CSE 3105/ CSE 3137

OBJECT ORIENTED ANALYSIS AND DESIGN

FALL 2020

COURSE PROJECT: *Media Browser Application*

System Design Document

Group 1

Ecenur Atıgan– 180315063

Çisem Akman– 170315020

Barış Alp Aslan – 170315009

Ömer Şirin – 150315013

Gülben Emiroğlu – 160315041

Erdem Emiroğlu – 170315026

16 January 2021

Table of Contents

1 Introduction

Our program opens and displays movies and behind-the-scenes footage, movie credits and sound files. The aim of the program is to combine movie and music application in one application. We developed it with reference to common apps such as Netflix, Youtube and Spotify.

1.1 Purpose of the System

1.2 Design goals

Design Goals for Watchy

-Usability:The free time for the movie selected by voting every week is one week.

-Usability:User can't comment or like without premium membership.

-Usability:User can't download content without premium membership.

-Reliability:User can login or continue as guest. A guest user can become a premium member and set a password. Premium member can edit their profile.

-Supportability:Membership trial period can't be more than 30 days.

-Security:Watchy should be secure,must not open on nonauthorized systems.[deduced from application domain]

-Well-Defined Interfaces: Watchy should be has readable and understandable interface for user and guest.[anticipation of change by developers]

-Fault Tolerance:Watchy should be fault tolerant to video freezing or application slowness.

Trade-offs

Cost vs. Robustness

-If the application has fault tolerance,security or well-designed interfaces,low cost system does not provides robustness.

Fault Tolerance vs. High Performance

-If the system has high performance on a authorized system, fault tolerance decrease.

Reliability vs. Backward-compatibility

-If the application still can use with old versions, it decreases the reliability. With researches, application's hacking possibility increases in old versions.

2 Current Software Architecture

In our application, users can both play videos (movies) and play music. Similar to our app, we can show you *Netflix*, which allows us to watch movies, and *Spotify*, which allows us to listen to music. Let's consider their software architectures separately;

1. **Netflix:**

It works based on Amazon cloud computing services and Open Connect. Both systems must work together to provide high quality video streaming services globally. In terms of software architecture, Netflix consists of three main parts: Client, Backend and Content Delivery Network.

- Netflix can adapt its streaming services transparently under certain circumstances such as slow networks or overloaded servers.
- Client, provides the best viewing experience for each and every client and device.
- Backend handles everything not involving streaming videos.
- Open Connect Appliances (OCAs) optimized for storing and streaming large videos.

Some common failures that the system have been addressed as follows:

- A failure in resolving service dependencies.
- A failure of executing a microservice would cause cascading failures to other services.
- A failure of connecting to an API due to overloading.
- A failure of connecting to an instances or servers such as OCAs.

(To detect and resolve these failures, the API Gateway Service Zuul ([20]) has built-in features such as adaptive retries, limiting concurrent calls to Application API.)

2. **Spotify:**

It works with a Microservice architecture, where all software developers code in a closed “territory” with his specific functionalities. Each Microservice have only one simple responsability and in most cases they have a private database with his own logic that cannot be intervened by another process. This gives to the application the ability to escalate quickly, deploying different services in different machines with personalised perfomance depending in the demand of each service.

Microservice architecture is;

- Easier to scale based on real-world bottlenecks
- Easier to test
- Easier to deploy
- Easier to monitor
- Can be versioned independently

