Project 4: Full Text Search, Autocomplete, Android Application, Fuzzy Search

Due May 28 by 11:45pm **Points** 100

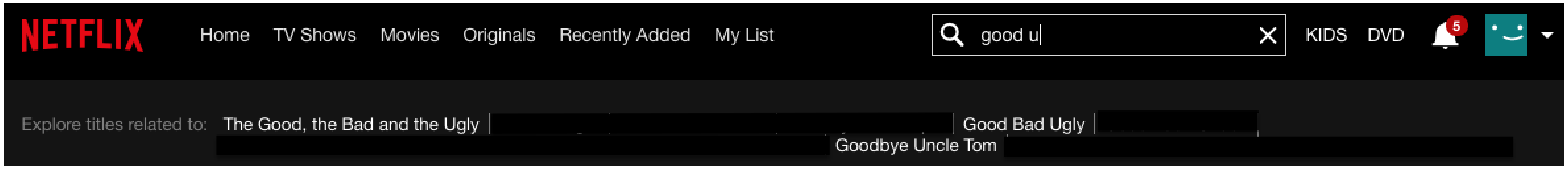
▾ Task1: Improving the Fabflix by Full-text Search and Autocomplete

The "Advanced Search" feature you implemented on project 2 is not user-friendly. Most search interfaces nowadays simply have one input box that can perform a search intelligently. Full-text Search and Autocomplete on the main search bar is now considered a must-have functionality.

Full-text Search on the movie title

Full-text Search section specifies the requirements when the users click the search button and what should show up in the movie list result.

If the query string has multiple keywords, each should be treated as a prefix. For example, the query "good u" is first tokenized into two tokens: "good" and "u". The search results will be all the movie titles that contain a word starting with "good" and a word starting with "u", such as "The Good, The Bad and The Ugly", and "Goodbye Uncle Tom". The following Netflix screenshot is an example of a full-text search.



Explore titles related to: The Good, the Bad and the Ugly | [redacted] | Good Bad Ugly | [redacted]
[redacted] Goodbye Uncle Tom [redacted]

[(https://grape.ics.uci.edu/wiki/public/attachment/wiki/cs122b-2019-winter-project4/netflix.png)](https://grape.ics.uci.edu/wiki/public/attachment/wiki/cs122b-2019-winter-project4/netflix.png)

