Personal Media Library

Final Project Report

***Abstract*—Personal Media Library is a unified cloud media library service that enables adaptive streaming of cloud-hosted video files. This report will provide a description of the Personal Media Library application, its design, architecture and technical implementation and discuss quality attributes achieved by the service. Contributions of project team members and experience acquired over the course of the project will also be addressed.**

I. Introduction

Media consumption is a growing trend in the modern era, quickly moving alongside technologies that enhance its availability and access convenience. Older formats of storing media objects are consequently being replaced by newer digital storage devices and services since they are overall more convenient to use and allow for more flexibility. These devices and services have seen a rapid growth to the point where physical media storage devices such as DVDs are becoming obsolete. Nowadays a great majority of media consumers store their media collections on personal computers or external hard disks, often connected to a local network. The limitations of these methods of storage are limited portability and availability of private media files, as well as decreased safety or security since files can be easily accessed or destroyed and be eventually lost forever. Solutions involving cloud and web applications were created to make digital files available through the internet, prevent their loss and ensure their security. However, currently, these services do not provide a platform for portable media consumption - a much-desired feature in an age where the increasing share of media consumption is done on the go through portable devices like notebooks, smartphones, and tablets. As such, little to none of the current cloud file hosting systems carter to avid media consumers or media professionals who work on content creation.

Personal Media Library is a project that aims at providing a simple web solution that would allow users to host their personal media files on the cloud and make them available in a form of a media library via the internet, while ensuring their security as well as preventing any potential loss to which other storage methods are susceptible. To achieve these goals, the system will provide a unified media file storage, organization, viewing and managing functionality, as well as feature adaptive streaming support for video files. The following sections will provide a more detailed look at how the system is implemented as well as how it accomplishes the goals of the project.

II. Functions and Features

The main functionality of the Personal Media Library is to provide a media library hosted on the cloud. To assist this main function, several features are defined within the service as follows:

- Authentication and Authorization process ensuring the validity of the user whilst preventing any unauthorized access, such as access to other users' libraries and files. Authentication is available using either a locally created account or Google account.
- CRUD operations that allow the user to upload, edit, view, and delete any media file in the system.
- HTTP dynamic adaptive streaming of video files enabling the users to stream their personal files from the service using the quality most suitable for their current bandwidth. This provides the benefit of not having to download any data on the user device's storage, as well as reduced network usage.
- The inclusion of Google Drive support and potentially other file hosting services thus allowing users to use their existing cloud file hosting provider without requiring migrating their media from one service to another.
- A simple and intuitive media library UI that allows the user to start using the system without a steep learning curve overhead.
- An Android application that makes the system more convenient to access on Android devices.
- Use of cloud technologies with high availability and scalability, which will ensure the quality of the web service.

These features are made available in the initial release, but many other features could be added to the system if they are implemented within a deadline outside of the scope of the project. For instance, one of such interesting features is adaptive audio streaming, with CD-quality music adaptively streaming in an mp3 quality on mobile devices, which has not been yet provided by anyone in the market. We believe that there exists an actual interest from many digital consumers enabling this project to become a valid business idea.

III. System Architecture

The system is implemented within the Phoenix web framework. The following section illustrates the system architecture and how it integrates within the flow of the framework, as well as explains design decisions made to implement the functionality of the system. The actual

implementation generally follows the high-level design, with some minor deviations in certain components that were deemed necessary.

The MVC model

The framework used in the implementation of this system is based on a variation of the MVC model using a functional programming language, thus providing a simple high-level architecture for the project. In Fig. 1, we can see a brief overview as to how the MVC model is used in our system.

