Extra Ears – Subtitles and More for Movie Theater Accessibility

Project Team:

Benjamin Schmitt (bjschmitt@wpi.edu)

Jack Ciroli (jvciroli@wpi.edu)

Advisor

Professor Johnathan Weinstock

Introduction

In a recent survey, 50% of Americans utilize subtitles to watch content most of the time, while 55% believe that content has become harder to hear and understand.[1] Although this survey specifically inquiries about Streaming and Cable programming, it exemplifies the trend of subtitle usage among the American population. Many complaints arise from poor audio mixing, dark scenes, and an overall inability to focus. In this study, many pointed out that subtitles were a way of focusing on the plot and improving their overall experience.

On a non-convenience side, subtitles offer people with hearing impairments to enjoy content. Theaters allow for an encompassing experience of screen and sound, and with impairments or poor sound, the experience is immensely impacted. Creating a system to allow subtitle usage can greatly improve the access of theatres and the overall experience for all.

The purpose of this project is to develop a system that allows individuals to enjoy subtitles in Movie Theaters. This project will be a combination of a desktop app and mobile application. The Extra Ears system is to aid in the accessibility of movie theaters for theatergoers. The system will be designed to allow end users to receive subtitles on their mobile devices during a film, improving the experience for the hearing impaired and/or users who just enjoy subtitles. By creating an easy-to-use system, Extra Ears can make movies more accessible to all individuals.

[1] Zajechowski, “Survey: Why America is obsessed with subtitles.”

Background

1.0 Accessibility

The first key aspect of any successful software product is accessibility. Traditionally, it is defined as something easily reached or used.[2] In the case of software, it's imperative to allow people of all backgrounds to utilize the technology being provided. W3C denotes accessibility as being "…essential for developers and organizations that want to create high quality websites and web tools, and not exclude people from using their products and services."[3] Furthermore, the United Nations imposes Article 9, stating that individuals with disabilities have the right to partake in all aspects of life, including "…information, communications and other services."[4] In an age with increased digitalization, it is imperative that software is designed with all people in mind. In the case of movie theaters, many have been slow to adapt, and with widely acknowledged sound and screen issues, the bar of accessibility will keep raising without active involvement.

In the context of Extra Ears, one of the main goals is to make theaters more accessible to all. By providing an easy to use, simple to understand client-side application, Extra Ears can be a perfect solution to movie theater accessibility.

[2] Merriam Webster, "Accessible."
[3] W3C, "Accessibility."
[4] United Nations, "Article 9 – Accessibility."

2.0 Compatibility

As the number of mobile device options for consumers to choose from increases, it is becoming increasingly more important that mobile applications can work on different platforms. To accommodate as many potential users as possible, the Extra Ears mobile application utilizes the React Native framework to ensure simple compatibility with both iOS and Android platforms using a single codebase. The framework also provides native-like performance and UI/UX features as well as many libraries that run smoothly on both platforms. By leveraging the power of React Native, the app is easily available to a wider audience without sacrificing performance, function, or usability.

3.0 Current Technology in Use

Currently, theaters around the United States have been slow to adapt to new accessibility requirements. AMC Theaters, one of the U. S's largest theaters, has begun offering closed captioning at certain showtimes.[5] Although it provides a minor solution for hearing impaired clients or individuals who enjoy subtitles, it vastly limits the freedom of people to choose when to visit the theaters or what films they'd enjoy seeing. Apart from designated screen times, some devices/tools exist to provide the closed captioning functionality to patrons. Some theaters offer utilities like mirror captions, special smart glasses or small devices that can be held.[6]

Although there are solutions that provide subtitling and closed captioning services for patrons, there doesn't exist an easy to use, hassle free solution.

---

[5] NPR, "The world's largest movie theater chain is adding open captions at 240 U.S. locations."
[6] Circle Translations, "What is Closed Captioning in Movie Theatres & How It Works."

4.0 Java, JavaFX

For developing a desktop application as well as prototype infrastructure, there are a multitude of libraries, languages, and software available. For this software, the team decided to utilize Java and JavaFX for the desktop side of the application. Java has a multitude of advantages over other languages such as being object oriented and being platform independent.[7] Furthermore, Java holds a plethora of libraries vital to large scale applications, such as networking and UI development tools.

To develop a front-end of the application, JavaFX will be vital to integrating it into the backend of our application. JavaFX is highly customizable and intuitive, with even more libraries and easy integration into existing Java code.

5.0 Mobile Development

When developing a mobile application, there are a multitude of languages and frameworks to choose from, however, React Native is one of the best choices for the Extra Ears system. The React Native framework provides clear benefits over other languages and frameworks, including an extensive library support and excellent cross-platform compatibility. React Native is also very intuitive, as it is written very similarly to popular languages, and customizable to allow developers to seamlessly integrate their app with other software and the app's backend. This creates a very cohesive development experience and a great user experience. React Native provides everything developers need to quickly build excellent apps for both iOS and Android platforms, which is why it was used for the development of the Extra Ears mobile application.

---

[7] IBM, “Advantages of Java.”

Software Requirement Specifications (SRS)

1.0. Introduction

1.1. Purpose

The purpose of this Software Requirements Specification (SRS) document is to provide a description of the Extra Ears System. It aims to explain the purpose of the system, proposed features, and potential constraints. This document is intended for stakeholders and Extra Ears system developers.

1.2. Scope of Project

This software will be a combination of a desktop app and mobile application. The Extra Ears system is to aid in the accessibility of movie theaters for theatergoers. The system will be designed to allow end users to receive synced subtitles on their mobile devices during a film, improving the experience for the hearing impaired and users who just enjoy subtitles. By creating an easy-to-use system, Extra Ears can make movies more accessible to all individuals.

Specifically, this system has two major components – a mobile application and a desktop command and control program. The desktop side will allow movie administrators to relay movie information to connected mobile devices, while the mobile application will play time-synced subtitles. The system will also allow for timing management and language options.

