**Project Plan**

*Movie upload*

|  |
| --- |
| **Date** **:** **14 Dec 2021** |
| **Version** **:** **1.0** |
| **State : In progress** |
| **Author** **:** **Andrei Filip** |

#### Version history

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Author(s)** | **Changes** | **State** |
| 1.0 | 14/12/2021 | Andrei Filip |  | In progress |

**Distribution**

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Receivers** |
|  |  |  |
|  |  |  |

Contents

[1. Project assignment 4](#_Toc92656893)

[Context 4](#_Toc92656894)

[Goal of the project 4](#_Toc92656895)

[Scope and preconditions 4](#_Toc92656896)

[Preconditions 4](#_Toc92656897)

[Strategy 5](#_Toc92656898)

[End products 5](#_Toc92656899)

[Supported processes 6](#_Toc92656900)

[2. Project organisation 7](#_Toc92656901)

[Stakeholders, semester coach and team members 7](#_Toc92656902)

[Communication 7](#_Toc92656903)

[3. Activities and time plan 8](#_Toc92656904)

[Phases of the project 8](#_Toc92656905)

[Time plan and milestones 8](#_Toc92656906)

[4. Testing strategy and configuration management 9](#_Toc92656907)

[Testing strategy 9](#_Toc92656908)

[Test environment and required resources 9](#_Toc92656909)

[Configuration management 9](#_Toc92656910)

[5. Architectural choices 10](#_Toc92656911)

[Microservice architecture 10](#_Toc92656912)

[Used technologies 11](#_Toc92656913)

[6. Risks 13](#_Toc92656914)

[Risk and mitigation 13](#_Toc92656915)

[7. Bibliography 14](#_Toc92656916)

# Project assignment

## Context

Nowadays the best ways to immortalize precious moments is through recording videos. Most of us own a mobile phone or a recording camera, but what happens with the videos after we record them? The answer is that they end up on the local storage of the phone, or on a memory card, or even on a hard drive, which is quite convenient, but at the same time it may not always be the right thing to do. It is highly recommended to always back up your data to the cloud and to not only store it locally since, in case of damage or theft of the device, the videos can be lost forever.

## Goal of the project

The goal of Movie upload, as the name suggests, is to allow the user to upload a movie to the cloud, so that he can watch it and even share it with other people. The application assures that the movies are accessible from anywhere, at any time, having in mind security and offering a great user experience.

## Scope and preconditions

|  |  |
| --- | --- |
| **Inside scope:** | **Outside scope:** |
| 1. Upload a movie. | 1. Sharing the movie |
| 1. Play the movie. | 2 Content moderation |
| 1. Creating user accounts |  |
|  |  |

### Preconditions

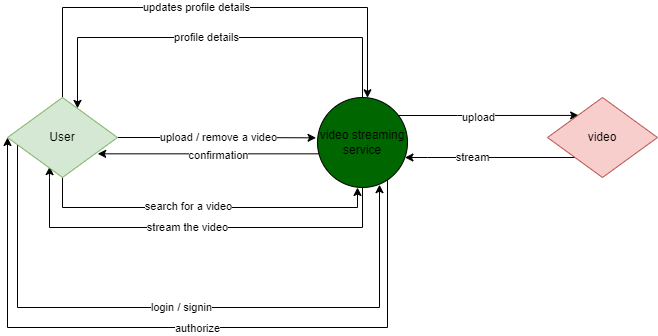
* Using microservices based architecture
* The apis must be RESTful
* The front-end must be created using a JavaScript library
* The application will be use by a large number of people

## Strategy

To ensure the even flow of the project and reaching the set goals, I have decided to pick an Agile way of working, by using the Kanban methodology. This choice has been made since Kanban is easy to pick up, it allows for a frequent feature delivery and is suitable for my situation, since I am working individually on this project at the moment. It also allows me to steadily improve the quality of the project over time and to make sure I am adding value to it.