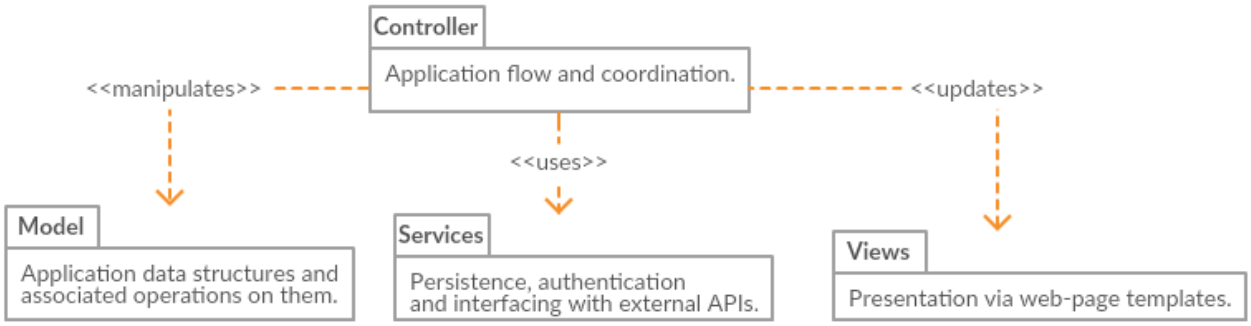


Fig. 1. MVC Architecture Overview

The designs of each package are further discussed in the following sections.

The Model

The model encapsulates all the data structures in the system and the methods operating on them, as well as provides persistence of these structures to the database. In Fig. 2, the design of the model package is presented in a UML style format.

The main data structures in the system relate to the users, videos, and sessions.

User structures represent user accounts and contain information about the user profiles, their associated external accounts and access credentials (such as Google account) and their media files (videos) associations.

Video structures represent video files belonging to the user and stored in the system. They contain metadata about the videos, as well as information allowing generating access links to them, such as video file storage service and file path on that service.

Finally, session structures represent current web session and provide login/logout functionality by creating and deleting user sessions.

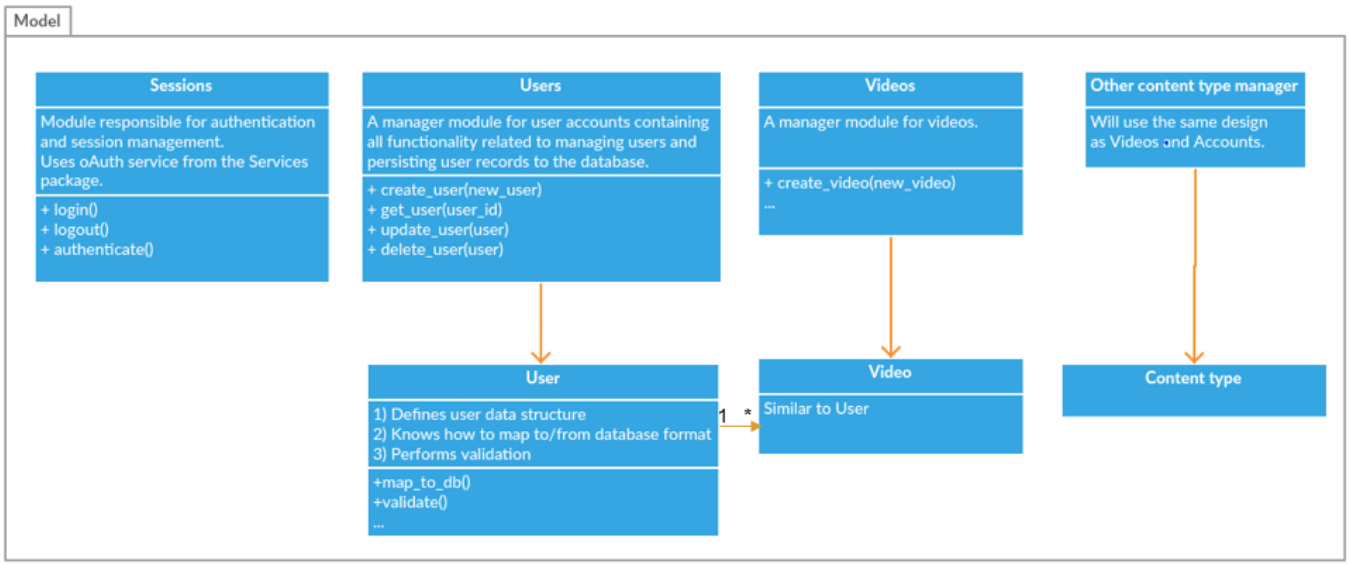


Fig. 2. Model Data Structures Overview

The Controller

Controllers provide the communication between models, services, and views. They act acts as an intermediary between the model and view entities. Fig. 3 displays the general design structure of the Controllers package.

