ClassTranscribe

Class Transcribe Mobile App Documentation

Matthew Walowski, Akshaya Jagannadharao, Andrew Zhang, Suxiang Han, Gary Liu

8 May 2021

Table of Contents

Project Description

Class Transcribe is a platform that students use to view lecture videos from their instructors. However, Class Transcribe currently only comes in the form of a web app. This has a few drawbacks like restrictions on downloading videos when on a mobile device. This project focuses on creating a mobile application for Class Transcribe. This application aims to allow students to find, watch, and download lecture videos for their classes.

End-User Manual

Our mobile application is targeted at students, so the current features implemented involve managing a user's account. We support browsing courses, starring courses, watching videos, checking last watched videos, and downloading videos. For instructors and course staff, we suggest using the website to manage course playlists, videos, and permissions.

Currently, only our application is only officially supported for Android devices, but it will work on iOS as well.

Run-on Physical Android Device:
You can find the latest version of our app on Expo. The link will download the most recent APK version as of 8 May 2021. You may need to enable installation from unknown sources. Follow this guide for details.

Run-on Android Emulator:

If you don't have an Android phone, you can still use our app on an emulator. Here are the required steps:

1. It will be easiest to do this all through Android Studio, so install that if you don't have it.
2. Install and run the emulator. Follow the instructions below to download the APK file onto the computer (not directly onto the emulator)
3. Then install the APK onto the emulator. Here are a few ways to do that.

Run-on iOS:
You can also run an iOS emulator with Expo (https://docs.expo.io/workflow/ios-simulator/). You need to install Xcode and then run expo. An iOS emulator would automatically be created after you choose the iOS emulator option.

Process

We mostly used XP for our development process, with a focus on frequent iterations, adapting to change, and pair programming. However, we did deviate slightly from XP as the semester went on to accommodate a remote team.

Communication

Since this semester was entirely virtual, communication was a large challenge for our team. We had members spanning across multiple time zones, so proper communication was very important to us. We had many different outlets for communication.

Slack
We utilized slack to communicate with each other asynchronously. Team members were expected to use slack for all informal communication, including progress updates, asking for help, and meeting reminders.

Pair Programming

Initially, we tried to utilize pair programming, but some pairs found this a little difficult to enforce due to differing time zones and the virtual nature of the project.

Iterations

On each iteration, we would decide which features we were going to implement as well as their tests at weekly planning sessions. These features depend on the findings and bugs from the previous iteration.

Documentation

Trello

We keep track of all user stories and update the team's work and plans each iteration. We set an expected number of hours to be completed for each task and update the cards based on where they are in the process (ex. Backlog, To Do, Doing, In Review, etc.). Once a card has been identified to be done, we comment on the actual time it took to complete the task.

Wiki

Listed the notable risks and challenges in the project alongside user documentation and notes from each weekly meeting. Also contains the project and development overview.

Development

During development, all features and changes must be made on a new branch (not master). When code is ready to be merged into master, multiple CI/CD actions will be run, and all must pass before being merged. In addition, at least 1 other team member must approve the pull request before merging. To prevent PRs from being merged before all team members have a chance to merge, we strongly encourage all developers to leave their PRs open for at least 24 hours before merging, even if they receive approval before then.

We generally enforce that all PRs must have tests, however, if a PR is rather large, we occasionally allow for the tests to be merged under a separate PR. Any refactorings that occur after a PR has been merged must be made on a new branch.

Continuous Integration and Deployment

We use multiple Github Actions for CI and CD. There are outlined by workflow file as follows:

Android.yaml
On every push to master and PR to master, creates an android build through the Expo server. The build will be located on the Expo website.

Code.yaml
On every push to every branch, runs the linter and runs the tests.

Requirements

Current Use Cases

Our application focuses on the student experience, so our use cases are structured accordingly. When determining the priority for our use cases, we emphasized replicating the existing functionality of the Class Transcribe web app, and we placed a lower priority on new features. Below is a high-level list of all user stories and use cases.

- Log in to the app with CILogon
    - This allows students to use their university's authentication provider
- Browse participating courses

- View videos in the course
- Watch video
- Pause video
- Download video
    - The video is saved to local storage so that it is persisted across phone restarts
- Change video speed
- Star a course
    - This allows users to save a course so that it is easier to access
- Search for a course

As is clear from our implemented features, we didn't create much new functionality that the web application is lacking. However, all of our features work to achieve the ultimate goal of enabling students to watch lecture videos.

