How to Use this Template

1. Create a new document, and copy and paste the text from this template into your new document [Select All → Copy → Paste into new document]
2. Name your document file: "**Capstone_Stage1**"
3. Replace the text in green

GitHub Username: AndreWeiner

Simple CFD

Description

Simple Computational Fluid Dynamics (CFD) is an application to learn the basic principles of fluid mechanics and the numerical simulation of fluid flow. Several lectures are available, each one explaining an encapsulated concept. You can use Simple CFD to study only a single concept or algorithm in detail (like the SIMPLE algorithm), or, if you start from scratch, follow the suggested course order to get a comprehensive picture of CFD. The lecture materials are build on videos, written explanations or derivations and quizzes. New and updated lectures are regularly made available online and can be downloaded onto the device.

Intended User

The intended audience are undergraduate students in the fields of engineering, mathematics, physics and related subjects, but also graduates and professionals who want to strengthen or refresh their knowledge in the field.

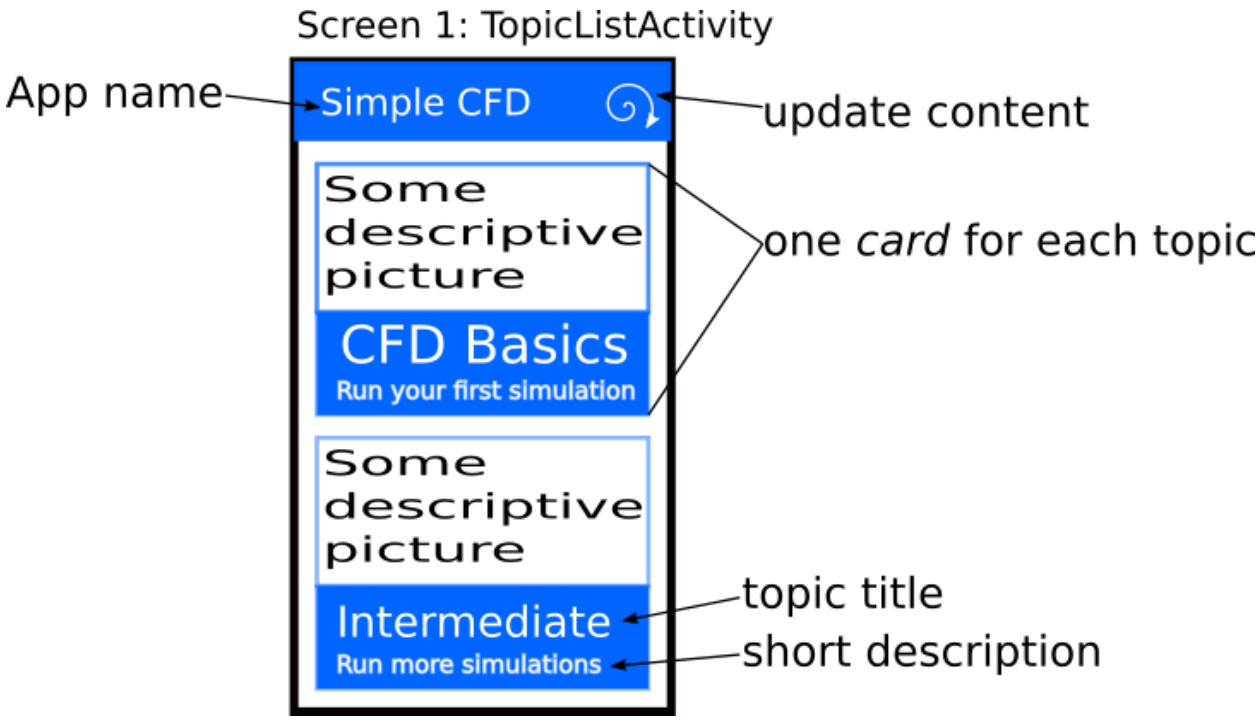
Features

Main features (this are features I would like the app to have in the future)

