Informe de prácticas de ISDCM

Entrega proyecto entrega 2

Alumno/a: Junjie Li

Alumno/a: Moritz Höhne

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Content

Introduction

This report presents the development and architectural design of a modular, web-based video management application, developed as part of the ISDCM practices project at the Universitat Politècnica de Catalunya (UPC), BarcelonaTech, Faculty of Informatics. Authored by Junjie Li and Moritz Höhne, the document outlines the project's major components, including frontend interfaces, backend services, and data storage solutions.

The system is built following the **Model-View-Controller (MVC)** design pattern and is divided into three primary layers:

1. **Frontend** (webApp): A set of JSP pages that provide user interfaces for login, registration, video listing, playback, editing, and searching.

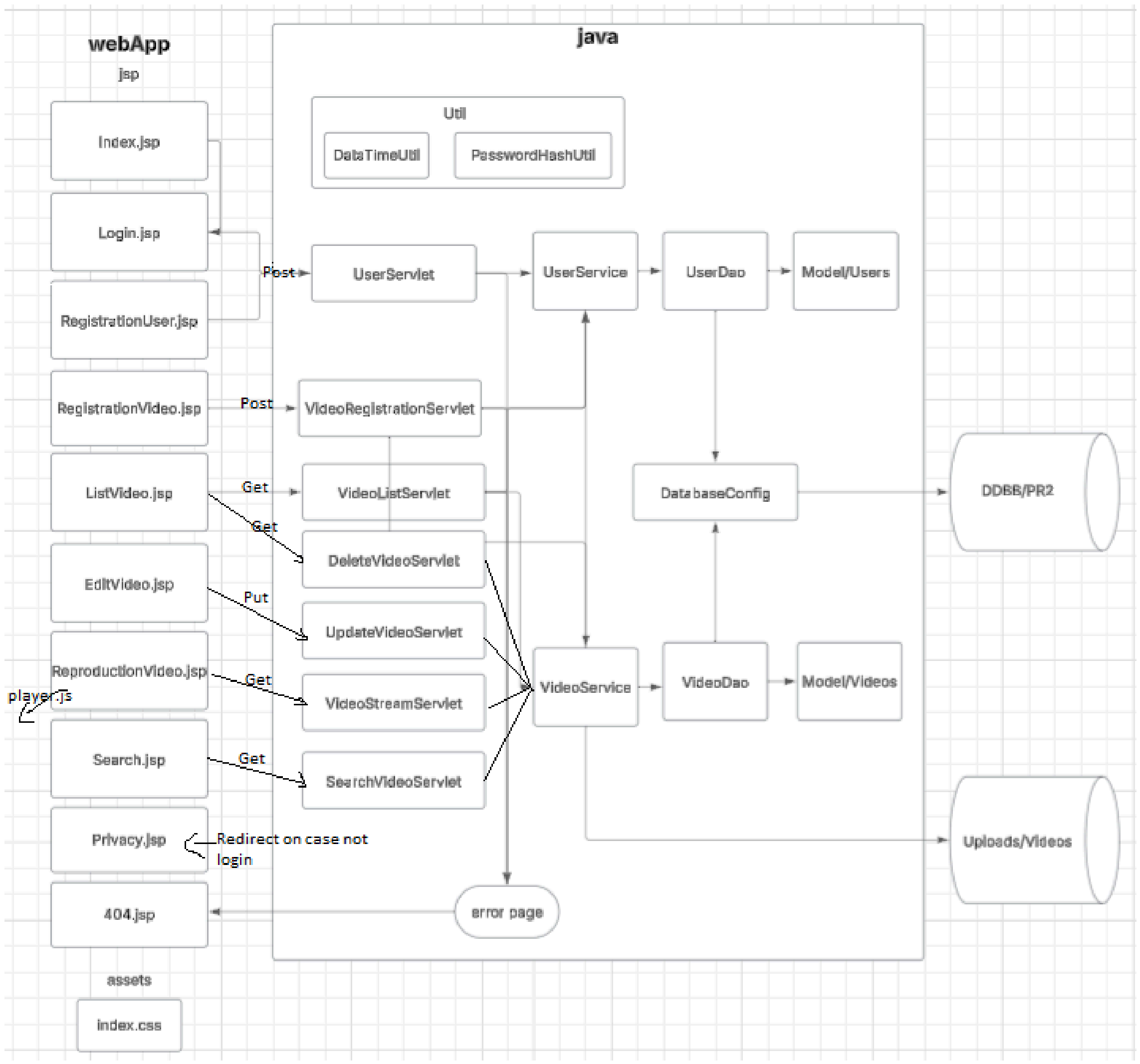
2. **Backend** (Java): Composed of utility classes, multiple servlets, service classes, and data access objects (DAOs) that handle business logic and database interaction.