## End products

## Supported processes



# Project organisation

## Stakeholders, semester coach and team members

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Abbreviation** | **Role** | **Availability** |
| *Leon Bokhorst* | *-* | *Stakeholder* | *Wednesday | 9:00 – 16:00*  *Thursday | 9:00 – 16:00* |
| *Andrei Filip*  *a.filip@student.fontys.nl* | *-* | *Team member* | *Wednesday | 9:00 – 16:00*  *Thursday | 9:00 – 16:00*  *Friday | 9:00 – 16:00* |

## Communication

We decided to communicate using Microsoft Teams during the time when we are not present in R10. This means of communication is used for short or longer questions, for feature showcase, but also for regular feedback talks.

# Activities and time plan

## Phases of the project

At the beginning of the project a lot of research is required, since most of the concepts covered in this project are new and require a firm understanding before diving into implementing features. I need to get used to the microservices approach of structuring the application, to how to build an API, how to create and use a mongo DB collection, how to create a CI/CD pipeline, how to do unit, integration, and end to end tests, but also how to use docker to create images and containers.

After this phase I will make sure that the required feature are implemented accordingly, by actively asking questions and keeping in touch with the main stakeholder.

The final phase of the project will be spent doing quality assurance and testing, making sure everything accordingly.

## Time plan and milestones

Each work item will have a set deadline which will be decided upon having in mind the research needed to be done, the preexistent understanding of the topic, if appliable, the complexity and the possible impediments to be encountered.

# Testing strategy and configuration management

## Testing strategy

For this project there will be used more testing strategies:

* unit tests, but only where it is necessarily, since they are not crucial for the microservice architecture
* integration, for assuring the components that have been individually tested communicate together appropriately
* end to end tests, to guarantee the user receives a quality experience, testing the application workflow from start to finish

## Test environment and required resources

The testing environment is included in the phased approach DTAP, in which the software passes through each environment. The code will go through development, where it is created, to testing where the test suites are run when the code is pushed to GitHub, which assures that it passes the tests. The acceptance phase follows, where the stakeholder gives feedback on the feature, then finally ending into the production phase where the application is optimized and hosted for production.

## Configuration management

The code will be stored on GitHub, which is the platform of choice for version control. There will be two branches: main and development.

* Main corresponds to the last DTAP phase. It is protected and cannot be directly pushed onto; code can only be merged into this branch by pull request, which triggers the production workflow, which runs the test suites, but also creates a docker image that is pushed to Docker hub.
* The development branch is used for the first phase of DTAP. When code is pushed to this branch it triggers the development workflow, which runs the test suites.

# Architectural choices

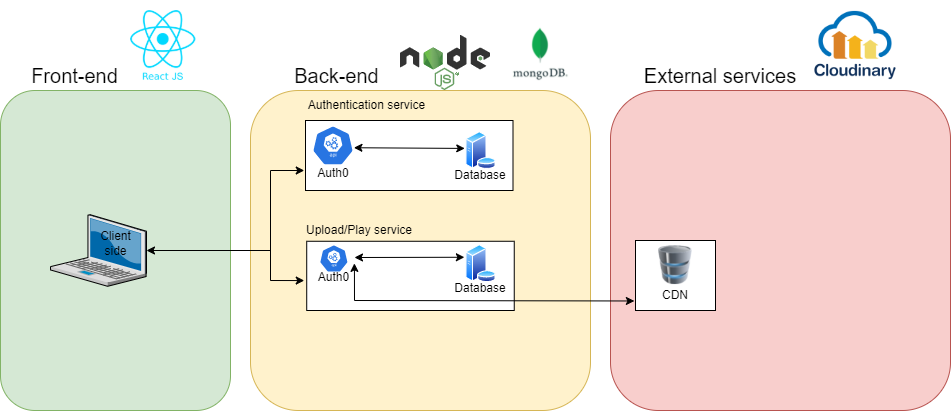
## Microservice architecture

For this project, I have decided to use the microservice architecture over the monolith architecture given to the benefits it comes with, but also to the precondition that the application will be used by many users.