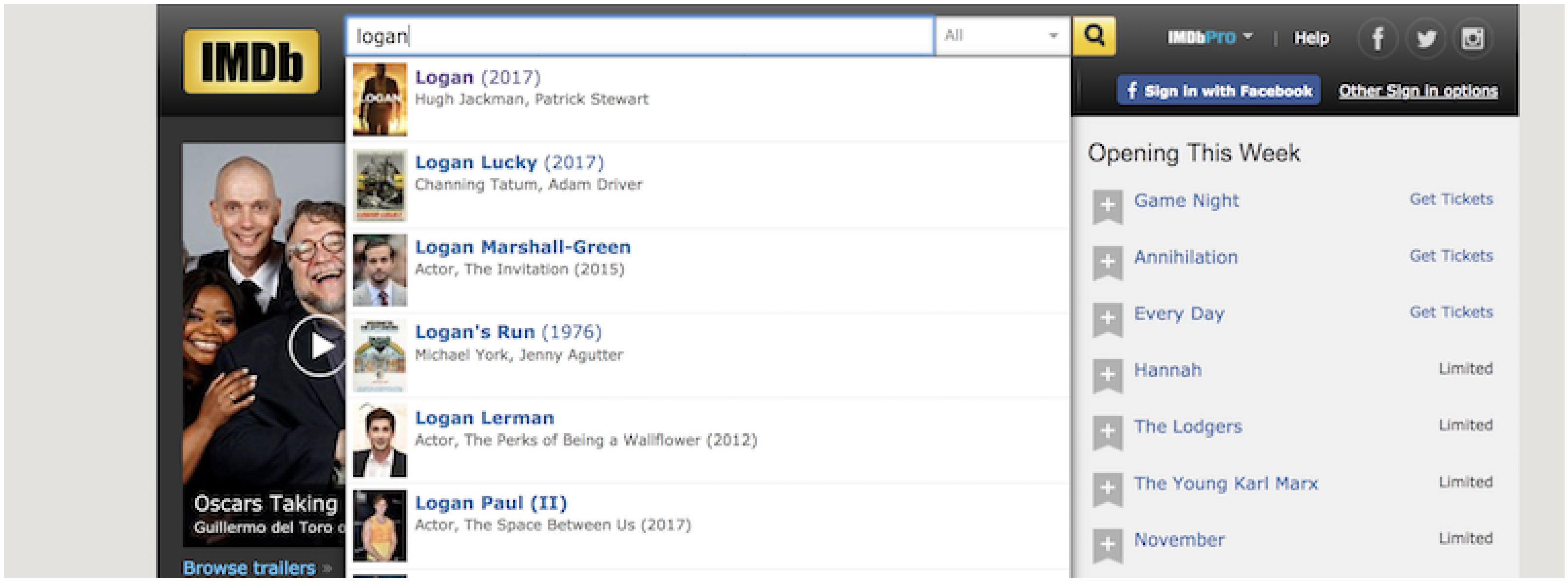
Requirements for full-text Search

- Have a main search box on the Main Page (or a separate search page) of the Fabflix website. You can reuse the search box if you already have one from Project 2.
- Implement full-text search with the query on the movie title.

- If the customers click the "Search" button, the site should jump to the Movie List Page that displays the full-text search results.

Autocomplete suggestion:

As the customers type in the query in the search box character by character, the frontend should send the query to the backend server to get a list of suggestions, then display them in the dropdown list. Check the **IMDB** (http://www.imdb.com/) search interface as an example.



(https://grape.ics.uci.edu/wiki/public/attachment/wiki/cs122b-2019-winter-project4/imdb.png)

Requirements for Autocomplete

- The Autocomplete is required to perform full-text search on movie title field.
- The Autocomplete suggestion list should not have more than 10 entries.
- The Autocomplete should support keyboard navigation using ↑ ↓ arrow keys. When a suggestion entry is selected, the entry should be highlighted, the text (query) in the search box should be changed to the entry's content (say, movie title).
- Clicking on any of the suggestion entries, or pressing "Enter" Key if an entry is selected during keyboard navigation, it should jump to the corresponding Single Movie Page directly.
- If the customer just presses "Enter" Key or clicks the search button without selecting any suggestion entry, it should perform the same full-text search action as stated above in the "full-text Search" requirement.

- When the customer types only one or two characters, it should not perform any Autocomplete search because the results may not be helpful yet. You should only perform the Autocomplete search when the customer types in at least 3 (>= 3) characters.
- When the customer types in the query, it should not perform the Autocomplete search on every keystroke because the customer is still typing. Moreover, you don't want to send too many requests to the backend at the same time. Set a small delay time (300ms) so that the frontend only performs the Autocomplete search after the customer stops typing for that delay.
- If the Autocomplete query has been sent to backend server before, it is not ideal to send the duplicate query to the backend server again (for example, when the customers delete some of the characters from the query).
  - Cache the suggestion list of each query (sent to the backend server) in the frontend, you can use LocalStorage or SessionStorage.
  - Whenever Autocomplete search is triggered, check if the query and its suggestion list are in the cache. If not, send the query to the backend server to get a new suggestion list.
- The Autocomplete search needs to be very fast. Note that the total running time the customer feels is "delay time + query time". Make sure that each Autocomplete search finishes within 2s.
- To verify that the implementation satisfies the requirements, please print logs to the Javascript console using "console.log()".
  - The logs are outputting to DevTools "Console" panel.
  - Print and only print log for the following cases:
    - The Autocomplete search is initiated (after the delay);
    - Whether the Autocomplete search is using cached results or sending an ajax request to the server;
    - The used suggestion list (either from cache or server response).

Resources:

