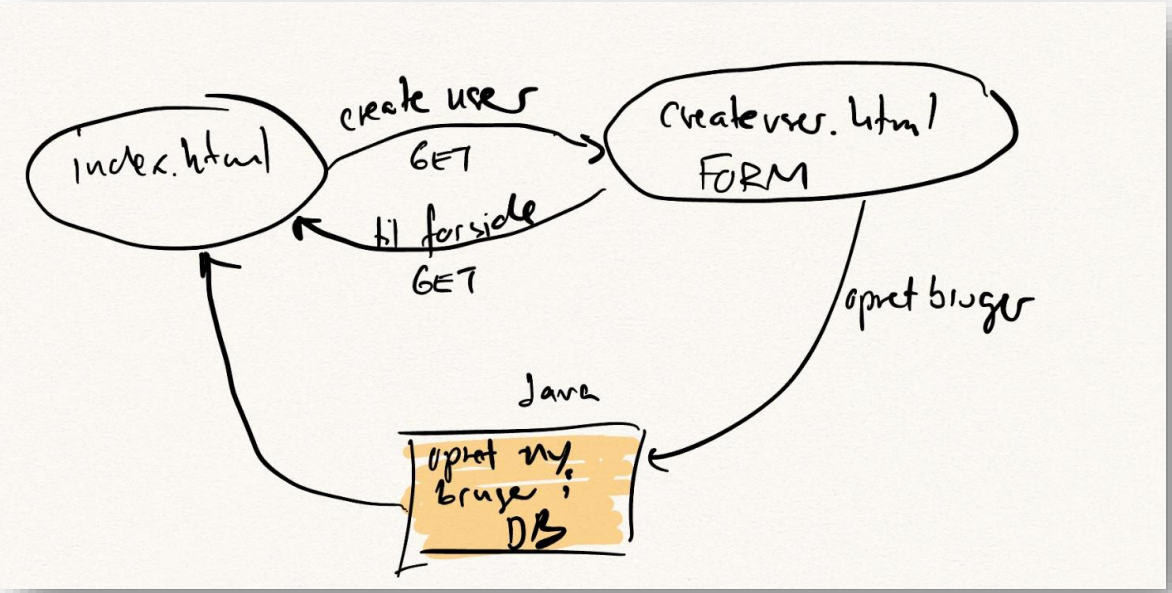
Hvordan gør vi?

Duer ikke at lave statisk reference i HTML , f.eks.

```
<a href="createuser.html"> Ny bruger </a>
```

Vi skal **definere route**, f.eks.