But

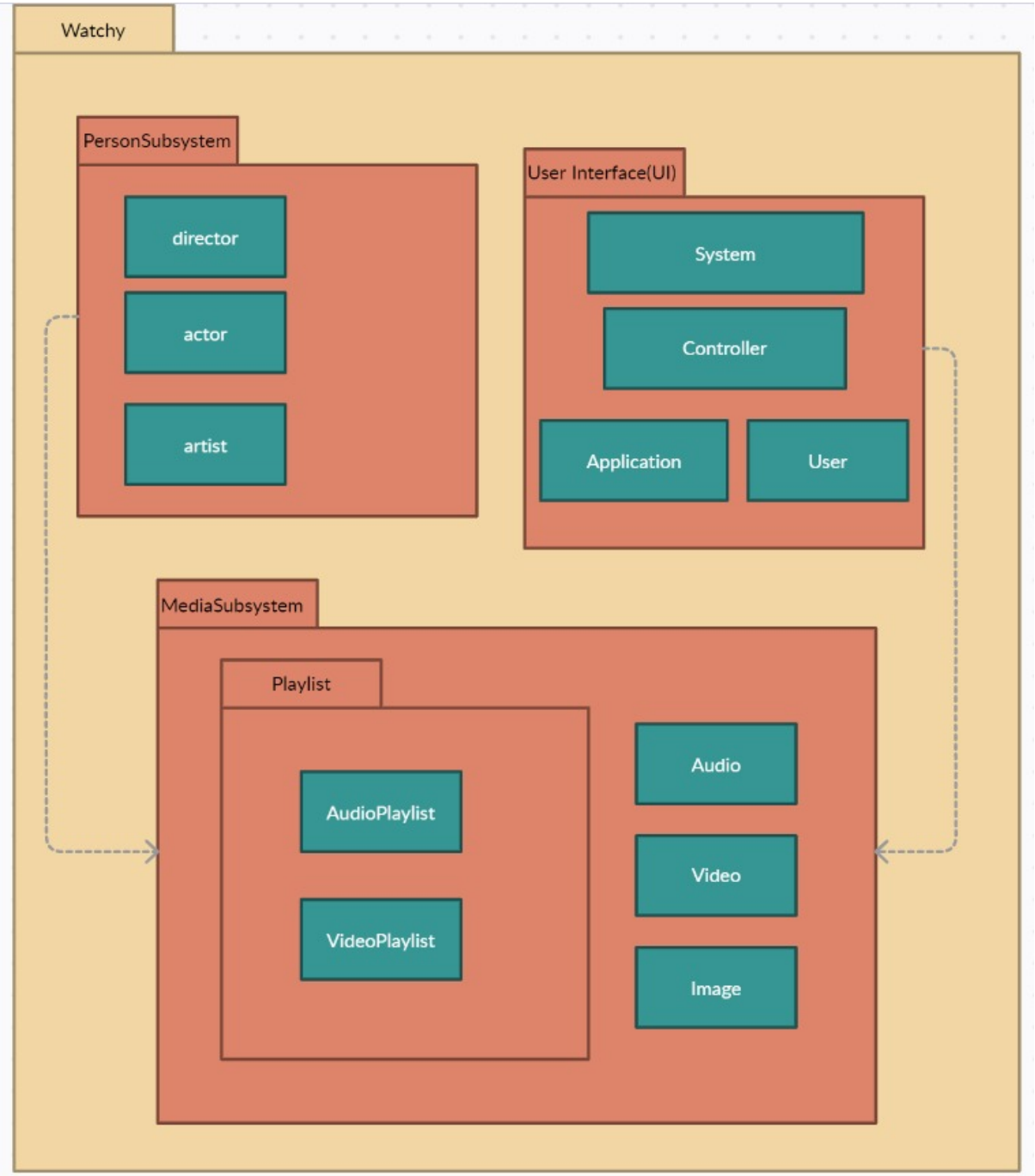
- Need good documentation and discovery tool
- Create increased latency

Also Niklas Gustavsson, Spotify backend developers says;

- The Spotify backend architecture is heavily service oriented. The backend is composed of about a hundred services, most of them fairly small and simple.
- The various clients keeps a persistent connection to a backend service called "accesspoint".
- Clients for desktop, mobiles and our embeddable library, all share a common code base.
- Audio is retrieved from local cache, peer-to-peer or from our storage.
- Infrastructure is heavily based on Debian and open source software in general.