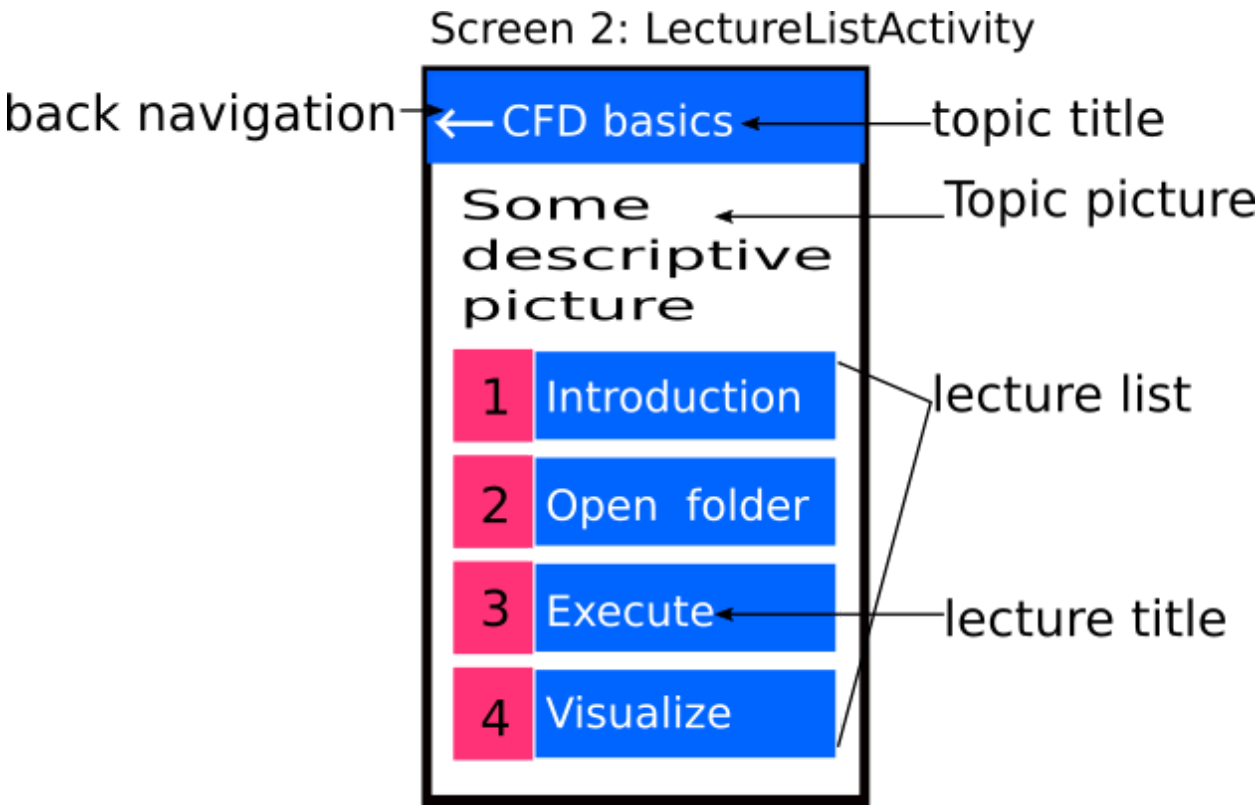
- Covers the most important topics in CFD
- High quality lecture materials including videos, derivations and quizzes
- LaTex rendered equations in written explanations
- Keeps track of your progress
- Frequently new and updated lectures

User Interface Mocks

These can be created by hand (take a photo of your drawings and insert them in this flow), or using a program like Google Drawings, www.ninjamock.com, Paper by 53, Photoshop or Balsamiq.