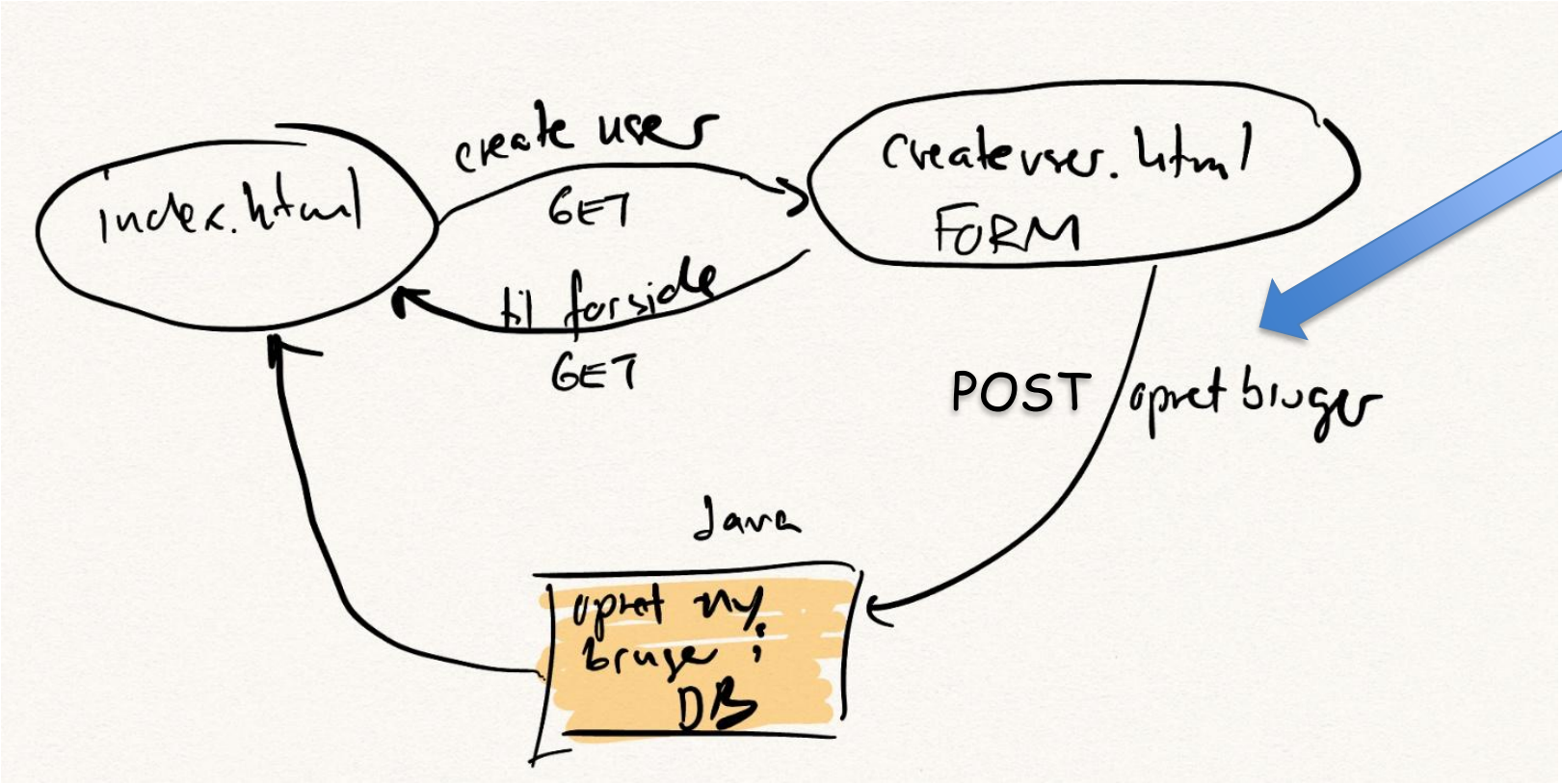
```
app.get("/createuser", ctx -> ctx.render("createuser.html"));
```