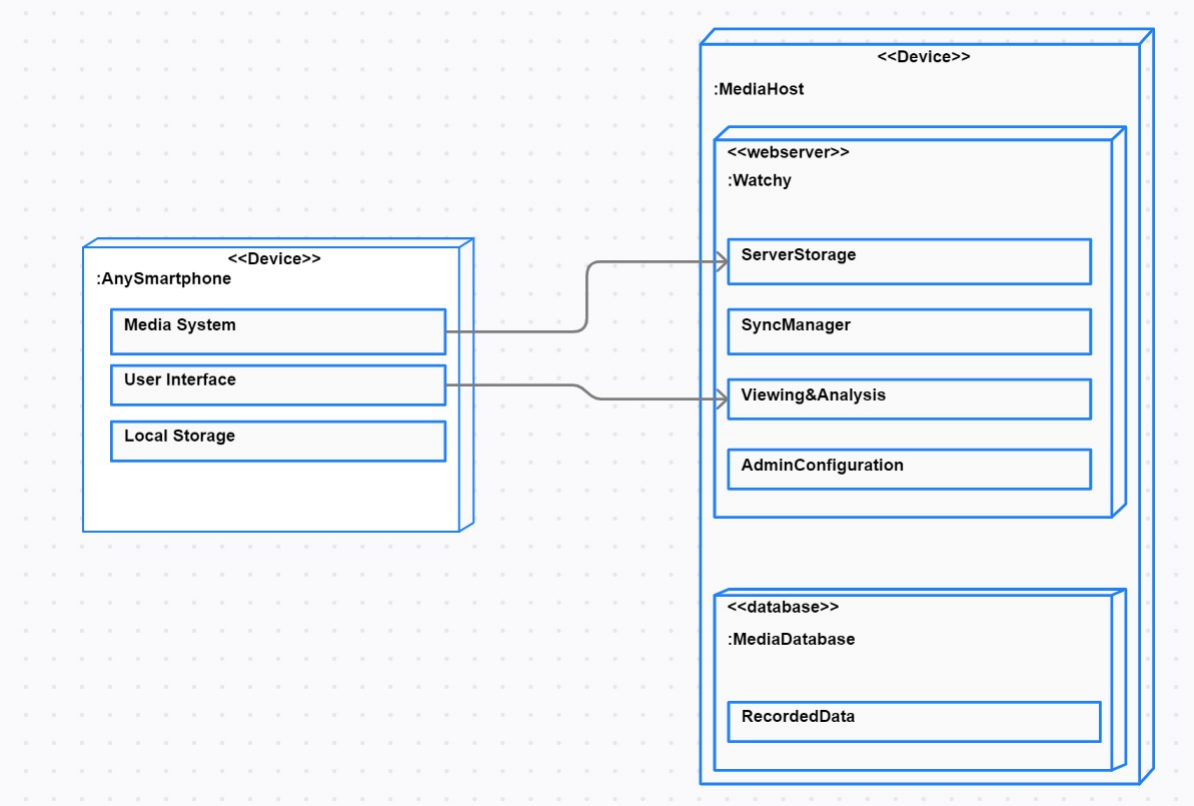
3 Proposed Software Architecture

3.1 Subsystem decomposition



- PersonSubsystem: Holds the information about people in the movie (director - actor - artist)
- UserInterface: Areas the user is responsible for within the application.
- MediaSubsystem: Responsible for keeping all information of audio video and image.
- PlaylistSubsystem: Playlist is created, and these playlists of audio and media are kept in the playlist subsystem, which is a subsystem of the media subsystem.
- We can reach media subsystem with UserInterface.
- The video image and audio information in the media subsystem includes a person subsidy.
- Since all classes and subsystems are interrelated, we used open layer architecture.

3.2 Hardware/software mapping



- Media System / User Interface / Local Storage: Runs on AnySmartphone.
- ServerStorage / SyncManager / Viewing&Analysis / AdminConfiguration: Runs on Watchy server.
- RecordedData: Runs on MediaDatabase.

3.3 Persistent data management

User is persistent and this user is kept in the database.