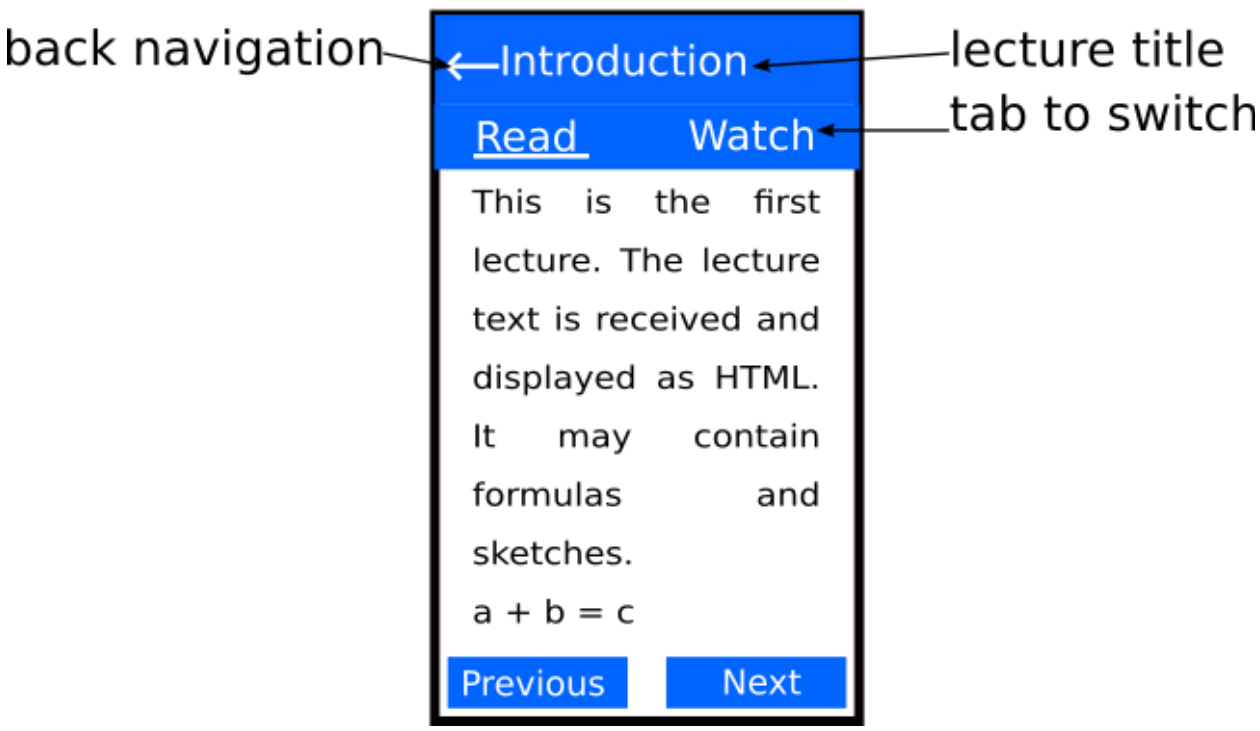


When opening the app the user is presented with a list of topics. Each topic card consists of some descriptive picture, the topic name and a short description. When selecting a topic, screen 2 opens where an overview of all lectures belonging to a topic is displayed.