The system will prevent access to pages to which a user is unauthorized and will enable their functionality only if authentication requirements are met.

When a user accesses the web service, they will be initially greeted with a front page granting them two available options:

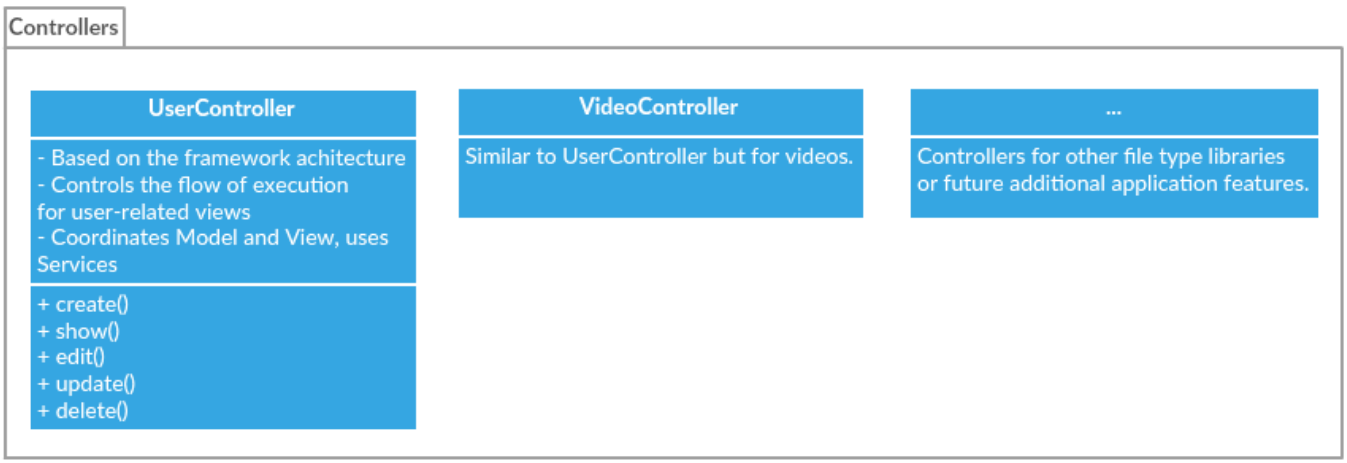


Fig. 3. Controllers Structures Overview

The two main controllers are User Controller and Video Controller. They are responsible for processing REST requests through delegating necessary operations to models and services and rendering the views requested.

The View

View entity represents the User Interface of the application. Views operate by rendering HTML templates using the data provided by controllers. Fig. 4 demonstrates the flow of view pages and access restrictions imposed on them.

register or login. After creating and/or logging into their account, users are presented with their personal media library in which they can start uploading, viewing and managing content. The settings page lets the user associate their external accounts and synchronize files to the media library database.

The Services

Services package provides an interface to the external services' APIs used by the system. The main services featured in the system are the storage-related services such as Amazon S3 and external file hosting services such as Google Drive, authentication services using OAuth and video transcoding