▶ Steps on performing the full-text search on MySQL

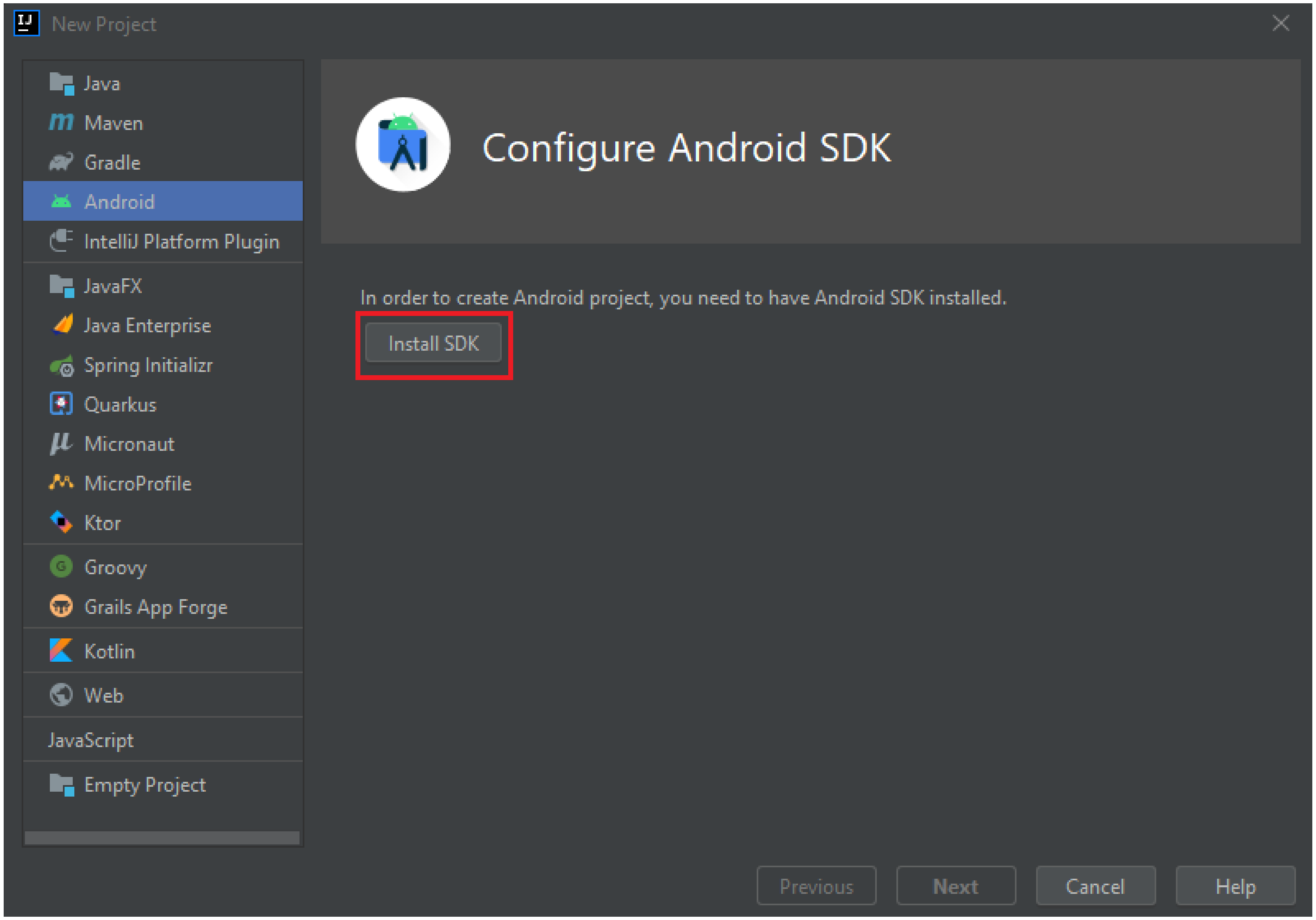
You should use a Javascript autocomplete library instead of implementing the autocomplete on your own. There are many autocomplete libraries, such as **https://github.com/devbridge/jQuery-Autocomplete (https://github.com/devbridge/jQuery-Autocomplete)**. The autocomplete library can help you easily achieve some of the requirements.

Here's an **example (https://github.com/UCI-Chenli-teaching/cs122b-project4-autocomplete-example)** that implements the most basic features of the autocomplete search.

▾ Task2: Developing an Android App for Fabflix

In this task, you will implement an Android app as a new frontend for Fabflix.