Future Use Cases

Some web functionality is missing from the application. The following are use cases that would be next on the list to implement.

- Edit transcripts in the app
- Show user data statistics
- Notify students when a new video is uploaded for a starred course
- Have a settings page to view user profile
- Sync data between Class Transcribe web and mobile
- Crowdsource video processing

Architecture

Our application uses React Native with Expo and Jest for testing. Following common practice in React/Jest, we have split up much of our code into components and

containers, and all of our tests are located in the __tests__ directory. Since our code is entirely functional as opposed to class-based, we tend to use composition over inheritance, which works well with common React patterns.

Another pattern worth mentioning is the use of PropTypes within our code. Since Javascript is a dynamically typed language, it can be hard to keep track of each variable's type, especially in regards to props with React. So we require that all components must have a propTypes attribute that describes the types of all props they will receive.

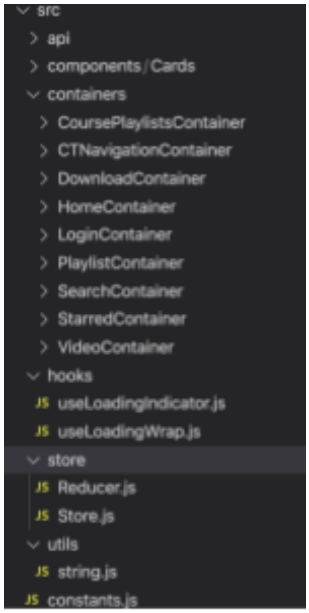

Fig 1. High level directory structure of code

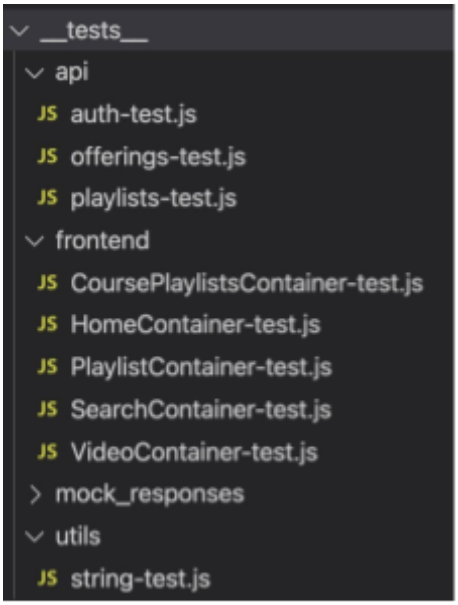

Fig 2. Structure of tests

The general structure of our application is shown above. You can find an explanation for each directory below:

__tests__

API

Contains all tests for our API wrapper. These tests use endpoint mocking to test the behavior of our application when the backend gives different responses.

frontend
Contains all tests for the front end of our application. These tests utilize React Native Testing Library to render components and interact with them.

utils
Contains all tests for utility functions.

mock_responses
Raw response data for the mocked endpoints in the **API** tests.

API

This directory contains all helper functions that interact with the CT backend server. Files in this directory are organized by the category of the endpoint that they deal with.

Components

Holds all reusable React components.

Containers

This directory contains all screens (or containers) of the application. Each subdirectory contains two files: a `.js` file for the rendering logic, and a `.style.js` file for the style sheet. Here is a brief description of each container

CoursePlaylistsContainer
Shows all of the videos within a playlist.

CTNavigationContainer
The main navigation container. This essentially acts as a router for all other containers.

DownloadContainer

Shows all downloads that a user has and allows them to play their downloaded videos. The container would scan the local download directory and extract related information from video files, such as file size and thumbnails. For each local video, the user can choose to play or delete the video file.

HomeContainer

The main home screen of the application.

LoginContainer

The login screen for the app. This will be shown if the user isn't authenticated.

PlaylistContainer

Shows all playlists for an offering

SearchContainer

Allows a user to search through all courses and star courses.

StarredContainer

Shows a user's starred offerings.

VideoContainer

Contains the video player for the application.

This video container can be used for both online videos and local videos with different parameters. It is implemented primarily based on the Expo Video component (More details: https://docs.expo.io/versions/latest/sdk/video/). Following functionalities are added to improve the user experience:

1. Icon buttons for Pause / Play / Replay / Fast Forward / Rewind
2. Adjust the video play rate
3. Download video to local

Hooks

All custom React hooks are located in this directory.

Store

Both the reducer and store for our global state are located here. Note that the store does not use Redux. Instead, it makes use of the `useContext` hook.