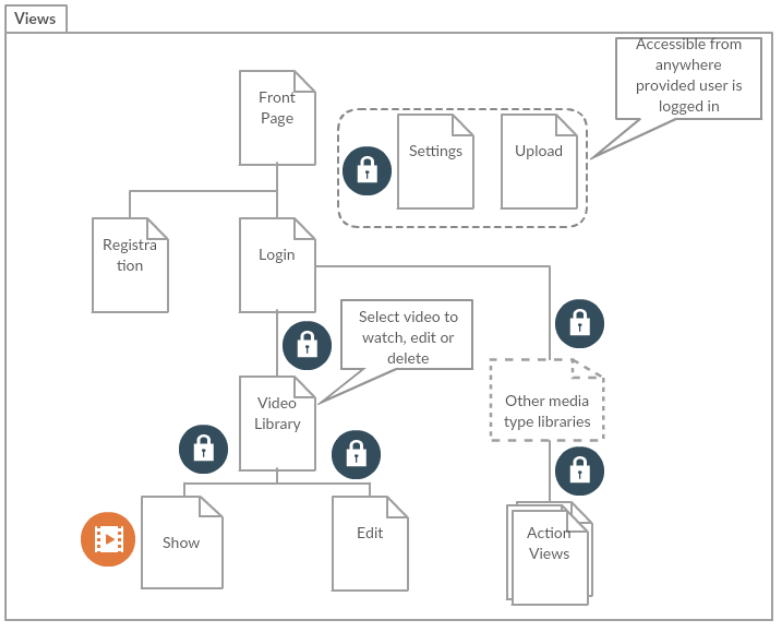


Fig. 4. Views Structures Overview

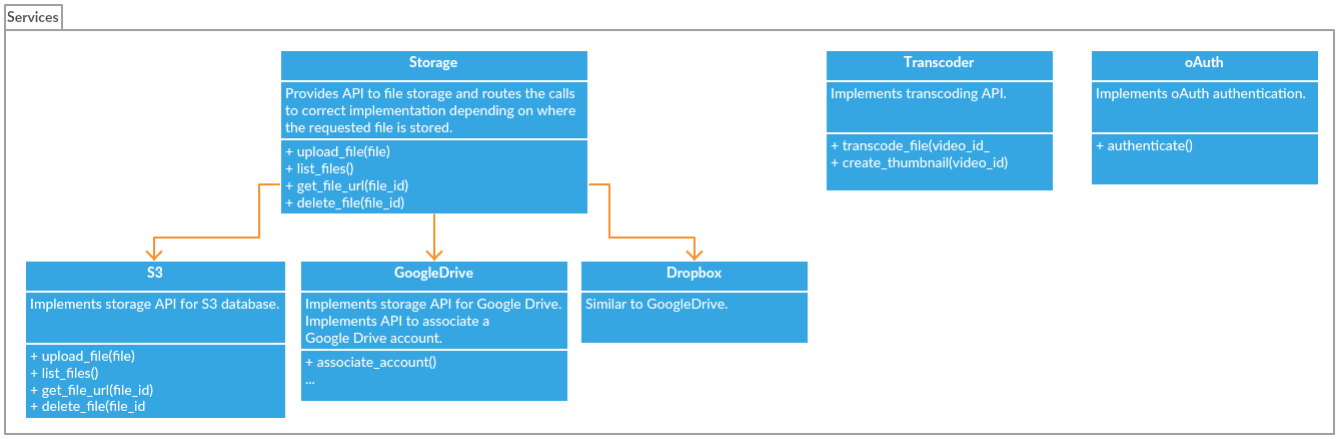


Fig. 5. Services Components Overview

services. A brief overview of the services package design is displayed in Fig. 5.

IV. Technical Implementation

The following section will discuss the technologies used to implement the solution as well as their main benefits.

The Web Framework

The system was implemented within the Phoenix web-development framework, which uses a functional programming language called Elixir. Elixir compiles to Erlang byte-code and executes on Erlang VM. This allows us to use all of the powerful Erlang features and libraries, most notably the Open Telecom Platform, which provides fault-tolerance and scalability. Most notable benefits of using these technologies are listed below:

- Functional Programming – suits perfectly for solving a web-server problem, which can be naturally represented using functional paradigm as a pipeline of functions processing incoming requests to generate responses.
- Server-side rendering – relieves computational load from user machines and helps to provide a unified experience on different platforms.
- OTP – provides fault-tolerance and distribution. Fault-tolerance is achieved via supervision trees, where all subsystems run within lightweight processes and are monitored by supervisors. In case of failure, supervisors terminate the failing process and immediately restart it with a valid state. Distribution is achieved by starting additional nodes running Erlang VM and registering them with the application. Our system does not use the distribution features but does make use of supervision trees to achieve fault-tolerance.
- Performance – Erlang VM is extremely fast and provides better performance compared to other popular web-framework, even without factoring the scalability features.