3.4 Access control and security

Access Matrix Table

Classes

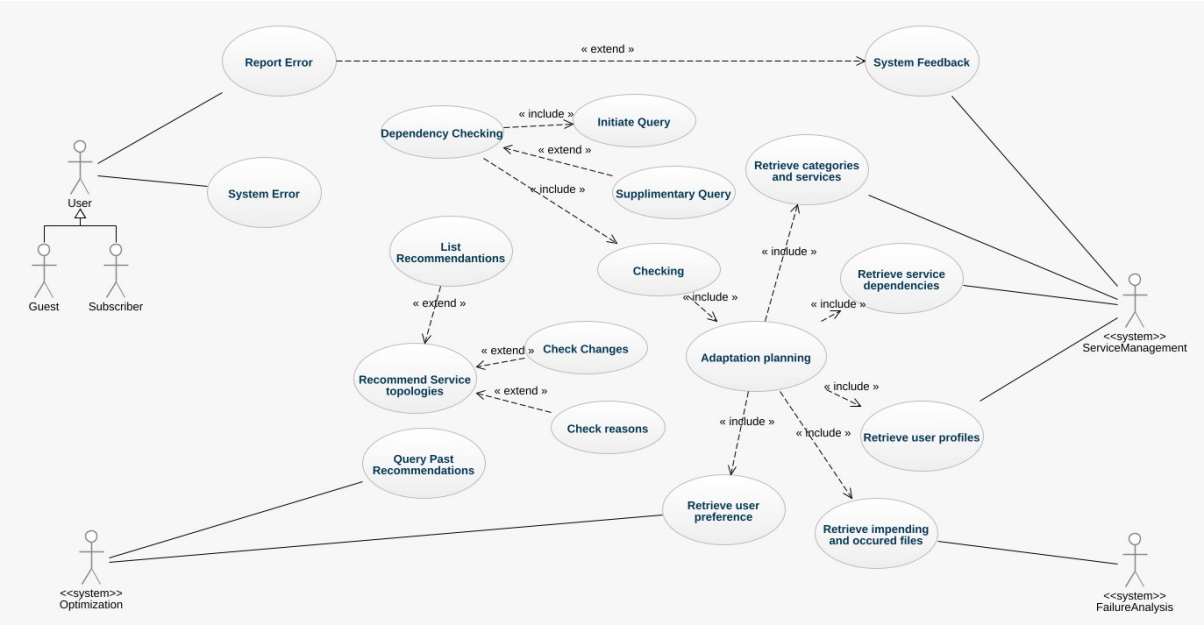
Actors

	Application	Controller	System
User	enterSystem() signIn()	<<guest>> <<sign>>	[entered]notify(guest) zoomIO() checkInformation() watchFreeMovie() watchMovie() createPlaylist() download() like()
Resource Manager	addVideo() addMusic() addImage()	<<save>>	checkOptions() checkSaved()

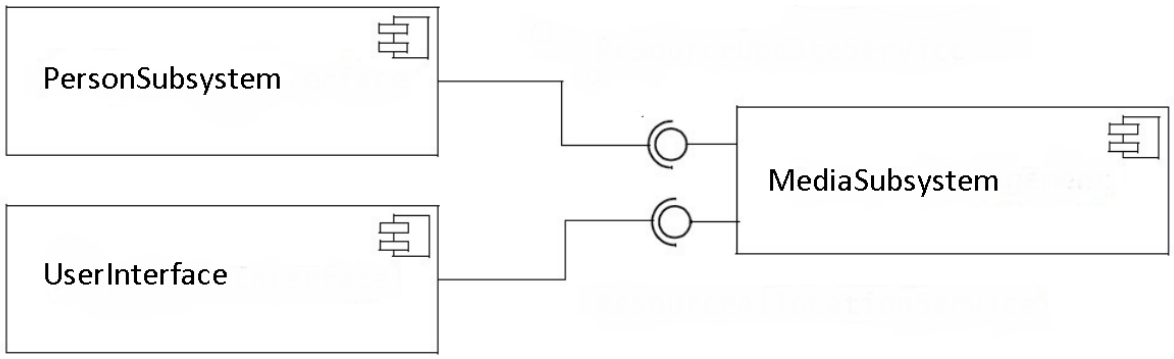
Program must able to save user password and some personal informations.