Utils

This directory contains various utility functions.

.github/workflows

The root of our project contains a `.github` folder with all of our Github Action workflows. We have workflows to build the project, run the test suites, run the linter, analyze code security issues, and analyze dependency vulnerabilities.

Additional Files

There is also a file named `constants.js` in the root of the source. This file contains all global constants, such as the URL of the backend.

Reflections

Matthew Walowski

I learned a lot about the process. Going into the semester, I didn't know the specific differences between Waterfall vs Agile, and I didn't know what user stories or iterations are. I also personally found the guest lecture on testing to be quite useful; whenever I'm writing tests now, I think about doing the proper setup/teardown so that there is no

dependence on ordering. From the technical side, I learned that sometimes simpler is better. When we first started the project, I initialized our repo with a very extensive boilerplate, but that ended up being much more than what we needed. And with authentication, my initial plan was much more sophisticated than the final solution.

Akshaya Jagannadharao

This project taught me a lot about mobile development. I have never used the React Native framework before, so it was an experience learning this particular technology. I have experience with the Agile methodology through my internships before, so the process was not too new to me. However, I haven't had a lot of experience contributing to open-source projects that were fairly active. Most of the projects I have contributed to had maybe one person adding features at any time. So it was a new experience contributing to a fairly active repository. I also had previous experience with CI/CD, but it is the first time that I was involved in setting up the system and integrating it into the repository.

Andrew Zhang

I learned a lot about the React Native framework and front-end UI design. Before this project, I had not been exposed to Javascript or HTML/CSS and had shallow experience in non-object-oriented programming. I had some experience with the software development cycle from CS427 but we had less time in groups for that part of the course and this course took a much deeper dive into the whole process (starting from scratch rather than building onto an existing code base).

Gary Liu

Before taking this course, I did not have any experience with front-end development and mobile development. During this course, I learned a lot about Javascript and the React Native framework, and principles for UI design. I also gained a lot of experience in software development and team collaboration. Previously, I did not have too much

experience in groups of over two people. This course helped me better understand the whole software development process from scratch.

Suxiang Han

I have attended CS427 in the previous semester and learned a lot from the previous course including the Agile process, pair programming, and testing. However, in CS428 I have extended the topic even further. As we are building projects from scratch, design patterns are really important and I found out the team's management is more important than the previous course. I also learned a lot about CI and flaky tests, which altogether inspired me on my personal project. I also learned the functional approach of React-native development, which is quite different from Component-based but has its unique advantage and shining point.

Appendix

Frontend Documentation

All of the public member functions within the Class Transcribe repository are documented with JSDoc. To generate and view documentation, follow these steps:

1. Clone the repository
2. Run `yarn install` in the root of the project
3. Run `yarn docs` in the root of the project
4. Open `index.html` in `/docs`

You can find the most recent version of the generated documentation at the end of this document.

Backend Documentation

The Class Transcribe mobile application uses the Class Transcribe web backend for all API calls. Find its documentation at https://classtranscribe-dev.ncsa.illinois.edu/swag/index.html.

Dependencies

Here is a table of all external dependencies. See `package.json` for a list of our library dependencies. All of such dependencies can be installed through yarn with `yarn install`.

Dependency Name	Version Number	URL
NodeJS	12.21.0	https://nodejs.org/dist/v12.21.0/
Yarn	1.22.0	https://classic.yarnpkg.com/en/docs/install/#windows-stable

Global

Members

(constant) addStarredOferring

Adds the given starred offering to the users list of starred offerings

Source: api/offerings.js, line 122

(constant) apiCall

Helper function to issue API calls to Class Transcribe back end

Source: api/api-requests.js, line 26

(constant) Context

The global context object for components to call useContext on

Source: store/Store.js, line 25

currentAuthenticatedUser

Object contains the following attributes: - authToken: The user's authentication token - emailId: The user's email - metaData: An object containing user preferences, such as "starredOfferings" - universityId: ID of the user's university - userId: ID of user

Source: api/auth.js, line 13

(constant) format

Formats a string where a number is replaced with the corresponding literal at that index. For example, format("HELLO

{0}{1}", "World", "!") = "HELLO World!"