1.3. Glossary

Term	Definition
Accessibility	The practice of making software/environments open and usable by all potential users.
Active Movie	Movie that is currently being managed by an administrator and being shown in a theater.
Administrator	Person managing the desktop-side application for controlling a selected movie.
End-User	Movie-going customer and user of mobile application.
Stakeholder	Any person with an interest in this project who is not a developer.
Software Requirement Specification	A document that describes the functions of a software system and its purpose and goals.

1.4. Document Overview

The next section of this document gives a general overview of the functionality of the Extra Ears system. It describes general requirements to establish background for the technical necessities of the product.

The third section, 3.0. Requirements Specification of this document is for developers and outlines technical details of the Extra Ears system.

Both sections 2.0 and 3.0 describe the Extra Ears System as a whole, with specific sections on both the mobile application side, and the desktop command and control side.

2.0. Overall Description

2.1. System Environment

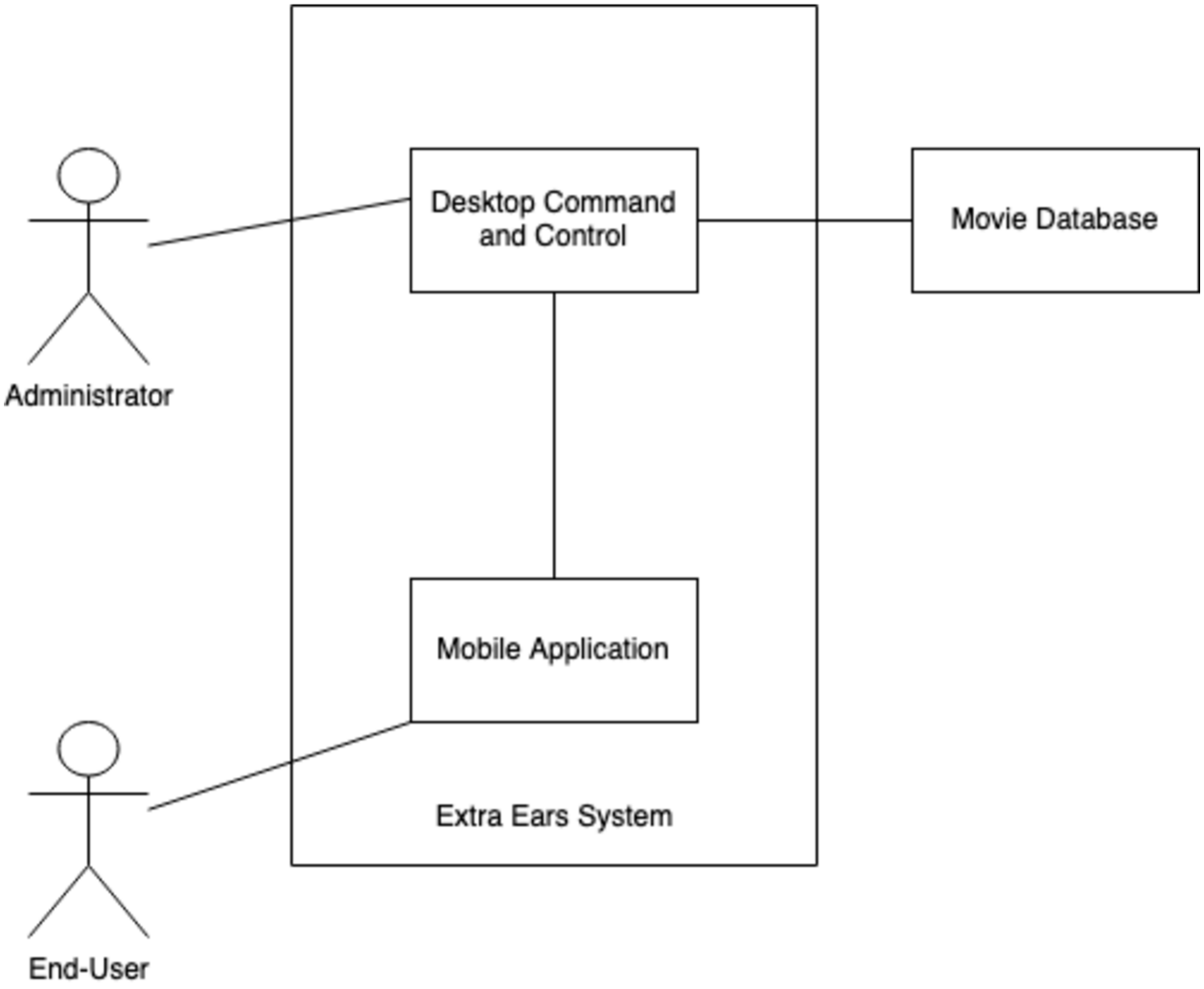


Figure 1 - System Environment

2.2. Functional Requirements Specification

This section outlines the use cases for administrators and users separately. The end-user has three use cases, and administrators have 3 use cases.

2.2.1 End-User Use Cases

The End-User has the following sets of use cases:

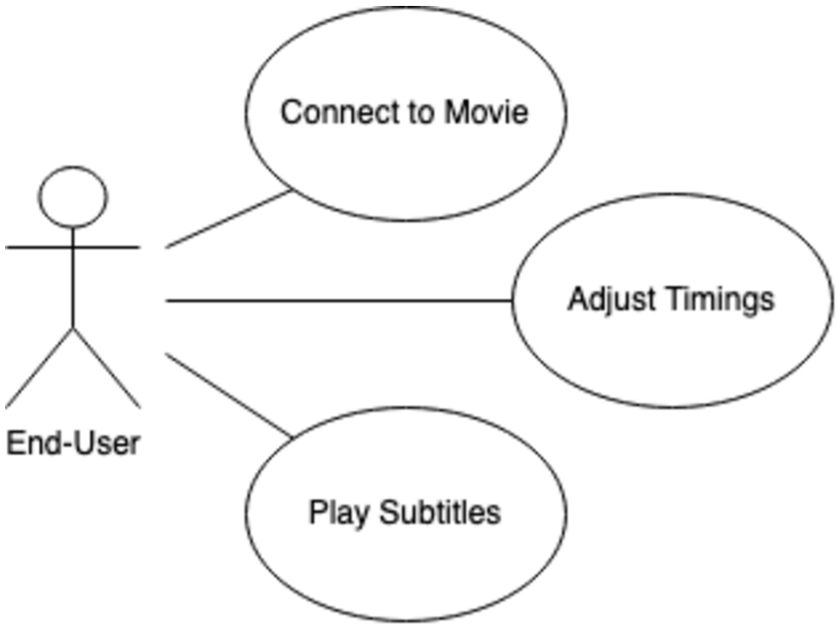


Figure 2 - End-User Use Cases

Connect to Movie Use Case

Use case: Connect to Movie

Diagram:

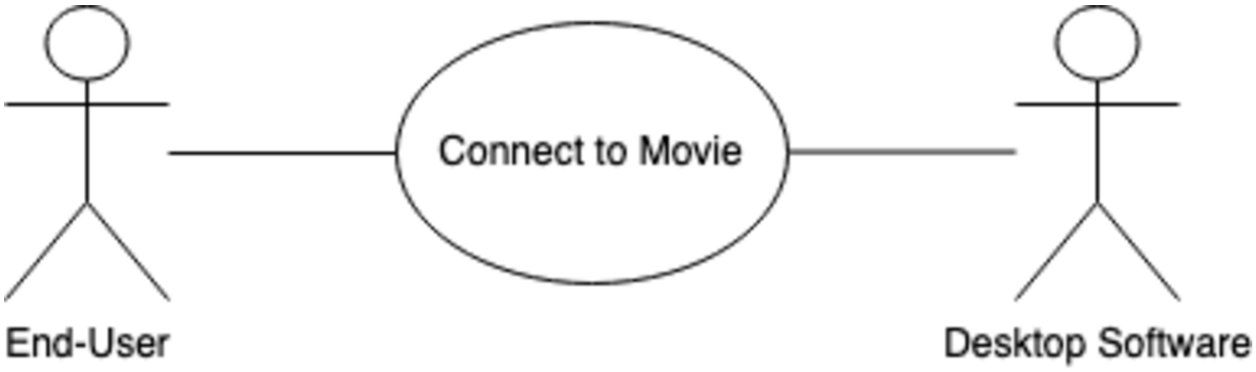


Figure 3 - Connect to Movie Use Case

Brief Description

The End-User connects to the local network and scans the given QR code on a ticket.

Initial Step-By-Step Description

Before this use case is started, the user has already paid for a ticket and has gained access to the theater network.

1. The End-User opens the Extra Ears application.
2. The End-User scans the QR code on their received ticket.
3. Local device connects to Desktop Software.
4. The system acknowledges connection and waits for the movie administrator.

Adjust Timings Use Case

Use Case: Adjust Timings

Diagram:

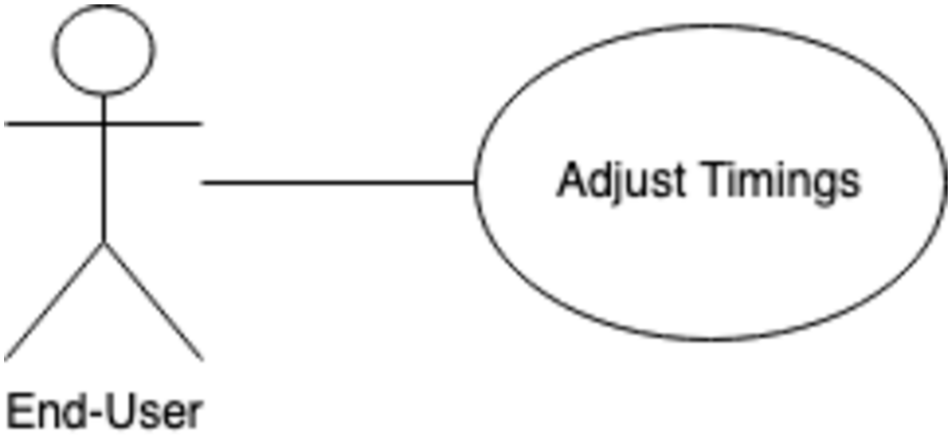


Figure 4 - Adjust Timings Use Case

Brief Description

The End-User adjusts local timings of subtitle playback during the movie.

Initial Step-By-Step Description

Before this use case is started, the End-User has successfully connected to the system as outlined in the previous use case.

1. The End-User has subtitles currently active.
2. The system presents an option to shift the playback of subtitles in two forward/backwards intervals.
3. The mobile application accepts the End-User's choice and adjusts playback accordingly.

Play Subtitles Use Case

Use Case: Play Subtitles

Diagram:

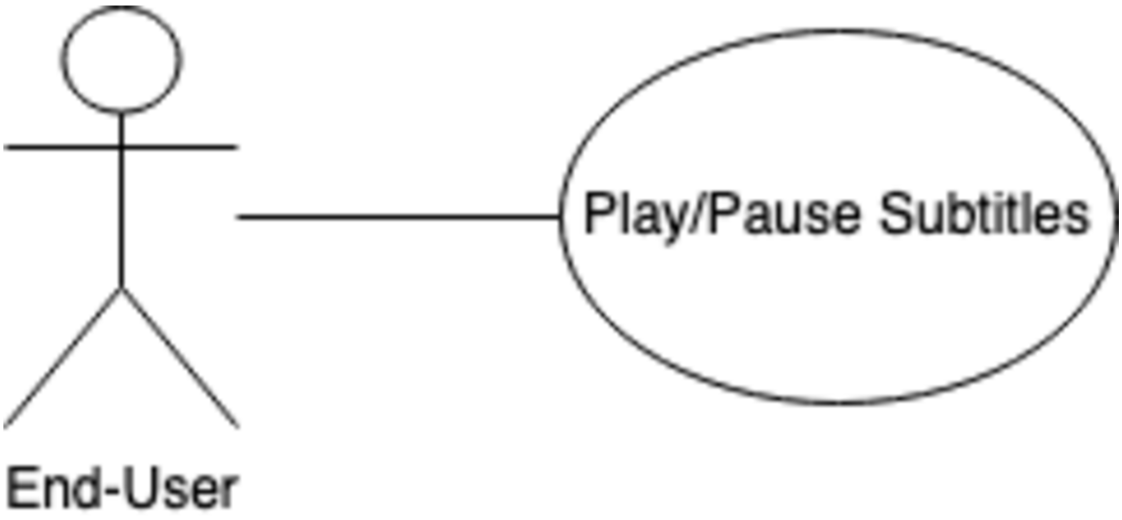


Figure 5 - Play/Pause Subtitles

Brief Description

The End-User can play/pause subtitles as desired.

Initial Step-By-Step Description

Before this use case can be started, the End-User has successfully connected to the system via the mobile application.

1. The End-User selects *Play Subtitles.*
2. The Mobile Application begins displaying relayed subtitles to the mobile application.

3. The End-User can stop playback at any time.

2.2.2 Administrator Use Cases

The Administrator has the following set of use cases:

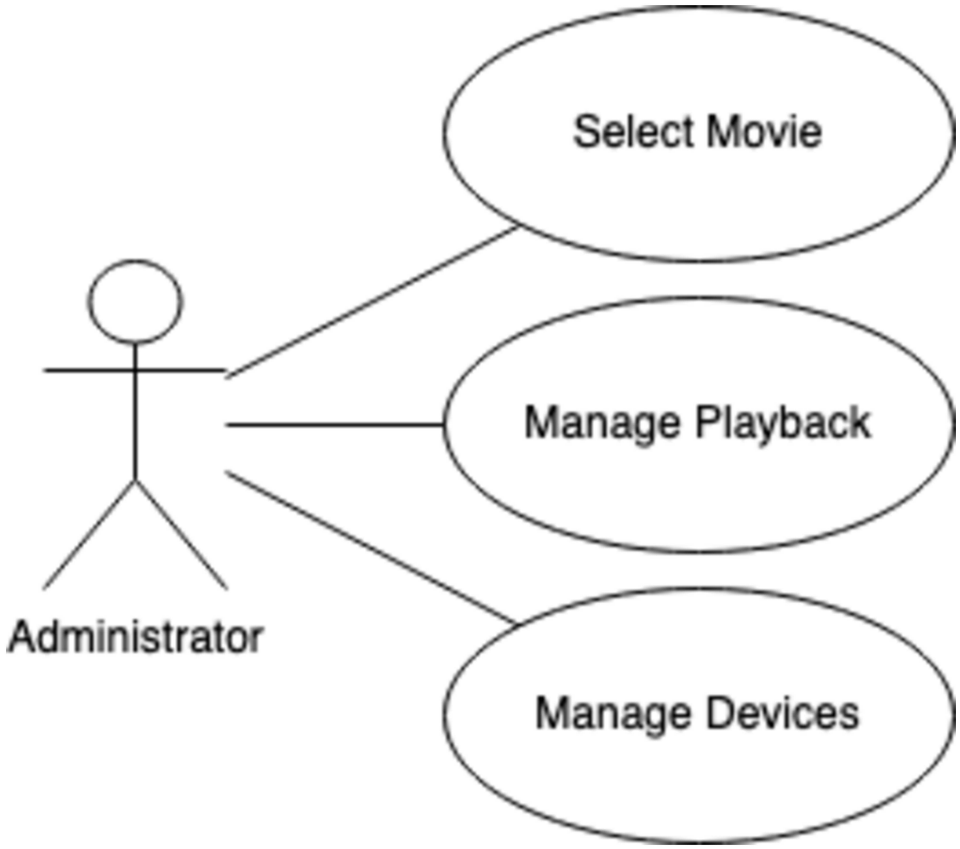


Figure 6 - Administrator Use Cases

Select Movie Use Case:

Use Case: Select Movie

Diagram:

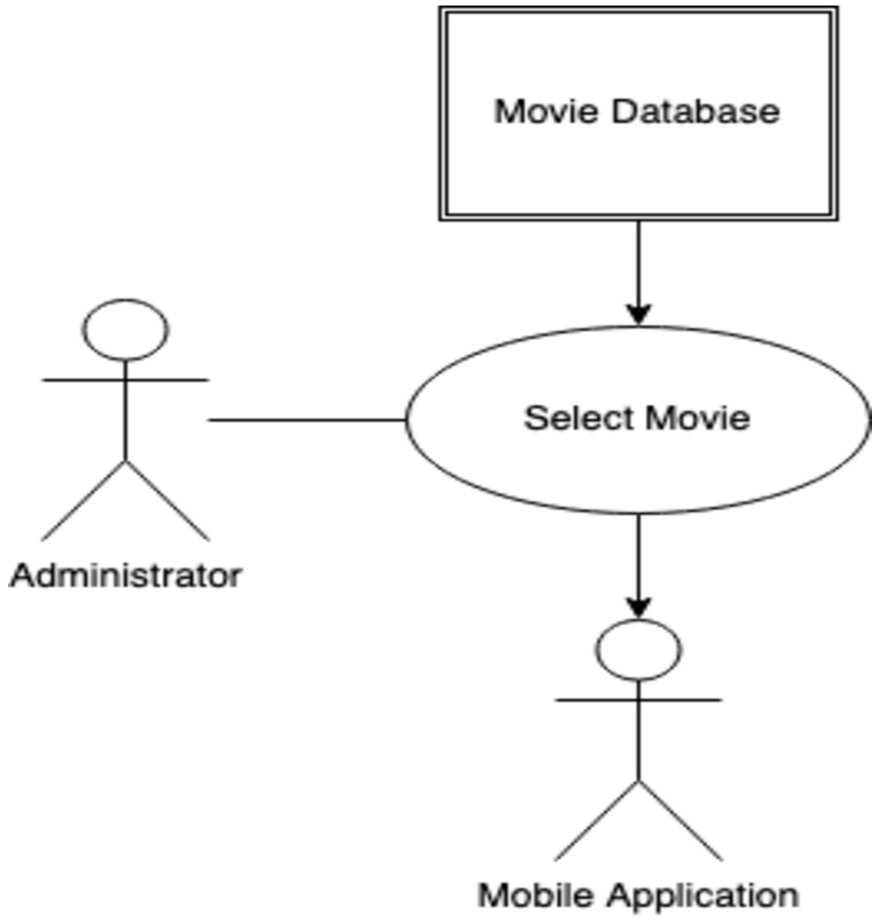


Figure 7 - Select Movie Use Case

Brief Description

The Administrator chooses from the list of available films provided by a database and queues it for use in a select theater.

Initial Step-By-Step Description

Before this step can be initiated, the Administrator has already gained access to the Desktop Application.

1. The Admin selects the correct theater.
2. Select from a list of available films.
3. The system updates to reflect the selected film.

Manage Playback Use Case:

Use Case: Manage Playback

Diagram:

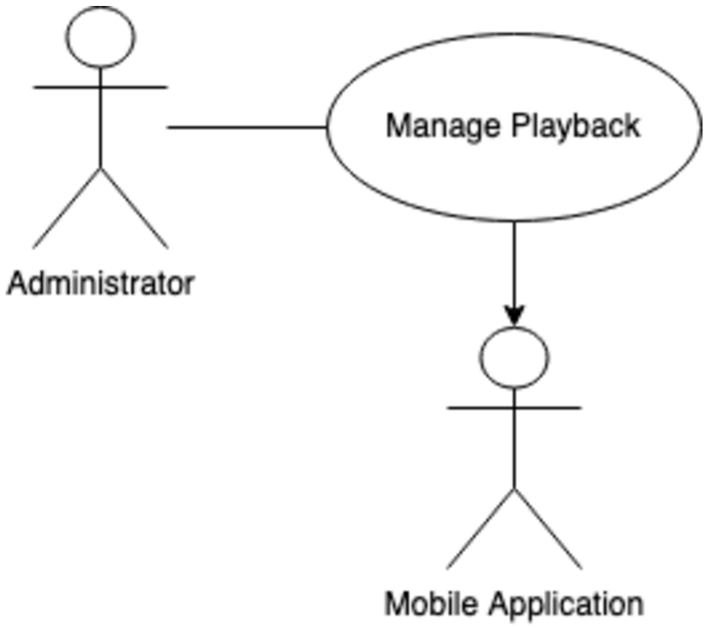


Figure 9 - Manage Playback Use Case

Brief Description

The administrator can play, pause, and alter playback of the movie alongside the subtitles.

Initial Step-By-Step Description

Before this step can be initiated, the Administrator has already chosen a movie to queue.

1. The Administrator has a film active.
2. Can select *Play/Pause* to play or pause the film/subtitles.

3. Can alter subtitle syncing to match timing of the film.

Manage Devices Use Case:

Use Case: Manage Devices

Diagram:

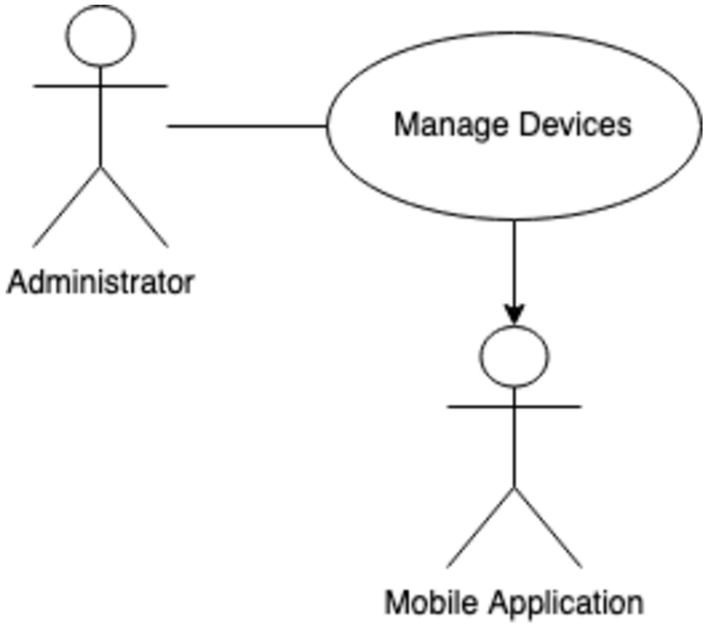


Figure 10 - Manage Devices Use Case

Brief Description

The administrator can see connected devices, their status and remove as they wish.

Initial Step-By-Step Description

Before this step can be initiated, a movie has been selected and End-User devices connected.

1. The Administrator can see a list of connected devices.
2. Hovering over a device, the Administrator can see connection status.
3. Devices can be removed if needed.

2.3. User Characteristics

The End-User is expected to have access to a mobile device with WiFi capability. Furthermore, they'll need to install the Extra Ears app to have access to the system.

The Administrator is expected to be trained on the system and have access to the Extra Ears desktop application.

2.4. Non-Functional Requirements

The Extra Ears system will compose of two parts: A desktop application and a mobile application. Both are expected to have access to a high-speed local network.

The machine used to host the Desktop command and control software is ultimately determined by the theater but must support connections to multiple network devices simultaneously. The strength of the connection/reliability is up to the strength of the local-area network (LAN).