The advantages of a microservice architecture are:

* **Flexible scalability:** the chosen architecture is more suitable in case of the mentioned precondition because it is more efficient. The application is made of different services and the requests from many users are spread out to the according microservice, which can be tuned to the number of requests it is expected to process, unlike in a monolith application where all the requests go to the same application.
* **Fault tolerance:** in a monolith architecture when a bug occurs it can make the whole application crash, creating a bad experience for the user. Since the services from the chosen architecture are loosely coupled between them, if a microservice goes down, the others will continue to work
* **Continuous delivery:**  Changes to a service do not affect the other services, resulting to a smooth integration, opposed to the situation of a monolith, where you would have to take down the whole application and to rehost it with the new small change.

This is an overview of the architecture of the application:



The front end is created with the java script library called React. The backend is made with the NodeJS framework, consisting of 2 microservices, one responsible for authentication of the users provided by Auth0 and the other one is responsible for uploading and playing the videos. The services are loosely coupled between them, each of them having its own database, created with Mongo DB. The upload/play service api also communicates with a CDN where the videos are stored at. The database from the upload/play service is used for keeping track of the which movies belong to a user and the relevant metadata for the video.

## Used technologies

#### Front-end

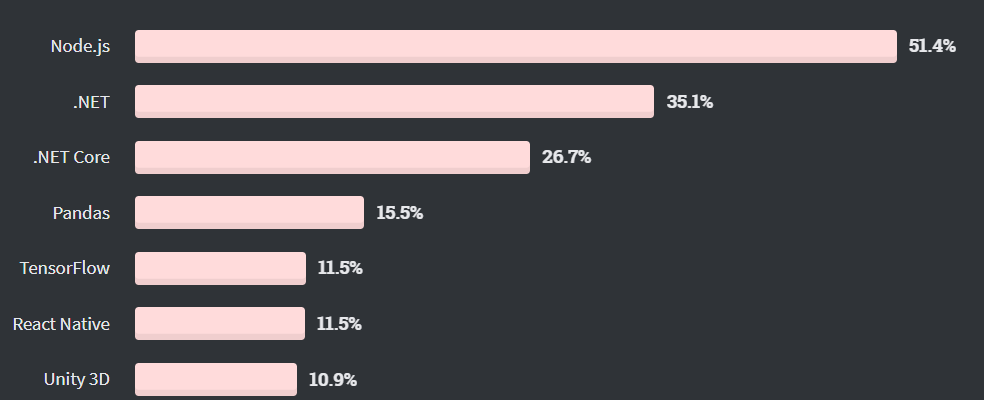
Given the precondition that the front-end must be created using a JavaScript library, after doing some research I had to choose between 3 popular choices: React, Angular, Vue. I decided to go with React since it is easier to pick up than the others, it is flexible, it has a broad community support, it does not impose an architecture, it a good choice for a dynamic website and for the performance.

React was created in 2011 by Facebook, nowadays it is being used by many large companies such as: Facebook, Netflix, Airbnb, Uber, etc. Having a basic understanding of HTML, CSS, and JS it is not hard to pick up React. The attractive parts of this library are the reusable components, which are independent pieces of code that can take an input (props) and return an output as HTML. In the past the components used to be created using classes, but nowadays they are done by using arrow functions. The library’s flexibility comes from the fact that once you have learned it, you can use it to build user interfaces on a variety of platforms. React Native can be used for creating mobile websites, Electron for desktop apps that run on Windows and Mac, Gatsby for static websites and Next.js for server side rendered applications, but more than this there is React VR for VR websites. The reusability of the components also contributes to this aspect. The JSX syntax, which “is a special syntax that looks like HTML, which converts React’s API calls and finally renders HTML” also contributes to its flexibility (Surve, 2021). Hooks are another great feature that allow you to “easily manage state logic between components, collocate similar logic in a single component and share data with components without props and classes” (Surve, 2021). React does not impose an MVC/ MVVM architecture like Angular or Vue, only being concerned with the view layer. Performance is being assured even when using many npm packages thanks to the JSX syntax, along with the Virtual DOM that is being updated in a minimal way by comparing the existent DOM with the new one and making the changes as few and efficient as possible.