Source: utils/string.js, line 8

(constant) getCurrentAuthenticatedUser

Returns the signed in user's data or null if no one is signed in. See @currentAuthenticatedUser for the attributes returned in the object

Source: api/auth.js, line 68

(constant) getDepartmentCourses

Gets the list of courses for a university's department

Source: api/universities.js, line 28

(constant) getMedia

Fetches data about a media from its ID

Source: api/video.js, line 9

(constant) getOfferingData

Gets the data for an offering from the CT API

Source: api/offerings.js, line 12

(constant) getOfferingsByStudent

Gets the data for an offering from the CT API if the student is authenticated

Source: api/offerings.js, line 21

(constant) getOfferingsData

Returns an array of all the offering data for the current user. If no user is signed in, null is returned.

Source: api/offerings.js, line 33

(constant) getPlaylistsByOffering

Fetches a list of all playlists in an offering

Source: api/playlists.js, line 9

(constant) getStarredOfferings

Gets the starred offering course IDs for the current user. If no user is signed in, null is returned

Source: api/offerings.js, line 107

(constant) getStarredOfferingsData

Returns an array of all the starred offering data for the current user. If no user is signed in, null is returned.

Source: api/offerings.js, line 78

(constant) getUniversities

Gets the list of the available universities

Source: api/universities.js, line 8

(constant) getUniversityDepartments

Gets the list of departments for a university

Source: api/universities.js, line 18

(constant) getUserHistory

Retrives the complete watch history of the current authenticated user

Source: api/history.js, line 7

(constant) getUserMetadata

Gets all metdata for the authenticted user and stores it locally. Metdata includes starred courses.

Source: api/auth.js, line 34

(constant) getVideosByPlaylist

Fetches a list of all videos in a playlist

Source: api/playlists.js, line 19

(constant) HTTP_STATUS_CODES

Commonly used HTTP status codes by name

Source: api/index.js, line 4

(constant) isUserAuthenticated

Source: api/auth.js, line 75

(constant) removeStarredOffering

Removes the given starred offering from the users list of starred offerings

Source: api/offerings.js, line 142

(constant) setAuthToken

Set the authorization token for the current authenticated user

Source: api/auth.js, line 48

(constant) setUserData

Initialize user's data using the metadata that is passed in during authorization

Source: api/auth.js, line 57

(constant) signOutUser

Terminates the current users session

Source: api/auth.js, line 82

(constant) truncateString

Truncates a string to the provided length. If the string is shorter than maxLength, nothing is done. Else, The string is cut to size maxLength and "..." is added to the end of it.

Source: utils/string.js, line 29

(constant) useLoadingIndicator

Custom hook that allows components to set a global loading indicator.

Source: hooks/useLoadingIndicator.js, line 10

(constant) useLoadingWrap

Custom hook for displaying a loading indicator while a function executes. The hook returns a function that takes another function as a parameter. The hooks returned function also returns a cleanup function for useEffect

Source: hooks/useLoadingWrap.js, line 10

Methods

CourseCard(departmentAcronym, courseNumber, courseName, courseSection, courseTerm, courseDescription, isCourseStarred, offeringId)

Component to render a single course item. Displays information about the course -- used in the HomeContainer

Parameters:

Name	Type	Description
`departmentAcronym`	String	Example: "CS" or "ECE"
`courseNumber`	String	Example: "400" or "429"
`courseName`	String	The name of the course to be displayed
`courseSection`	String	Example: "AL1" or "ADD"
`courseTerm`	String	Example: "SP 21" or "FA 19"
`courseDescription`	String	The full description of the course. Long course names will be truncated.
`isCourseStarred`	Boolean	Indicates whether the user has starred the course
`offeringId`	String	The unique ID for this offering

> Source: components/Cards/CourseCard.js, line 23

Returns:

Children of component

`CoursePlaylistsContainer(courseId, navigation)`

Renders all playlists for a given course

Parameters:

Name	Type	Description
`courseId`	String	The offering ID of the course to render
`navigation`	Object	The stack navigator object to be used. This allows the screen to launch other screens when a playlist is clicked.

> Source: containers/CoursePlaylistsContainer/CoursePlaylistsContainer.js, line 15

CoursePlaylistsView()

Wraps the CoursePlaylistsContainer so that it can receive the proper props

> Source: containers/CTNavigationContainer/CTNavigationContainer.js, line 39

CTNavigationContainer()

This is the root navigator for the entire application. Contains a starred tab, search tab, and downloads tab. Within each tab, there may be additional navigators such as a stack navigator.

> Source: containers/CTNavigationContainer/CTNavigationContainer.js, line 135

DownloadContainer(navigation)

Contains the home screen of the application. Lists starred courses and gives the user the ability to search for courses. Clicking on a course shows the playlists for it.