Amazon Web Services

Several AWS services were used to implement this project. The main advantages that AWS offer are reliability, scalability, and integration. These services are Cloud services and are designed with Cloud applications in mind, thus they already provide quality of service attributes required.

To host the application, we used an Amazon EC2 instance. For the project, a free tier instance was used, so we did not take advantage of scalability features or the increased processing power. However, in a real scenario, Amazon EC2 would be able to provide the scalability fairly easily by just creating more instances running Erlang VM and using OTP to handle the distribution.

Amazon S3 is used to store video files uploaded by the users and as an input/output storage for video transcoding jobs. It satisfies our requirements at a reasonable cost, operates well together with the Elastic Transcoder and allows for faster file transfers with EC2.

For video transcoding, the application uses Amazon Elastic Transcoder. The role of the transcoder is to process video files provided by the user and transcode them to an adaptive video format (MPEG-DASH) used for adaptive streaming. The benefit of using adaptive streaming as opposed to just supplying different video qualities is that the video quality can be changed in the middle of playback without interruptions based on the current network connection. Thus, when streaming video files, the quality will continuously change to ensure continuous streaming.

Using Amazon Elastic Transcoder allowed us to avoid having to create our own video-transcoding pipeline and dealing with video format copyright issues, thus suiting well the limited scope of our project. However, it is not the most

cost-efficient option in the long run and for the real-world production, performing transcoding in-house might be a better solution.

Other Services

Other services and technologies used in the system are MongoDB, OAuth and Google Drive.

MongoDB hosted on Amazon EC2 is used as a NoSQL database of choice to store user account profile and video data. Though there were some initial issues with integrating MongoDB with the data-mapper of Phoenix Framework, they were overcome by creating a small wrapper for the limited set of functionalities required by the application.

For authentication, OAuth is used since it provides a convenient way to use external sign-in providers for application login. It also provides the means to obtain permissions to use external services' APIs on behalf of the user, such as Google profile data or Google Drive.

Google Drive API is used to allow the user to synchronize their current media stored on Google Drive with the Personal Media Library database. Doing so enables the user to enjoy their media in a single unified environment, irrespectively of where the file is stored.

Android Application

As an extra feature, an Android client app for the web-application was developed.

This application provides a wrapper to the web-application while supplying native Android navigation and making sure that the UI is properly displayed on Android devices.

V. Quality Attributes Met by Using Cloud Technologies

The quality attributes of the system rely heavily on the quality attributes of the cloud services and technologies the system relies upon. We can assume that the web service in question is reliable at most to the point of guarantees provided by the services used by it.

The services in question offer high availability ensuring that the users can use the system without worrying about the service going down and becoming inaccessible.

They also offer scalability, which ensures that a growth in user population would not affect the quality of the service since the technologies used will dynamically adapt the system and handle increasing or decreasing workload.

Finally, an important attribute to which using cloud technologies respond is performance. Since most of the computations made by cloud technologies are made within the same cloud infrastructure (AWS), the network communication is faster and relies less on external networks and more on the internal in-cloud network infrastructure.

Considering our web-application implementation, we prioritized availability. In case of failure, only the user for which failure occurred is affected and service for that user is immediately restarted by the supervisor infrastructure to an operating state. Thus, while database consistency can theoretically be compromised in specific network failure scenarios, high availability of the service is provided to a high degree.

VI. Experienced Acquired

Throughout the scope of the project, the project team was presented with multiple technical and non-technical difficulties to overcome and acquired a wide range of experience related to cloud application development.

The main points we would like to highlight are presented below:

- Developing a web application using a web-development framework with cloud execution environment in mind
- Using a NoSQL MongoDB database and integrating it with the web-framework via creating a simple wrapper tailored to our needs
- Using external APIs and services either through provided language binders or directly using the REST APIs
- Developing an Android client app for a web application
- Deployment of solutions to the cloud infrastructure services, most notably AWS

VII. Team Member Contributions

Please find the breakdown of each team member contributions given below.