The mobile device must be able to run React Native applications. With proper programming, the Extra Ears application will be able to run on both iOS and Android devices.[8]

[8] React Native, "Platform Specific Code."

3.0. Requirements Specification

3.1. External Interface(s)

For the Extra Ears system, the only link to an external system is the Movie Database. This database will contain relevant information about available films for the Extra Ears system. For the system, the data fields of interest will be Movie Name, Movie ID, subtitle reference and time information.

3.2. Functional Requirements

3.2.1. End-User Functional Requirements

Connect to Movie

Use Case Name	Connect to Movie
Xref	Section 2.2.1
Trigger	The End-User selects which theater they're in.
Precondition	The End-User has installed the mobile app and connected to the local network.
Basic Path	1. The End-User connects via the *2.2.1 Connect to Movie* use case. 2. The End-User is connected to the Extra Ears system and waits for the administrator.
Post Condition	The End-User waits for the movie to begin.
Exception Paths	Failed Connection to the Extra Ears system.

Adjust Timings

Use Case Name	Adjust Timings
Xref	Section 2.2.1
Trigger	The End-User taps the screen to open controls or moves to portrait orientation.

Precondition	The End-User has connected to the System and a Movie is playing.
Basic Path	1. The End-User connects to an active movie. 2. End-User taps screen OR moves device to portrait orientation. 3. The End-User can alter the timing of subtitle playback.
Post Condition	The local playback is altered.
Exception Paths	Extra Ears system disconnection/connection failure.

Play/Pause Subtitles

Use Case Name	Play/Pause Subtitles
Xref	Section 2.2.1
Trigger	The End-User taps the screen to open controls or moves to portrait orientation.
Precondition	The End-User has connected to the System and a Movie is playing.
Basic Path	1. The End-User taps the screen to open playback controls. 2. The user can select the play/pause button to stop playback of subtitles.
Post Condition	The playback has been stopped/started.
Exception Paths	Lost connection/failed connection to system.

3.2.2. Administrator Functional Requirements

Select Movie

Use Case Name	Select Movie
Xref	Section 2.2.2
Trigger	The Administrator selects the theater to begin system setup.
Precondition	The Administrator has access to the Extra Ears system.

Basic Path	1. The Administrator selects the correct theater. 2. The Administrator selects the Movie selection tool. 3. The Administrator selects the correct movie from the database and loads required content.
Post Condition	The system queues the movie data to be processed.
Exception Paths	Connection error to the movie database.

Manage Playback

Use Case Name	Manage Playback
Xref	Section 2.2.2
Trigger	The Administrator selects any of the playback tools on the UI.
Precondition	The system has a movie playing.
Basic Path	1. The administrator can alter timing. 2. The administrator can start/stop subtitle playback for users.
Post Condition	The playback was altered.
Exception Paths	Lost connection to devices.

Manage Devices

Use Case Name	Manage Devices
Xref	Section 2.2.2
Trigger	The Administrator enters the Manage Devices area of the UI.
Precondition	The system has end-user devices connected.
Basic Path	1. The Administrator enters the Manage Devices area of the UI. 2. The Administrator can remove/check the status of connected devices.
Post Condition	Connection of devices has been checked or devices removed from system.
Exception Paths	Connection error to connected End-User devices.

3.3. Detailed Non-Functional Requirements

3.3.1. Security

The Movie Database server will have restricted *write/delete* access to prevent unauthorized editing of Movie information. The only services using the Movie Database server will be the Extra Ears system with *read* access granted to it. The PC that runs the Extra Ears system will have its own security. Only a trained Administrator will have access to the machine and Extra Ears software.

The connection between Extra Ears command and control system and mobile devices will be limited on the application layer. Furthermore, the mobile application will only need access to discover devices on the local network and will not require any elevated privileges.

Implementation

This section details the technology used, processes and workflow of the entire system. Both the desktop application, server, and mobile implementation are discussed below.

1.0 Desktop Application

1.1 Front-End

As mentioned in the background, we chose to utilize JavaFX for our frontend. Utilizing IntelliJ IDEA and Scene builder, we created an easy to use and understand front end (Appendix A). To further the look and feel of the application, we also chose MaterialFX, which provides material design components to JavaFX.[9] On our interface, the administrator is presented with a main screen, containing three main parts: a theater list, currently managed movie, and connected devices. Theaters can be added via the backend, and they will populate on the main screen. Furthermore, the main screen displays current timing, selected subtitles, and allows for the admin to generate tickets to use for connecting to the Extra Ears service.

To have users connect, the administrator can generate a QR code, from the *Generate Ticket* button. The UI dynamically generates the ticket, based on the current network conditions (Appendix B, C). The admin can also export the generated ticket to the local machine, allowing for later use. Once users scan the generated QR code, they will automatically connect and appear on the right-hand side.

For managing subtitles, the application provides two main features: movie/subtitle selection, and timing management. Pressing the *Movie Management* button, administrators can

[9] https://github.com/palexdev/MaterialFX

set the active movie and corresponding subtitle. At the center of the screen, the box art is displayed, with current timestamps and play-pause and skip functionality.

1.2 Server/Backend

One of the biggest tasks was managing performance. For the front end, server, and connected clients, we designed and implemented a multithreaded application. Upon startup, the application splits into two threads: an application thread, and a server/backend thread. The server binds to the host machine's IP address as well as a designated port.[10] This is to ensure neither process within the application will 'hang' if they start processing data. For managing client performance, each connected client is designated to its own thread, and will be paused and resumed accordingly.

When the app is launched, the server binds to a port, and begins waiting for clients. Once a client scans the QR and successfully enters the Extra Ears mobile application, the server instantiates a *DataInputStream* and *DataOutputStream* for easily reading sent/received data from the socket. The server instantiates a *ClientHandler,* and passes in data streams, socket information and associated device identifiers. Next, the server designates the connected client to a *ClientHandler* class and network operations continue.