3. **Storage**: Incorporates a relational database for user and video metadata (DDBB/PR2) and a file system for video uploads (Uploads/Videos).

The backend implements extensive functionality including **user login verification, and registration**, **video registration and deletion**, **video playback**, **editing**, and a **multi-mode search system**. The search feature supports querying videos by title, author, or date using GET requests and dynamic SQL `LIKE` queries for flexible filtering.

This report details how the various components interact within the system, emphasizes the modular and reusable nature of the architecture, and provides a foundation for future scalability and feature extension.

Design decisions



Architecture Overview

The architecture is divided into three main parts: `webApp` (frontend pages), `java` (backend logic and services), and the database/file storage. Here's a detailed breakdown of each part:

`webApp` (Frontend Pages)

JSP Pages:

- **Index.jsp**: The homepage, likely serving as the entry point of the application.
- **Login.jsp**: The login page, where users authenticate their identity.
- RegistrationUser.jsp: The user registration page, used to handle new user registration.

- **RegistrationVideo.jsp**: The video registration page, possibly used for uploading or registering video content.
- **ListVideo.jsp**: The video list page, displaying existing video content.
- **404.jsp**: The error page, handling 404 errors (page not found).
- **EditVideo.jsp**: This page is used to edit uploaded video content.
- **ReproductionVideo.jsp**: This page is used to display video playback.
- **Search.jsp**: This page is used to search for videos.
- **Privacy.jsp**: User privacy page.
- **Additional note**: These pages will be redirected to error pages or login pages if you are not logged in.

Assets:

- **index.css**: A stylesheet file used to define the styling for the homepage.

`java` (Backend Logic and Services)

Utility Classes (Util):

- **DataTimeUtil**: A utility class for handling date and time, likely used for formatting timestamps.
- **PasswordHashUtil**: A utility class for password hashing, used to encrypt user passwords.

Servlet Layer:

- **UserServlet**: The controller layer for user-related operations, handling login and registration requests.
- **VideoRegistrationServlet**: The controller layer for video registration, handling video uploads or registration.
- **VideoListServlet**: The controller layer for the video list, handling the display of video lists.
- **DeleteVideoServlet**: The controller is used to delete videos.
- **UpdateVideoServlet**: The controller is used to update video content.
- **VideoStreamServlet**: The controller is used for video playback stream processing.
- **SearchVideoServlet**: The controller is used to handle video search functions.

Service Layer:

- **UserService**: The business logic layer for user services, connecting UserServlet and UserDao.

- **VideoService**: The business logic layer for video services, connecting VideoRegistrationServlet, VideoListServlet, and VideoDao. The VideoService now connects to multiple servlets: including VideoListServlet, DeleteVideoServlet, UpdateVideoServlet, VideoStreamServlet, and SearchVideoServlet.

Data Access Layer (Dao):

`UserDao`: The data access layer for user data, responsible for interacting with the database, associated with `Model/Users`.

`VideoDao`: The data access layer for video data, responsible for interacting with the database, associated with `Model/Videos`.

Model Layer:

`Model/Users`: The data model for users, defining the structure of user-related data.

`Model/Videos`: The data model for videos, defining the structure of video-related data.