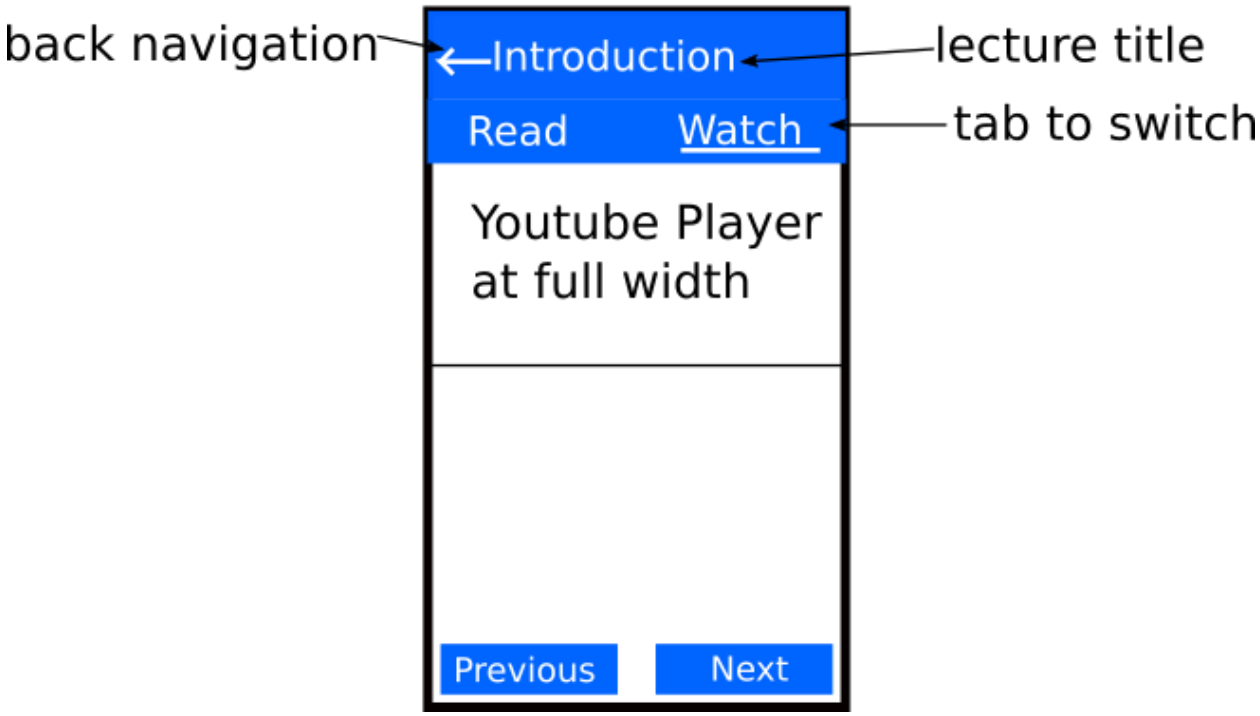


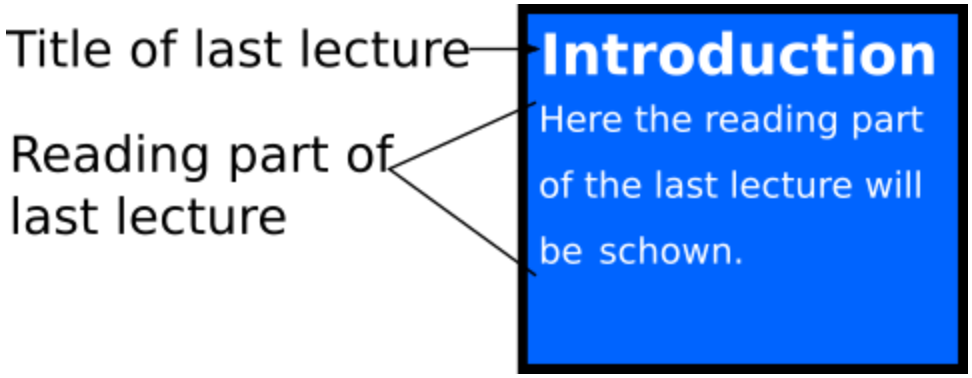
When clicking on a specific lecture in the list of lectures, screen 3 opens, where by default a written lecture is displayed. In the first version a single lecture consists of a written text and a video. The user can switch between the two types of media via clicking on the tab or swiping. Both, the lecture list and the lecture detail screen have an up-navigation. Within the lecture detail screen, the user can navigate to the previous or next lecture (if present) via buttons displayed at the bottom of the screen.

Screen 3: LectureActivity - Read tab



Screen 3: LectureActivity - Watch tab





The last part of the UI is a widget which allows the student to repeat or pick up quickly on the last lecture he or she was reading. The idea is simply to always show the reading part of the lecture the student was working on.

Key Considerations

How will your app handle data persistence?

Topics and lectures (titles, descriptions, HTML) will be stored in an Sql database. The database will be accessed via a ContentProvider. For the pictures and videos, only their IDs/URLs are stored. Consequently, videos won't be available in offline mode and picture will be replaced by a placeholder image.

Describe any edge or corner cases in the UX.

- When clicking on the prev./next button in the lecture details screen, the user will always land at the read-tab of the prev./next lecture. (the idea is to read before watching)
- The next/prev. Buttons are only displayed when it makes sense, e.g. when there is a next lecture.
- On screen rotation to horizontal mode, the video lecture will be displayed in fullscreen mode (and the video continues from the point where to video was paused).
- English will be the only language available, since it is the language of science. Therefore, there will be only an RTL layout.
- The phone with standard touch input will be the only supported platform for now. Later on, tablet support may be added.

Describe any libraries you'll be using and share your reasoning for including them.

All app components will be implemented in the JAVA language. I'll use the Android SDK version 26. Version 21 is the minimum required SDK. This is the default setting in my AndroidStudio

version 3.1.14 (on Ubuntu). The versions of Gradle and the Android plugin are 4.4 and 3.1.4, respectively.

- **Picasso** to load and display images from the web, version 2.5.2
- **Butterknife** to bind views/layouts, version 8.8.1 (including butterknife-compiler, same version)
- **YouTube Android Player API** to display videos from youtube, version 1.2.2
- **Firebase Core** to use the analytics service (my idea is to log when a topic is accessed to see their popularity), version 16.0.3

Describe how you will implement Google Play Services or other external services.

To use Google's API for Youtube, the following steps are necessary:

- Registration of the application to get an API-key (Link)
- Inclusion of API library in the Android project (Link)
- Implement/use the YoutubePlayerFragment (Link)
- A nice tutorial for all steps may be found here

To use Firebase Analytics, the following steps are necessary:

- Install the Firebase SDK in AndroidStudio (if not already pre-installed)
- Create a project in the Firebase console
- The data necessary to communicate with the Firebase server come as JSON file, which has to be included in the App folder of the application
- When Firebase is up and running one can send Log-messages, for example, to see how user use the app
- All steps outlined before are described in full detail here

Next Steps: Required Tasks

This is the section where you can take the main features of your app (declared above) and break them down into tangible technical tasks that you can complete one at a time until you have a finished app.