Illya Reutskyy:

- Base web-application architecture design and implementation
- MongoDB deployment, configuration, and in-app integration
- Amazon S3 integration
- Video transcoding and streaming
- Authentication (local) and authorization
- Google Drive account association and synchronization
- User Interface

Jafar Abbas:

- OAuth authentication
- Google Drive APIs: video streaming, synchronization, forwarding files to Transcoder by implementing file transfer between Drive and S3
- Testing and bug fixing

Daniel Makardich:

- Investigation of Google Drive and Dropbox APIs
- Android Application and integration of google service on the application service
- Testing and bug fixing
- Documentation

VIII. SUMMARY

The goal of this project was to design and implement a unified personal media library service on the cloud that enables adaptive streaming of cloud-hosted video files.

The implementation produced satisfies the requirements of the project and successfully achieves all the features included within the scope, except for Dropbox integration planned initially.

We believe that the experience acquired by the team over the course of the project will prove to be invaluable when working with cloud-based applications and technologies in the future.

IX. Appendix

A. \Links to services and repositeries

Web Application URL:

http://ec2-52-70-59-177.compute-1.amazonaws.com:4000/

Web Application Source Code:

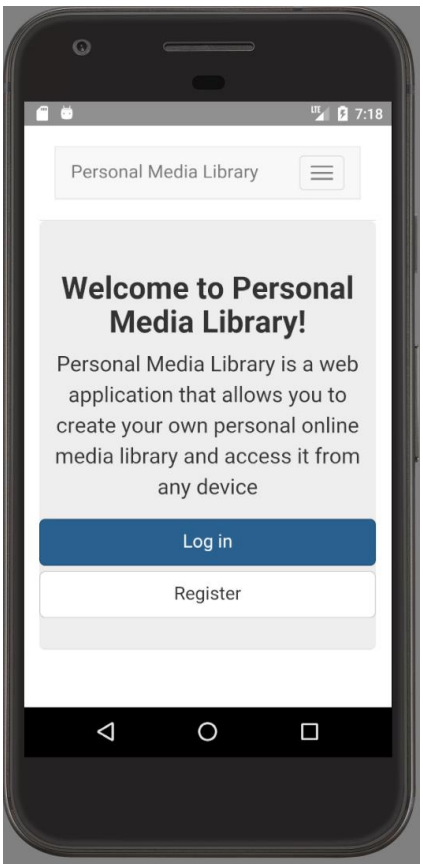
https://bitbucket.org/i-reuts/personal-media-library (The creation of an account to view source code on bitbucket is required)

Android Application Source Code:

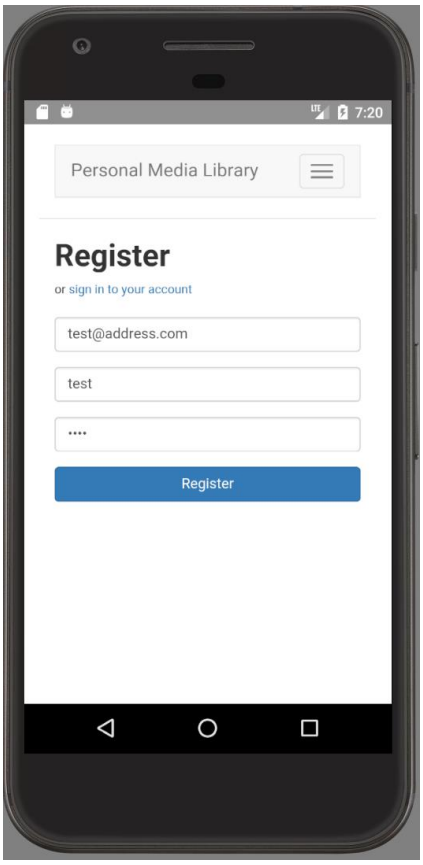
https://bitbucket.org/danmak/personal-media-library-android

