VIDEOH

Advanced Programming of Interactive Systems

2021/2022

GitHub repository:

https://github.com/FedericaBucchieri/Amato_Bucchieri_Videoh

Group 7

Amato Andrew - N° étudiant: 22106580

Bucchieri Federica - N° étudiant: 22106570

Project Description

VIDEOH is a video player application with educational purposes. The aim is to develop a platform application where teachers can upload video lessons and students can watch them in an interactive way.

Mainly, our application provides a video player for students and some interactive elements that allow them to specify if an argument was clear or not, to ask a question directly to the professor and match it to a specific part of the video lesson.

Going more in detail, VIDEOH allows you to enter the application both as a professor or as a student. Professors have a personal account and need to log in with username and password. Students instead can enter the application with a nickname in an anonymous way to incentivize participation, breaking the barrier of shyness.

So, after logging in, the application presents two personalized experience for the different group of users:

- **Professors** - This user group's homepage presents the list of all the video lessons uploaded by the specific professor. He/she has the chance to upload a new video inserting all the relevant details (title, description, ect..), to modify a previously uploaded video details or to see all the statistics concerning a specific video. Statistics comprise the all set of students' interaction, showing them accordingly with the video timeline. In this way the professor can clearly see which part of the lesson was clear or not and eventually takes note of all the questions posed by students in a specific moment. Each time a video is uploaded, the application generates a video code that the professor can share with its students.

- **Students** - This user group's homepage presents the possibility to insert a video-code and go directly to the corresponding video lesson. Each lesson is presented with a video player and some interactive buttons representing three possible types of interactions: positive, negative and questions. The first two buttons, if clicked, leave a tag in the video timeline providing visual feedback to the user. Each interaction is then stored to be shown from the professor's side. If the student wants to make a question with the available button, he/she has the chance to insert the desider question. Students have also the possibility to modify, delete or move in the timeline their interactions at any time with a drag & drop interaction. At the end of each lesson, users can control their questions, modify them or delete them completely in the review mode.

Programming models, languages and Toolkit

Our application is developed using Java programming language, Jakarta persistence API for database connectivity, vlcj Library for video support and Java Swing as Toolkit.

Overall Architecture and Code structure

Our application runs inside a single JFrame. It is instantiated by the Main class and all the required actions to display each scene are managed by a `SceneManager` instance.
This instance is responsible for displaying the correct Scene instance at any time, exploiting `CardLayouts` to switch between views *[Figure 1]*.

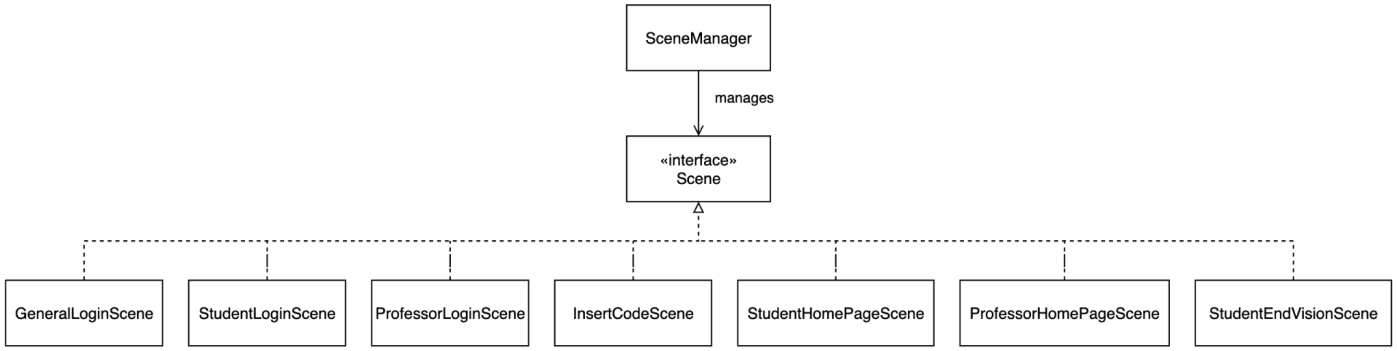

Figure 1. «Scene» interface implementation

Each scene has its own components and a mainPanel that can be attached to the general JFrame. So each Scene class manages the state and the displaying options of its own components. Each component is organized using a MVC architecture: the controller takes care of dispatching events to the scene and receiving events from every element of its view; the model takes care of the core logic and handles every interaction with the database; the view takes care of displaying each UI element required. Most of the scenes have only one Component, while the two main Scenes, namely the `StudentHomePageScene` and the `ProfessorHomePageScene` have multiple components *[Figure 2]*.