Task 1: Project Setup

- Create basic activity to start a new project
- Include the youtube player library
- Set up Firebase
- Set up gradle to use Picasso and Butterknife
- Implement required permissions in the manifest.xml (Internet, Internet access)
- Implement Activities

- Test application
- The app contains the following resource values: layouts, strings, dimensions, an app icon, which are all stored in the corresponding resource xml files (the icon is a drawing)

Task 2: Implement UI for Each Activity and Fragment

- Build UI for TopicListActivity (RecyclerView, Adapter, Viewholder)
- Build UI for LectureListActivity
- Build UI for LectureDetailActivity
- Build ReadFragment and WatchFragment for LectureDetailActivity
- Build the widget and widget communication

Task 3: Create dummy topic material

Topics will be available online as a JSON file, which I will host on github. The same repository will also contain image resources. The JSON file contains a list of topics (name, description, link to topic image), where each topic contains a list of lectures (name, HTML lecture text, ID of youtube video).

```
{
        "name":"CFD basic",
        "description":”Learn to run your first simulation“,
        “image_url”:”url/to/image/image.png”,
         "lectures": [
        { "name":"Introduction", to_read:”A lot of text.”, “video_id”:”xafdsaf123324” },
        { "name":"Step 2", to_read:”A lot of text.”, “video_id”:”xafdsaf123324” },
        { "name":"Step 3", to_read:”A lot of text.”, “video_id”:”xafdsaf123324” },
        ]
 }
{
...
}
```

Task 4: Implement a topic class

The Topic class will be a simple construct to bundle all information associated with a topic and its lectures. It will implement Parceable to transfer bundled topic data between activities.

Task 5: Implement retrieval and storage of topics

When the app is launched for the first time, or when the refresh button is clicked, the app will download and parse the JSON file described in 3. Similar to other projects in this course, a Utils class will implement the functionality to retrieve the JSON file and to create Topic-objects from it. The concrete classes are:

- **TopicOnlineUtils**: makes http requests, parses JSON and returns a list of Topic objects
- **TopicLoader**: implements LoaderManager.LoaderCallbacks<ArrayList<Topic>> to initilize loading from the TopicListActivity
- **TopicAsyncTaskLoader**: extends AsyncTaskLoader<ArrayList<Topic>>; allows the TopicLoader to fetch data off the main thread

To store topic data in the Sql database, to following classes will be implemented:

- **TopicDbHelper**: extends SQLiteOpenHelper; helps to create and upgrade the database
- **TopicListContract**: holds static and final variables and implements BaseColumns to define the Sqlite table
- **TopicProvider**: extends ContentProvider; implements query, insert and delete operations on the database
- **TopicCursorLoader**: implements LoaderManager.LoaderCallbacks<Cursor> to access the database from the main activity
- **TopicCursorAsyncTaskLoader**: extends AsyncTaskLoader<Cursor>; allows the TopicCursorLoader to query the database off the main thread

Task 6: Implement Widget

The last task will be to implement a widget to display the last read lecture. The implementation will be as follows:

- Class LastLectureWidget: extends AppWidgetProvider; the onReceive method will be overwritten to receive a String object holding the last lecture via an intent. The onReceive method then calls onUpdate to update the widget screen with the lecture text.
- The intent to update the widget is sent from the app via Broadcast whenever a new lecture is opened.

Task 7: Implement Firebase analytics

Analytics will be used in the following way: I want to see which topics are popular among the users. Therefore I will log the topic name whenever a topic is clicked on. In the onClick method of a topic-card in the TopicListActivity, the following log message will be sent:

```
Bundle bundle = new Bundle();
bundle.putString(FirebaseAnalytics.Param.ITEM_NAME, topicName);
mFirebaseAnalytics.logEvent(FirebaseAnalytics.Event.SELECT_CONTENT, bundle);
```

Task 7: Implement Youtube player fragment

The youtube player fragment will be committed by the LectureDetailActivity. If now internet is available, the user will be presented with a corresponding message. In order to use the Youtube Api, the Youtube App must be installed on the device. This has to be checked before initializing the player fragment. A detailed description how to implement the fragment may be found here. The player automatically starts playing when switching to the “watch tab”. In horizontal mode the player switches to full-screen.

Add as many tasks as you need to complete your app.

Submission Instructions

- After you’ve completed all the sections, download this document as a PDF [File → Download as PDF]
    - Make sure the PDF is named “**Capstone_Stage1.pdf**”
- Submit the PDF as a zip or in a GitHub project repo using the project submission portal

If using GitHub:

- Create a new GitHub repo for the capstone. Name it “**Capstone Project**”
- Add this document to your repo. Make sure it’s named “**Capstone_Stage1.pdf**”