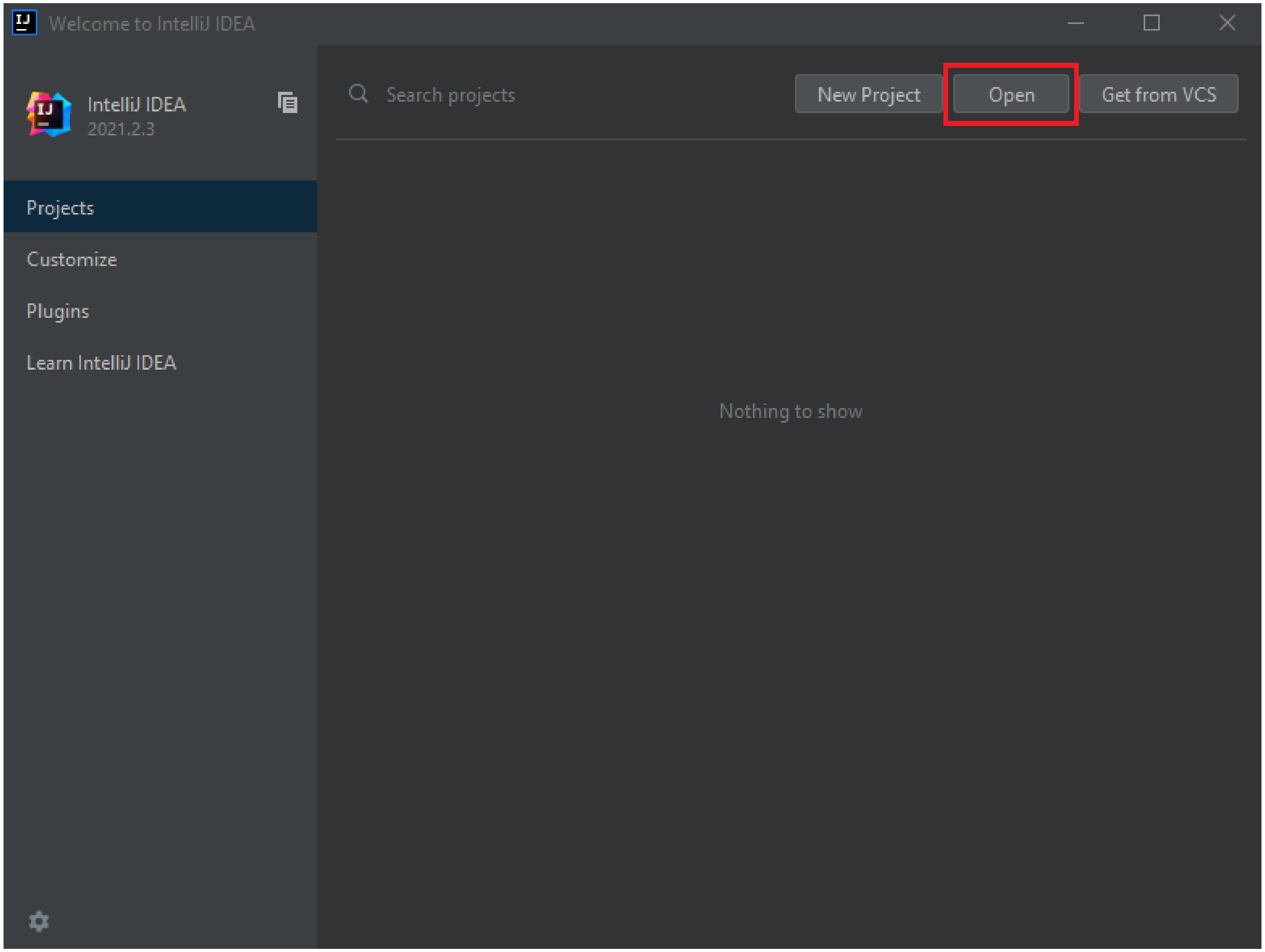
Step 1: Install Android SDK by creating an Android Project in IntelliJ-IDEA. If you haven't configured Android SDK in IntelliJ-IDEA, a button will appear for you to install it.