Configuration and Error Handling:

`DatabaseConfig`: Database configuration, managing database connections.

`error page`: Error page handling, likely a generic mechanism for error management. Added the description "Failed login or unauthorized access will be redirected to the error page."

Database and File Storage

Database (DDB/PR2): Stores user and video-related data, interacting with `UserDao` and `VideoDao`.

File Storage (Uploads/Videos): Stores uploaded video files, interacting with `VideoService`.

Relationships and Workflow

Interaction Between Pages and Servlets REST:

- **Index.jsp** likely serves as the entry point, redirecting to **Login.jsp** or other pages.
- **Login.jsp** handles user login through **UserServlet**.
- **RegistrationUser.jsp** and **RegistrationVideo.jsp** handle user and video registration through **UserServlet** and **VideoRegistrationServlet**, respectively with **POST**
- **ListVideo.jsp** displays the video list through **VideoListServlet**.
- **404.jsp** handles page-not-found errors.
- **EditVideo.jsp** uses **UpdateVideoServlet** to perform **PUT** requests.
- **ReproductionVideo.jsp** uses **VideoStreamServlet** to obtain video streams, not local url

- **Search.jsp** uses **SearchVideoServlet** for search functions.
- All pages redirect to error **Privacy.js**p when not logged in.

Interaction Between Servlets and Services:

`UserServlet` calls `UserService`, while `VideoRegistrationServlet` and `VideoListServlet` call `VideoService`.

Interaction Between Services and Dao:

`UserService` calls `UserDao`, and `VideoService` calls `VideoDao`.

All newly added servlets (such as Delete, Update, Stream, Search) call **VideoService**

Interaction Between Dao and Database/File Storage:

`UserDao` and `VideoDao` interact with the `DDB/PR2` database via `DatabaseConfig`.

`VideoService` may interact with `Uploads/Videos` through the file system to manage video files.

Utility Support:

`DataTimeUtil` and `PasswordHashUtil` provide support for time handling and password encryption across the backend.

Summary

This architecture follows the classic MVC (Model-View-Controller) pattern, using JSP as the view layer, Servlets as the controller layer, and Service and Dao layers for business logic and data access, with the database and file storage as the data layer. The system supports user management and video management functionalities, featuring a well-modularized design.

🔍 Search Functionality

The application provides a flexible **search feature** for querying video data based on different criteria. This functionality is implemented using the **SearchVideoServlet** and is accessible through the `Search.jsp` page on the frontend.

Request Method

The search functionality is triggered via an **HTTP GET** request, which allows the user to search without modifying server-side data.

Search Modes

There are **three distinct modes** of search supported:

1. **By Title**:
   Retrieves videos whose titles contain the input keyword.

2. **By Author**:
   Filters videos by the name or username of the uploader (author).

3. **By Date**:
   Returns videos uploaded on or around the specified date.

Backend Implementation

- When a user submits a search query, the frontend collects two parameters:
  - **Search field (option)** – title, author, or date
  - **Search value (keyword or date)**
- The `SearchVideoServlet` extracts these parameters and forwards them to the `VideoService`.

- Inside `VideoService`, the search request is passed to the `VideoDao`, which constructs a **SQL `LIKE` query** against the database table based on the selected mode:

```
SELECT * FROM videos WHERE <search_field> LIKE '%<search_value>%';
```

- The result set is then returned to the servlet, which formats and forwards the matching video list to the frontend for display.

This modular design allows the system to support flexible and scalable search capabilities while maintaining good separation of concerns in line with the MVC architecture.

Code repositories consulted

https://github.com/spring-attic/spring-mvc-showcase Demonstrates the features of the Spring MVC web framework

https://github.com/javalite/activeweb-simple Minimal ActiveWeb project

https://github.com/apache/tomcat

Bibliography consulted

Head First Servlets and JSP by Bryan Basham, Kathy Sierra, and Bert Bates (O'Reilly Media, 2008):

Java Persistence with Hibernate by Christian Bauer, Gavin King, and Gary Gregory (Manning Publications, 2015):