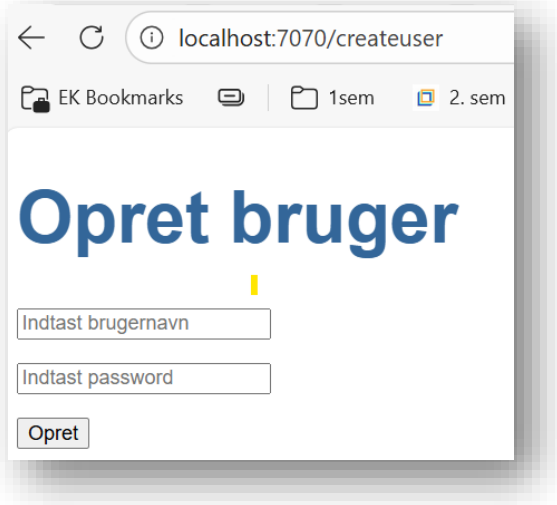
Kan kaldes fra HTML: ** Ny bruger**

Definere næste route

#2: Vi skal fra createuser.html → DB oprettelse



Data fra HTML til server

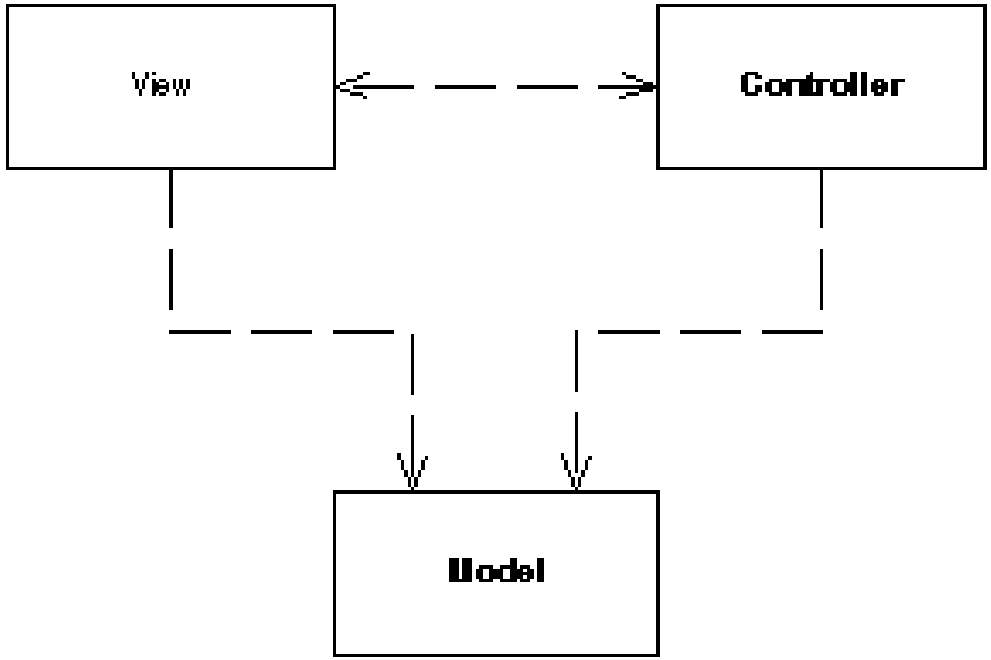


createuser.html med HTML form

```
<form method="post" action="/createuser">

    <p><input type="text" name="username" placeholder="Indtast
brugernavn"/></p>

    <p><input type="password" name="password" placeholder=
    "Indtast password"/></p>

    <p><button type="submit">Opret</button></p>

</form>
```

Route Handler klasse

Det bliver uoverskueligt at kode al routing i Main, så vi opretter ny klasse ***UserController.java*** i ***controllers*** package ud fra **MVC** pattern



```
public class UserController {

    public static void addRoutes(Javalin app) {
        app.get("/createuser", ctx -> ctx.render("createuser.html"));
        app.post(”/createuser", ctx -> createUser(ctx));
    }

    private static void createUser(Context ctx) {
        ctx.render("index.html");
    }
}
```

Main:

```
app.get("/", ctx ->  ctx.render("index.html"));

UserController.addRoutes(app);
```

Hvad skal handler kode gøre?

```
private static void createUser(Context ctx) {

    // hente data fra html form
    // oprette bruger i databasen
    // sende status tilbage til index.html

    ctx.render("index.html");
}
```

Hente data fra HTML form

Java

String name = **ctx.formParam**("**username**");
String password = **ctx.formParam**("**password**");

HTML:

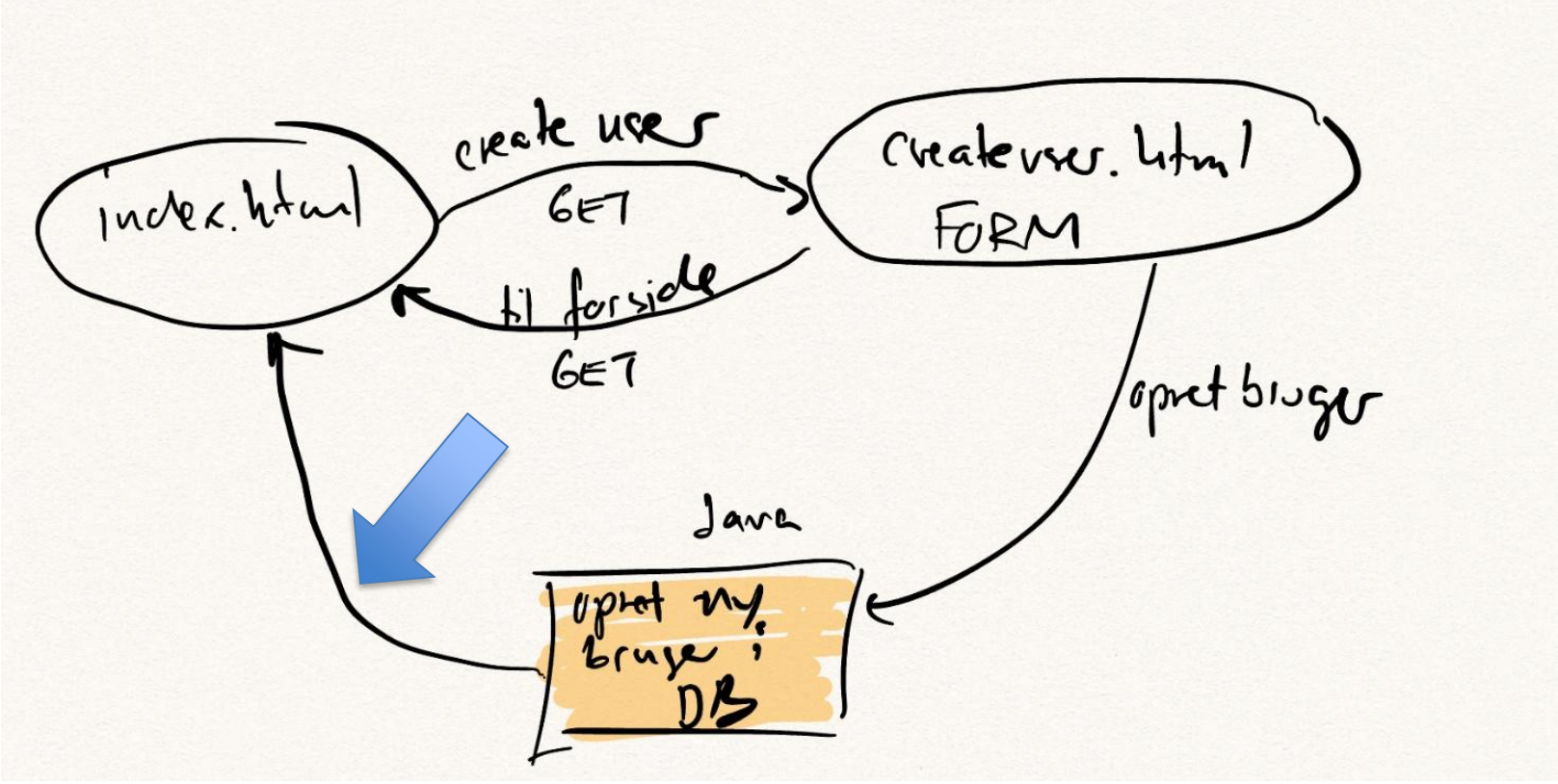
```
<form method="post" action="/createuser">

   <p><input type="text" name="username"/></p>

   <p><input type=”password" name="password"/></p>

   <p><button type="submit">Opret</button></p>

</form>
```