Nobody else can reach personal data and IP address.

3.5 Boundary conditions



4 Subsystem Services



- MediaSubsystem provides to UserInterface and PersonSubsystem.
- UserInterface and PersonSubsystem indicates MediaSubsystem.

5 Glossary

- **Password:** A string of characters that allows access to a computer system or service.
- **Trial period:** A period of time for testing or assessment.
- **Watchy:** Name of our application.
- **PersonSubsystem:** Subsystem that holds the information about people in the movie.
- **UserInterface:** Areas the user is responsible for within the application.
- **MediaSubsystem:** Subsystem that responsible for keeping all information of audio video and image.
- **PlaylistSubsystem:** Subsystem that playlist is created, and these playlists of audio and media are kept in the playlist subsystem.
- **Media System:** A system that holds video, image, audio.
- **Local Storage:** The regional storage space the app has on the phone.
- **ServerStoage:** Is a type of server that is used to store, access, secure and manage digital data, files and services.
- **SyncManager:** The structure responsible for synchronization.
- **Viewing&Analysis:** Viewing according to file type and analyze the given format.
- **AdminConfiguration:** Allows an Operator to configure deployed bundles.
- **RecordedData:** A data collected, stored or transmitted in Media Database of our application.
- **IP address:** A unique address that identifies a device on the internet or a local network.
- **Service Management:** Is a customer-focused approach to delivering information technology.
- **FailureAnalysis:** Is the process of collecting and analyzing data to determine the cause of a failure, often with the goal of determining corrective actions or liability.
- **Optimization:** The action of making the best or most effective use of a situation or resource.

6 References

- Medium. n.d. *A Design Analysis Of Cloud-Based Microservices Architecture At Netflix*. [online] Available at: <https://medium.com/swlh/a-design-analysis-of-cloud-based-microservices-architecture-at-netflix-98836b2da45f>
- Slideshare.net. n.d. *How Spotify Builds Products (Organization. Architecture, Autonomy, Ac...*. [online] Available at: <https://www.slideshare.net/kevingoldsmith/how-spotify-builds-products-organization-architecture-autonomy-accountability>
- Medium. n.d. *Microservices Architecture At Spotify*. [online] Available at: <https://medium.com/codebase/microservices-architecture-at-spotify-beac905e9622>
- Quora.com. n.d. *What Is Spotify's Architecture? - Quora*. [online] Available at: <https://www.quora.com/What-is-Spotifys-architecture> [Accessed 16 January 2021].
- Netflix.github.io. n.d. Netflix Open Source Software Center. [online] Available at: <https://netflix.github.io/>
- StackShare. n.d. Netflix - Netflix Tech Stack. [online] Available at: <https://stackshare.io/netflix/netflix>
- Ghiciuc, I., 2017. All The Phases Of The Mobile And Web Product Development Process. [online] Thinslices.com. Available at: <https://www.thinslices.com/blog/phases-mobile-product-development-process>
- DeBrusk, C., 2019. What Should We Do To Prevent Software From Failing?. [online] MIT Sloan Management Review. Available at: <https://sloanreview.mit.edu/article/what-should-we-do-to-prevent-software-from-failing/>
- Continelli, A., 2017. [online] Business.com. Available at: <https://www.business.com/articles/aaron-continelli-identify-and-prevent-software-failure/>
- Spotify Engineering. 2013. Backend Infrastructure At Spotify. [online] Available at: <https://engineering.atspotify.com/2013/03/15/backend-infrastructure-at-spotify/>
- Building a Media Browser Client. 2019. [online] Available at: <https://developer.android.com/guide/topics/media-apps/audio-app/building-a-mediabrowser-client>