Parameters:

Name	Type	Description
`navigation`	Object	A stack navigator that contains a video screen

> Source: containers/DownloadContainer/DownloadContainer.js, line 19

DownloadNavigator()

The navigator for the download tab. Contains a stack navigator.

> Source: containers/CTNavigationContainer/CTNavigationContainer.js, line 110

DownloadView()

Wraps the DownloadContainer so that it can receive the proper props

Source: containers/CTNavigationContainer/CTNavigationContainer.js, line 69

HistoryContainer()

Shows the user watch history

Source: containers/HistoryContainer/HistoryContainer.js, line 15

HistoryNavigator()

The navigator of the starred tab. Contains a stack navigator.

Source: containers/CTNavigationContainer/CTNavigationContainer.js, line 76

Home(navigation)

Contains the home screen of the application. Lists courses and gives the user the ability to search for courses. Clicking on a course shows othe playlists for it.

Parameters:

Name	Type	Description
navigation	Object	A stack navigator

Source: containers/HomeContainer/Home.js, line 16

HomeNavigator()

The navigator of the home tab. Contains a stack navigator.

Source: containers/CTNavigationContainer/CTNavigationContainer.js, line 91

HomeView()

Wraps the Home container so that it can receive the proper props

Source: containers/CTNavigationContainer/CTNavigationContainer.js, line 23

`LoginContainer(onAuthLevelChange)`

Contains the log in screen. If a user is not authenticated, this screen should be shown. The login screen simply renders a web version of the CT application, and then steals the authentication token from the browser before returning to the application.

Parameters:

Name	Type	Description
`onAuthLevelChange`	function	Callback function for when the user's auth level is changed. Takes 1 boolean parameter that is true if the user is authenticated.

Source: containers/LoginContainer/LoginContainer.js, line 14

`PlaylistContainer(playlistId, navigation)`

Renders screen that shows all videos for a given playlist

Parameters:

Name	Type	Description
`playlistId`	String	The ID of the playlist to show the videos for
`navigation`	Object	The stack navigator to use

Source: containers/PlaylistContainer/PlaylistContainer.js, line 16

Returns:

Children of component

`PlaylistView()`

Wraps the PlaylistContainer so that it can receive the proper props

Source: containers/CTNavigationContainer/CTNavigationContainer.js, line 47

`Reducer(state, action)`

Reducer for the global store. Called automatically by React framework

Parameters:

Name	Type	Description
`state`	Object	The current state of the global store
`action`	Object	The action being performed

Source: store/Reducer.js, line 9

Returns:

The updated state

`SearchContainer()`

Renders the search screen for the application. This screen contains a single search bar and as the user types in a query, cards corresponding to each offering are shown. The user can star and unstar courses from here.

Source: containers/SearchContainer/SearchContainer.js, line 17

`SearchNavigator()`

The navigator for the search tab. Contains a stack navigator.

Source: containers/CTNavigationContainer/CTNavigationContainer.js, line 122

`sortOfferings(offerings)`

Sorts offerings by their department acronym

Parameters:

Name	Type	Description
`offerings`	Array	Array of offerings data from the backend

> Source: api/offerings.js, line 61

Returns:

Sorted array of offerings

`StarView()`

Wraps the History container so that it can receive the proper props

> Source: containers/CTNavigationContainer/CTNavigationContainer.js, line 31

`Store(children)`

The global store. The entire application should be wrapped in this component so that all children have access to the store. The useContext hook can be used by children in order to access the state and the reducer.

Parameters:

Name	Type	Description
`children`	Array	All the components children. The children will be rendered. This prop is passed in implicitley as the children prop typically is.

> Source: store/Store.js, line 17

Returns:

Children of component

`VideoCard(name, ratio)`

Renders information about the watch history of a video

Parameters:

Name	Type	Description

Name	Type	Description
`name`	String	The video name
`ratio`	String	A number in [0, 100] representing the percent of the video watched by the user

Source: components/Cards/VideoCard.js, line 12

Returns:

Children of component

`VideoContainer(videos, index, downloaded)`

Renders the screen where a user can view videos for a course

Parameters:

Name	Type	Description
`videos`	Object	An array of video data for this playlist
`index`	Number	The current selected video index within the playlist
`downloaded`	Boolean	True if downloaded, false otherwise

Source: containers/VideoContainer/VideoContainer.js, line 25

Returns:

Children of component

`VideoView()`

Wraps the VideoContainer so that it can receive the proper props

Source: containers/CTNavigationContainer/CTNavigationContainer.js, line 55