Oprette bruger i databasen

Vi bruger pseudokode og simulerer ☺

Data fra server til browser

#3: Vi skal sende status info til html om database-transaktion gik godt



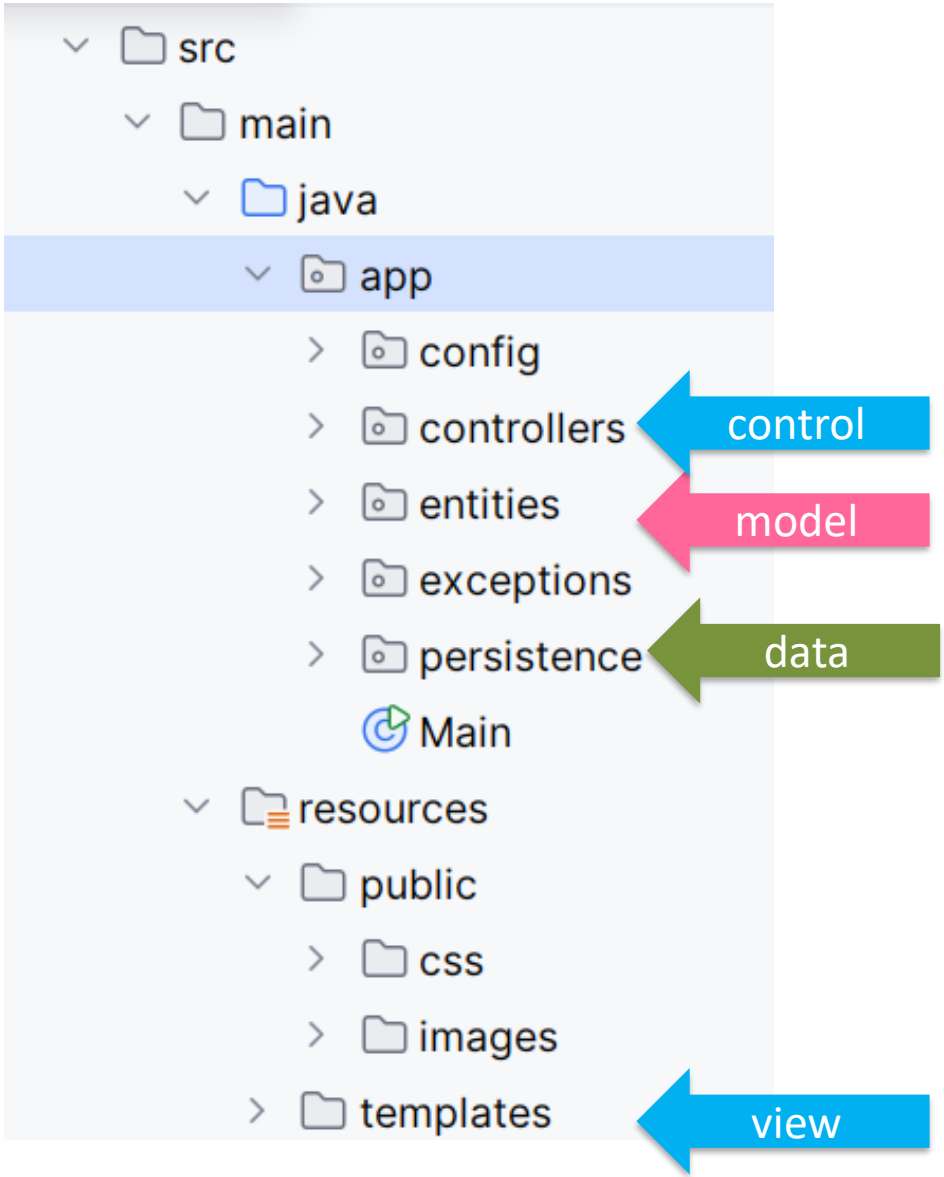
Data tilbage til HTML

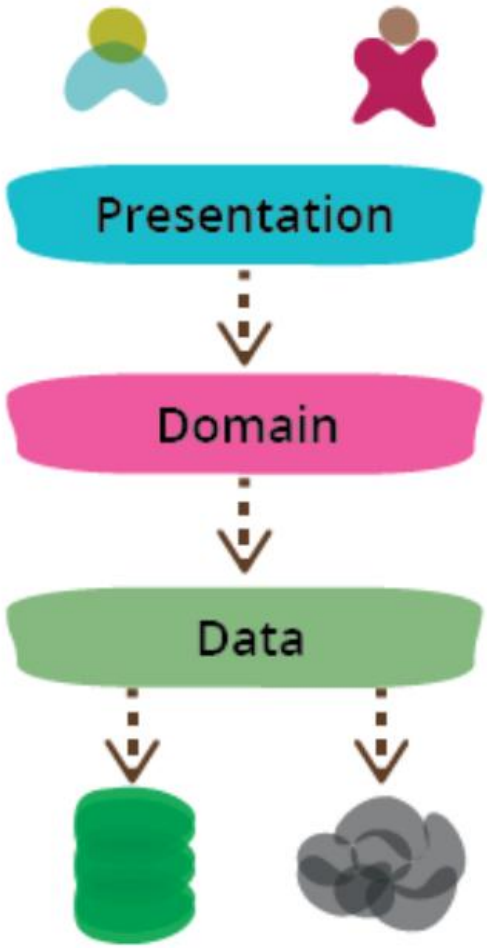
Java

ctx.attribute("**msg**", ”Bruger ikke oprettet!");

HTML

Vores arkitektur - 3 logiske lag





ConnectionPool klassen

Giver forbindelse til vores database

Brug den i første omgang som en black box fra en mapper klasse, dvs. du behøver ikke forstå den i detaljer. F.eks.

```
public class UserMapper {

  public void createUser(String userName,
            String password, ConnectionPool connectionPool)
            throws DatabaseException {
    String sql = "insert into users (username, password) values (?,?)";

    try (
          Connection connection = connectionPool.getConnection();
          PreparedStatement ps = connection.prepareStatement(sql)
        ) {
            ps.setString(1, userName);
            ps.setString(2, password);
            …
```