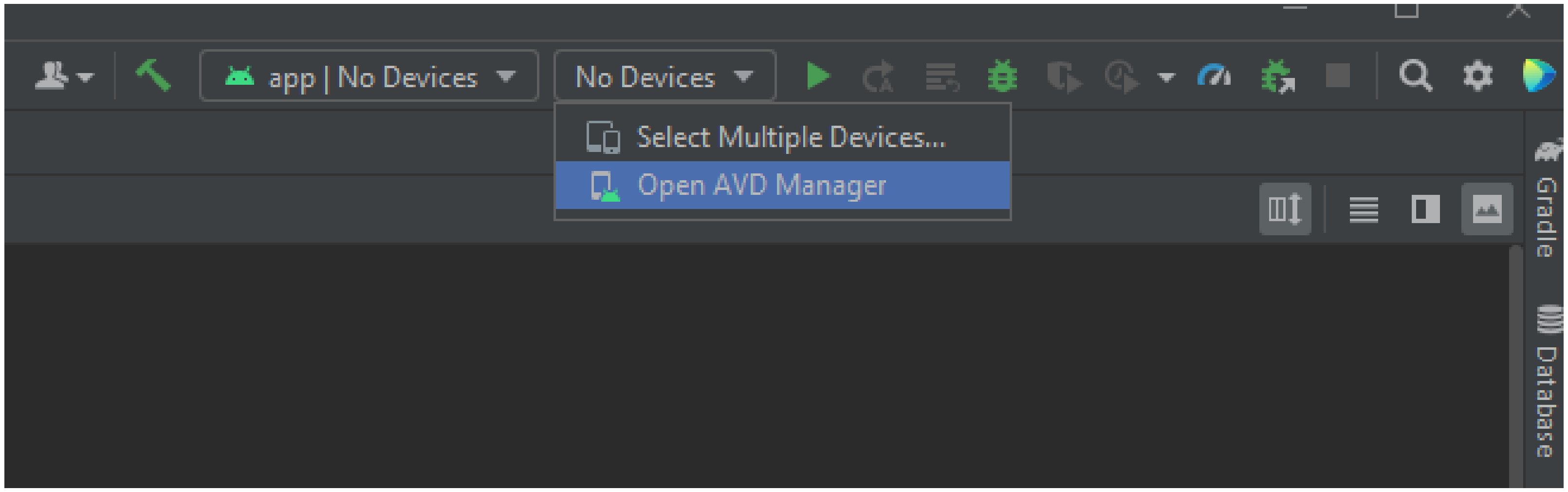
Configure Android SDK

In order to create Android project, you need to have Android SDK installed.

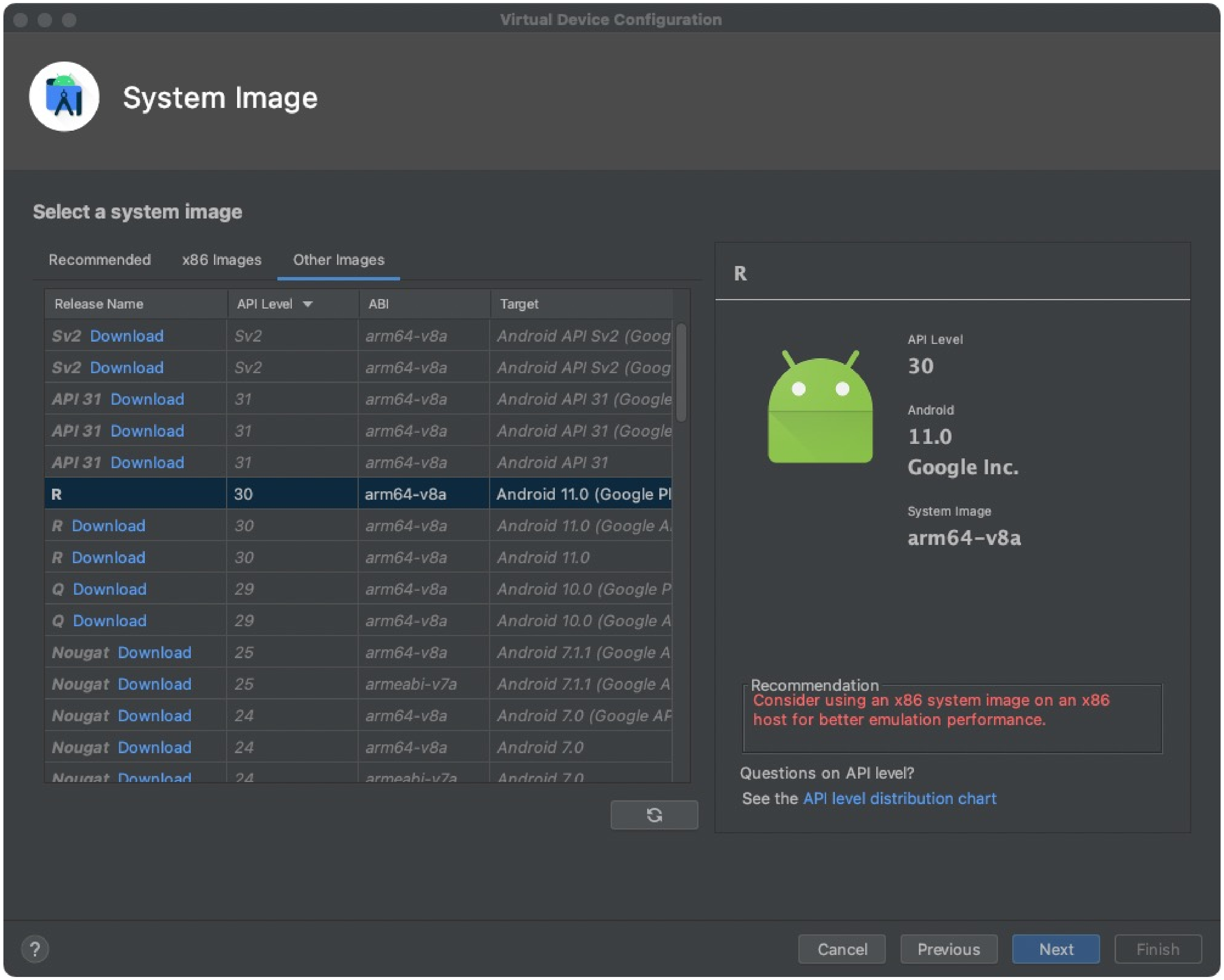
Step 2: Clone the Android Example from [cs122b-project4-android-example](https://github.com/UCI-Chenli-teaching/cs122b-project4-android-example) (https://github.com/UCI-Chenli-teaching/cs122b-project4-android-example) . Open the cloned project in IntelliJ. Wait until the Gradle finishes building.