Due to the socket.io-client library we utilized in the mobile application, the server must first read for an HTTP upgrade header and respond. This is to ensure that the WebSocket used on the client side can read binary data and keep the socket open. Once this process completed, the server sends the selected subtitle file to the client and begins waiting for further commands. Now

[10] In our testing we used port 3542.

that a client is connected, the server can begin sending timing information and play/pause codes instructions.

2.0 Mobile Application

As mentioned in the background, our mobile application was written entirely in React Native. Using Visual Studio Code and the Expo CLI, we created a very clean and easy-to-use mobile application (Appendix D). In addition to the React Native framework and the extensive supply of packages and libraries it provides, the Expo CLI played a key role in the development of the Extra Ears mobile application. Not only does the inclusion of the Expo CLI provide even more libraries and features, but it also adds a native logging system and creates a very simple interface for testing the app natively, or through an emulator, on both iOS and Android devices (Appendix E).

To ensure that the application connects to the correct WebSocket, we used the Expo CLI's expo-barcode-scanner to scan the QR code created by the server which contains the WebSocket's host and port (Appendix F). Custom styling was also used to remove the QR reader from the user's view to ensure a quick, clean, and simple user experience (Appendix D).

To connect to the server's WebSocket, we used the socket.io-client library, which allows developers to easily implement various types of socket clients and automatically connect to the server. In our implementation, the client first establishes a connection with the server and sends an HTTP upgrade request. Then, it waits for and processes the incoming SRT file and waits for play/pause commands from the server.

To create intuitive navigation controls and seamless flow between pages, we chose React Native's Native-Stack-Navigator library, which provides a way to transition between screens

where each new screen is placed on top of a stack. The application will automatically navigate to the subtitle display page once the theater's QR code is scanned, and a back button is placed on the top of the subtitle display page to allow the user to navigate back and scan another theater's QR code.

To display the movie's subtitles, we used React Native's react-native-video and react-native-subtitles packages. The react-native-video package provides a way to create instances of and customize each platform's native video player to play a chosen video, or in our case display a black image for the duration of the subtitles. The react-native-subtitles package can be paired with the react-native-video package to overlay subtitles from a .SRT or .VTT file onto the video and provides many options to customize the appearance of the subtitles. We use the react-native-subtitles package to display the .SRT sent to the client device from the server (Appendix G).

Discussion

The Extra Ears system works and achieves our goal of making movie theaters more accessible to all. We successfully created a prototype application which allows theaters and clients smoothy receive subtitles for any movie they attend. Utilizing an easy-to-understand administrator console and intuitive mobile application, the team was ecstatic and proud of the prototype we created.

On the desktop/server side, the Extra Ears system performs very well. During testing, the team was able to accommodate 10+ connected devices, and with a dedicated application and server split, as well as code optimizations, the system can easily be expanded to encompass multiple theaters and hundreds of clients. To improve the overall performance, design and functionality of the system, there are two main areas of focus: the backbone and the database. For ensuring maximum performance in a real-world situation, the back end should be written in a high-performance language like C++ or Rust for maximum control and efficiency. Furthermore, a designated database that can be dynamically updated with movie info, theaters and subtitle files is a necessity but out of the scope of this prototype.

During the development of the Extra Ears desktop and mobile applications, we faced a few difficulties that required us to switch our approach. On the desktop side, originally, the team was planning to utilize more JavaScript and Electron, but ultimately shifted towards a complete Java application.[11] The team embraced the Java/JavaFX because of advantages stated earlier (extensive library support and excellent debugging and error handling) and it being the more comfortable language to work with for us.

[11] https://www.electronjs.org/

On the mobile side, we were able to successfully create a minimalistic and intuitive application that fits the needs of our stakeholders. The application loads and operates very quickly, functions stably on both iOS and Android, and passed all our usability tests. It is also able to consistently create a stable connection with the server, receive and process the movie's SRT file, and pause/play the subtitles upon the server's instruction.

One difficulty the team faced was ensuring all of the targeted users could easily use the application. Along the way, the mobile application went through many different design iterations that included login screens, extra branding, more colorful graphics, and other clutter that all distracted from the app's focus. However, after asking a few potential stakeholders for their opinions, the team quickly realized that our target demographic prefers simplicity over all else and the team landed on our current, very minimalistic, mobile design.

The biggest difficulty came about when devising a way to connect the mobile application to the server. Initially, the team had used React Native's react-native-tcp-socket package, which allows developers to create simple TCP client and server sockets. However, this package was very difficult to integrate and refused to consistently create a stable connection to the server. To solve this problem, the team decided to switch to the socket.io-client library which perfectly suited our needs. Then, the team noticed that the server would hang and reject subsequent connections once the first client had connected. Luckily, the team quickly realized that adding multithreading to the server, where each thread is designated to handle one client connection, was a fantastic solution to this problem.

Currently, the Extra Ears desktop and mobile applications are fully functional. However, the team has noted some changes they would like to make before deeming it production ready. Most importantly, the team would like to make the mobile application even more accessible by

switching to an automatic location-based connection over Wi-Fi rather than requiring users to manually scan a QR code. The team would also like to add the ability to stream the movie's audio through a Bluetooth connection to hearing aids to assist a wider range of people. Also, the team would like to provide more controls and customization to our desktop application to allow theaters to provide a more enjoyable experience for their patrons. And lastly, the team would of course like to improve the design of both the desktop and mobile applications.

Outside of theaters, this technology has an immense range of uses. Although designed specifically for theaters, the fundamental idea of synchronous captioning sent right to a mobile device is immensely applicable. From live events such as concerts, speeches to even classrooms this kind of technology is under-utilized and easy to introduce. It's important to expand access to services to anyone imaginable, which is why the team embraced Extra Ears.