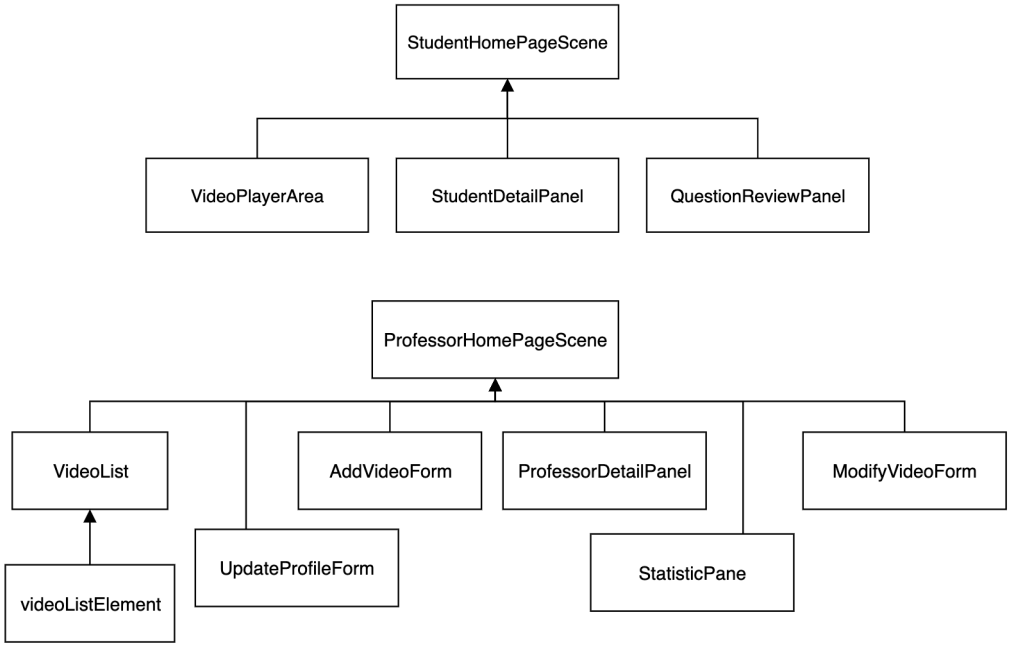

Figure 2. StudentHomePage and ProfessorHomePage components scheme

Only two components have been reused both for Student and Professors views: `VideoBox` (videoPlayerArea package) and `InteractionPanel` (interactionList package). Those two components have been personalized in order to be reused in different scenes.

In order to react to the user interaction with each different component, we implemented a personalized event mechanism to dispatch events bottom-up. The `Listener` interface allows every class to listen to events dispatched from other components. All the different types of Events can be found in the EventManagement package.

Library used: specifications

The core functionality of our app is “playing videos”. We didn’t expect that Java Swing doesn’t have any native Object to manage MediaFile; so to embed this feature in our app we relied on an external library: vlcj. This library allows the application that uses it to embed a native VLC media player: this also means that VLC must be installed on the machine that launches Videoh.
This library works almost fine everywhere, except on MacOS since, starting from Java 1.7, “on macOS, there is no longer any "heavyweight" window toolkit, everything is lightweight. This is a problem because VLC requires the window handle of a heavyweight window so it can be told where to render the video into.”[1] What this means in practice is that the `EmbeddedMediaPlayerComponent` (that is the object that takes care of all the video rendering) doesn’t work on any Java version later than 1.6 on

[1] From https://capricasoftware.co.uk/projects/vlcj-3/tutorials/prerequisites

macOS, so we needed to build our own `DirectMediaPlayerComponent`. This made things a little bit harder, since both of us use MacOS.
Our implementation is based on what we found on the vlcj official website and is well documented within our project.

App usage

For an initial setup and for a guided first usage, please see our README file on our Git repository.

Challenges and Simplifications

The main challenge for us was dealing with video playing, as already said. This slowed us down and prevented us from inserting more advanced features as expected.
As result, we didn't realize the drag&drop feature of interaction into the videosurface as stated in the previous document.
The main advanced interaction and personalized component in our application is the InteractionList: interactions can be moved in the timeline with drag&drop, it's possible to delete them and all the UI updated correspondly thanks to a good event dispatching mechanism.
Also, displaying the interactions according to the timeline of the video was not easy at all and we are conscious that on resizing there is the possibility that some values can be badly positioned in time. This is the consequence of the bad video handling: to get the actual video length the video has to run for some milliseconds: this prevents us from setting everything in advance and can create some blinking effects.