Step 3: Setup and start a new Android emulator using the "AVD Manager" and follow the instructions on the AVD Manager.



Please select arm64 if you are using an Apple silicon (M1) chip Mac, x86 for the other user.



Select a system image

Release Name	API Level	ABI	Target
Sv2 Download	Sv2	arm64-v8a	Android API Sv2 (Goog
Sv2 Download	Sv2	arm64-v8a	Android API Sv2 (Goog
API 31 Download	31	arm64-v8a	Android API 31 (Google
API 31 Download	31	arm64-v8a	Android API 31 (Google
API 31 Download	31	arm64-v8a	Android API 31
R	30	arm64-v8a	Android 11.0 (Google Pl
R Download	30	arm64-v8a	Android 11.0 (Google A
R Download	30	arm64-v8a	Android 11.0
Q Download	29	arm64-v8a	Android 10.0 (Google P
Q Download	29	arm64-v8a	Android 10.0 (Google A
Nougat Download	25	arm64-v8a	Android 7.1.1 (Google A
Nougat Download	25	armeabi-v7a	Android 7.1.1 (Google A
Nougat Download	24	arm64-v8a	Android 7.0 (Google AP
Nougat Download	24	arm64-v8a	Android 7.0
Nougat Download	24	armeabi-v7a	Android 7.0

R

API Level
30

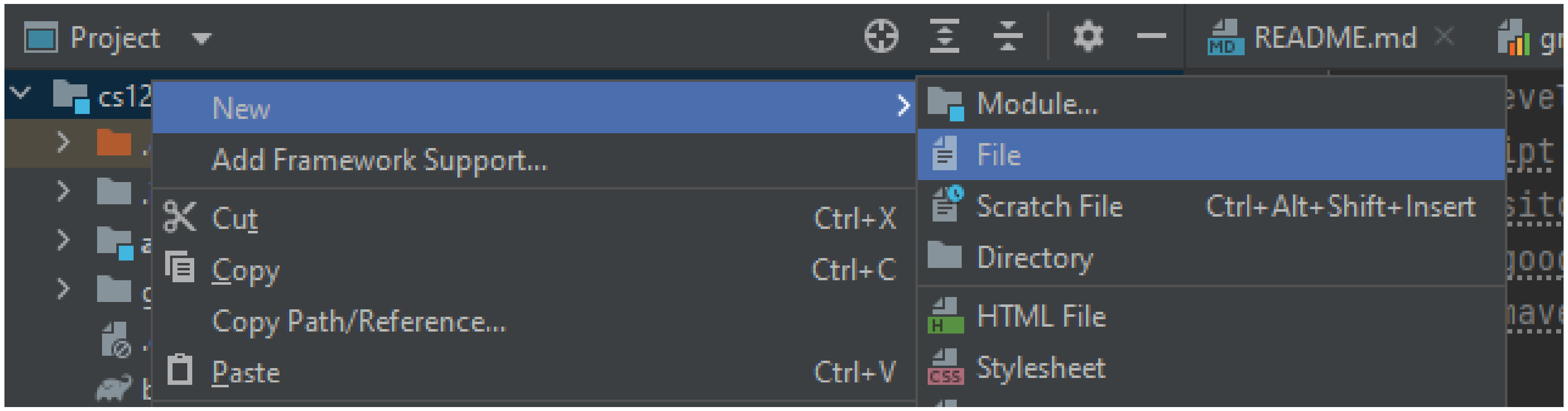
Android
11.0
Google Inc.