Conclusion

Extra Ears aims to provide an easy-to-use solution for moviegoers with hearing impairments or those who prefer subtitles. By utilizing Java and JavaFX for the desktop application, a multithreaded server, and React Native for the mobile application, Extra Ears will be easily accessible on a wide range of platforms. Overall, the project accomplished the overarching goal of making the movie theater experience more enjoyable for all. A robust, easy to understand mobile app with little to no human input makes the client side of the experience a breeze. The desktop application incorporates an easy to understand and utilize front end, dynamically generating tickets, and simple movie and timing management. By creating an easy-to-use system, Extra Ears can make movies and other industries more accessible to all individuals.

Appendices

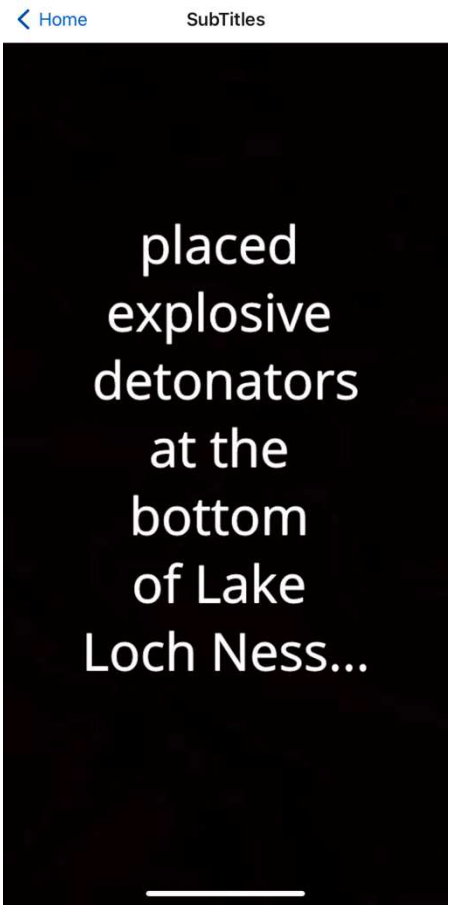
Appendix A: *JavaFX Front End utilizing MaterialFX.*

Appendix B: *Generate Ticket UI.*

Appendix C: *Generate Ticket Code.*

```java
/**
 * Generates QR Code Image
 */
2 usages  Ben Schmitt
public Image generateQRImg() throws WriterException {
    QRCodeWriter qrCodeWriter = new QRCodeWriter();
    BitMatrix bitMatrix;
    String qrCodeText = this.info;
    int width = 250;
    int height = 250;
    try {
        bitMatrix = qrCodeWriter.encode(qrCodeText, BarcodeFormat.QR_CODE, width, height);

        BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
        Graphics2D graphics = image.createGraphics();

        // Set background color to white
        graphics.setColor(Color.WHITE);
        graphics.fillRect( x: 0,  y: 0, width, height);

        // Set QR code color to black
        graphics.setColor(Color.BLACK);

        // Write QR code to image
        for (int x = 0; x < width; x++) {
            for (int y = 0; y < height; y++) {
                if (bitMatrix.get(x, y)) {
                    graphics.fillRect(x, y,  width: 1,  height: 1);
                }
            }
        }

        return SwingFXUtils.toFXImage(image,  wimg: null);
    } catch (WriterException e) {
        throw e;
    }
}
```

Appendix D: *React Native Mobile Front End*



Appendix E: *Expo CLI Command Line Interface*

Appendix F: *Expo-barcode-scanner Implementation*

```
12    const [hasPermission, setHasPermission] = useState(null);
13    const [scanned, setScanned] = useState(false);
14
15    //get permission to use camera
16    useEffect(() => {
17      const getBarCodeScannerPermissions = async () => {
18        const { status } = await BarCodeScanner.requestPermissionsAsync();
19        setHasPermission(status === 'granted');
20      };
21
22      getBarCodeScannerPermissions();
23    }, []);
24
25    //send QR code data to SubTitles.js and navigate to SubTitles.js
26    const handleBarCodeScanned = ({ type, data }) => {
27      setScanned(true);
28      navigation.push('SubTitles', {data: send});
29    };
30
31    if (hasPermission === null) {
32      return <Text>Requesting for camera permission</Text>;
33    }
34    if (hasPermission === false) {
35      return <Text>No access to camera</Text>;
36    }
37
38      return (
39          <View style={styles.container}>
40              <Image style={styles.logo} source={require("./assets/logo-color.png")}/>
41              <BarCodeScanner
42                onBarCodeScanned={scanned ? undefined : handleBarCodeScanned}
43                style={StyleSheet.camera}
44              />
45              {!scanned && <Text>Point your back camera at your theater's QR code to start</Text>}
46              {scanned && <Button title={'Tap to scan another QR code'} onPress={() => setScanned(false)} />}
47          </View>
48      );
49  }
```

Appendix G: *react-native-video and react-native-subtitles Implementation*

```
return (
    <View style={{flex: 1}}>
        <Video
            ref={videoRef}
            source={require('./assets/black.png')}
            resizeMode="cover"
            style={{flex: 1 }}
            shouldPlay={true}
            durationMillis={fileLen}
            onPlaybackStatusUpdate={status => setStatus(() => status)}
        />
        <Subtitles
            source={loadedFile}
            style={styles.subtitles}
        />
    </View>
);
```

References

Merriam-Webster. n.d. *accessible.* https://www.merriam-webster.com/dictionary/accessible.

React Native. n.d. *Platform Specific Code.* https://reactnative.dev/docs/platform-specific-code.

United Nations. n.d. *Article 9 - Accessibility.* https://www.un.org/development/desa/disabilities/convention-on-the-rights-of-persons-with-disabilities/article-9-accessibility.html.

W3C. n.d. *Accessibility.* https://www.w3.org/standards/webdesign/accessibility.

Zajechowski, Matt. 2022. *Survey: Why America is obsessed with subtitles.* 6 17. https://preply.com/en/blog/americas-subtitles-use/.