B. Android ApplicationFunctioniality

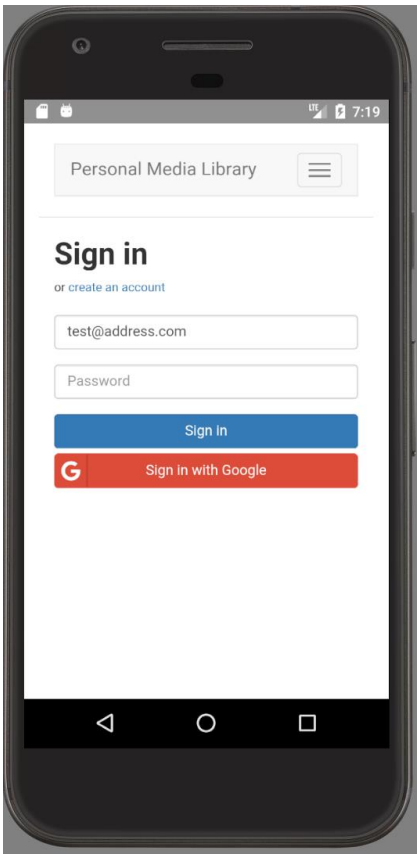
The following section is only meant as extra content displaying some additional features of the service and are not considered part of the report. The Android Application is displayed in the following pictures displaying the functionality of the system on a mobile device. The flow of the pictures from left to right show how the new user navigates through the pages.



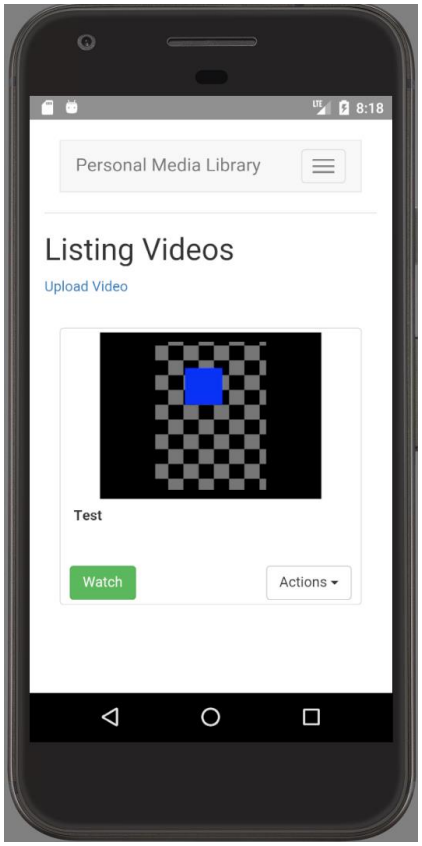
Landing Page



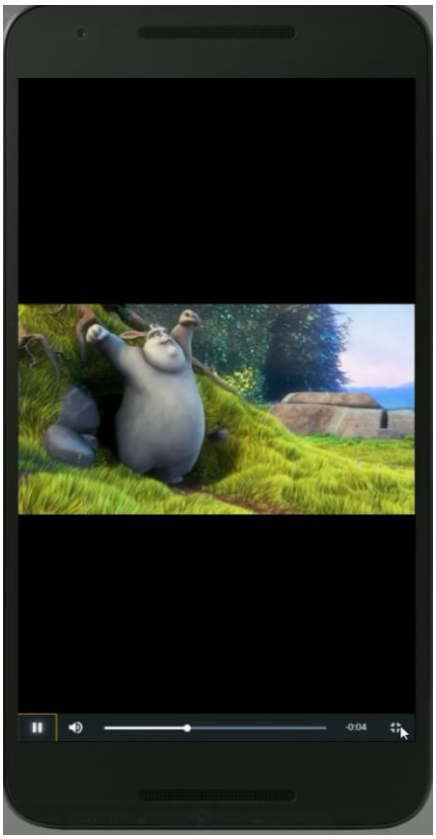
Register Page



Sign-In Page



User Page with a Video and a thumbnail



Video Playback