#### Back-end

For managing the server side of the application, I had a few choices: Java, C#, and JavaScript. I decided to go with JavaScript and NodeJS given that I had a curiosity about how come the same language can be used for front-end, but also for back-end, and for the desire to learn the MERN stack.

Throughout the community NodeJS was the favorite of the developers, occupying the top stop for 2 years in a row (2019, 2020) in the annual developer survey [source](https://insights.stackoverflow.com/survey/2020#technology-other-frameworks-libraries-and-tools-all-respondents3), which means there is broad community support for it.



Having the opportunity to use the same language in front-end and back-end results in increased speed of work, but also making the job easier and more efficient. Another advantage brought by NodeJS is that it is up to the developer to use JavaScript, when loosed type is preferred for convenience and speed of development, or TypeScript when the precision of a strong typed language is needed. The presence of the node package manager is an advantage, allowing for quick and easy installation of node modules, which are mostly plug and play.

When creating a web application NodeJS is not enough to be used by itself, here Express, the most popular Node web framework which is a “fast, unopinionated, minimalist web framework” (https://expressjs.com/, n.d.), comes in handy supporting handling of HTTP verbs, separately handling requests on different routes, setting common application settings (e.g.: port to use, location of templates for rendering a response). Express is quite lightweight itself, but there is a wide range of middleware packages developed for it, which make it extremely useful and versatile: body-parser, cors, session, helmet, multer.

#### Database

For storing the data relevant to my application, I had to choose between using SQL and NOSQL. I decided to go for NOSQ, even though I had little to no experience to it because of its benefits. The data to be stored is the id of a user tied to a list of the metadata of movies uploaded by him. Given the fact that this is the only one relation between the user id and the movie, using SQL would have been over engineering. MongoDB is the most popular choice and easy to use type of NOSQL database. It is great for this project since it needs to be scalable horizontally and not vertically, like a SQL database. Given the fact that the metadata may change at some point, NOSQL is a better choice since it offers flexible data models, opposite to the other fixed data model offered by SQL. Getting started with MongoDB was easy, since it uses JSON which I was already familiar with. For extra validation and object document mapper I decided to use Mongoose to avoid writing boilerplate code and to speed up the production of the application.

# Risks

## Risk and mitigation

Risks need to be always considered when developing software, given the complexity of the whole process. For each risk factor there may be prevention and mitigation activities, but there may be scenarios where this is not possible, for example when the developers get sick.

Poor productivity can be a result of more factors such as setting goals that are unachievable or feeling burned-out. To prevent this from happening close contact will be kept with the stakeholder and following a work schedule, while also taking enough breaks.

Poor implementation is a result of insufficient research of the used frameworks, but this is quite possible to happen given that the frameworks used in this project are new to me. To avoid this, I will make sure to get code review from the teacher and classmates, but also to follow tutorials from trusted sources, that explain the concepts that are to be used properly.

Sickness is hard to prevent and is an external factor which can occur at any time.

|  |  |  |
| --- | --- | --- |
| **Risk** | **Prevention activities** | **Mitigation activities** |
| 1. Poor productivity | Regular meetings with the stakeholder, getting peer review, sticking to a working schedule | Asking for help from the teacher, peers |
| 1. Poor implementation | Researching the conventions and good practices regarding the used frameworks. | Getting code review from the teacher and peers |
| 1. Sickness | - | - |

# Bibliography

StackOverflow. (2021). *2021 Developer survey*. Retrieved from StackOverflow: https://insights.stackoverflow.com/survey/2021#technology-most-popular-technologies

Surve, S. (2021, 2 18). *Why You Should Use React.js For Web Development*. Retrieved from freeCodeCamp: https://www.freecodecamp.org/news/why-use-react-for-web-development/

TechStrikers. (2021). *MySql Advantages and Disadvantages*. Retrieved from TechStrikers: https://www.techstrikers.com/MySQL/advantages-and-disadvantages-of-mysql.php