System Image
arm64-v8a

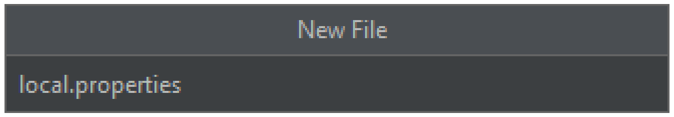
Recommendation
Consider using an x86 system image on an x86 host for better emulation performance.

Questions on API level?
See the API level distribution chart

Step 4: In the Project tool window, right-click the project root directory, select New from the context menu, and then select File.



Create a new file "local.properties".



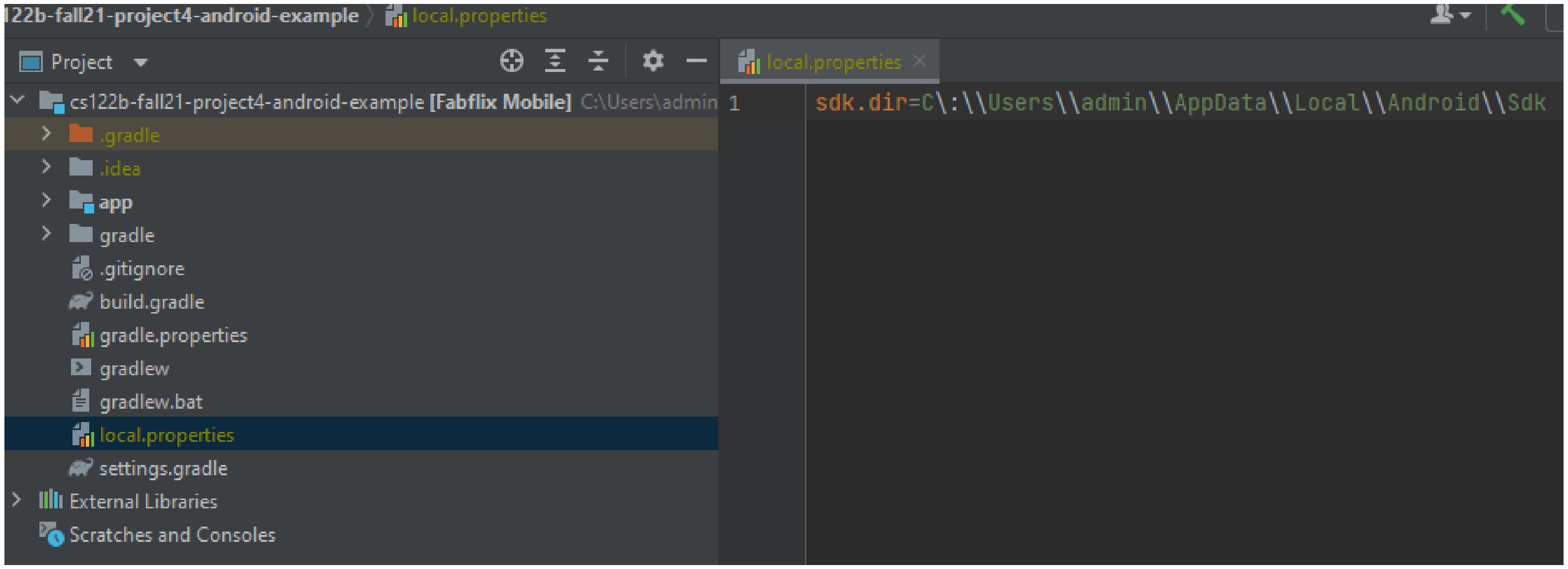
Open the file and paste your Android SDK path like below:

Windows: `sdk.dir = C:\\Users\\USERNAME\\AppData\\Local\\Android\\sdk`

macOS: `sdk.dir = /Users/USERNAME/Library/Android/sdk`

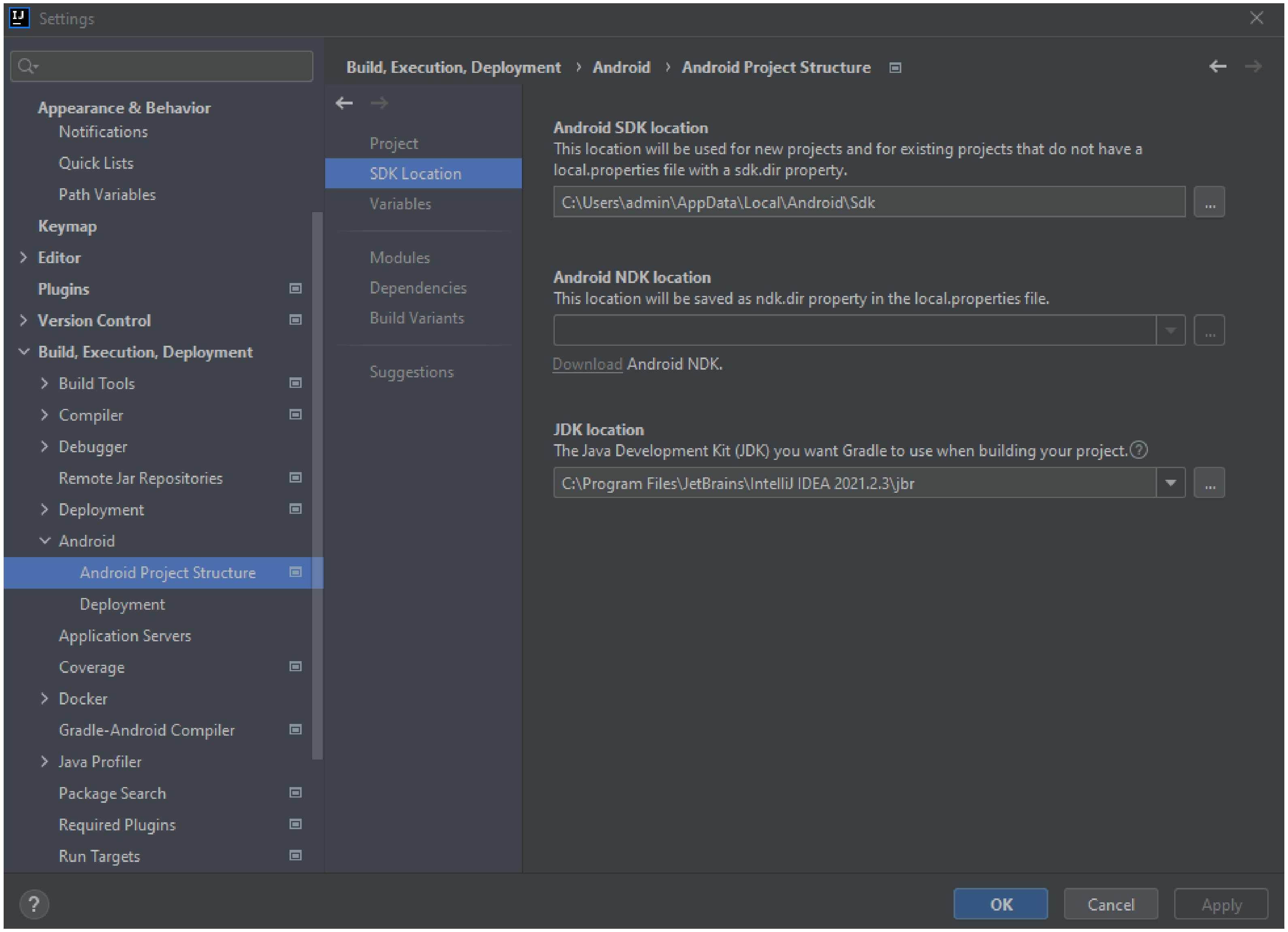
Linux: `sdk.dir = /home/USERNAME/Android/Sdk`

Replace USERNAME with your user name



```
sdk.dir=C\:\\Users\\admin\\AppData\\Local\\Android\\Sdk
```

You can also find the SDK path by clicking "File > Settings > Build, Execution, Deployment > Android > Android Project Structure > SDK Location".



Android SDK location
This location will be used for new projects and for existing projects that do not have a local.properties file with a sdk.dir property.
C:\Users\admin\AppData\Local\Android\Sdk

Android NDK location
This location will be saved as ndk.dir property in the local.properties file.
Download Android NDK.

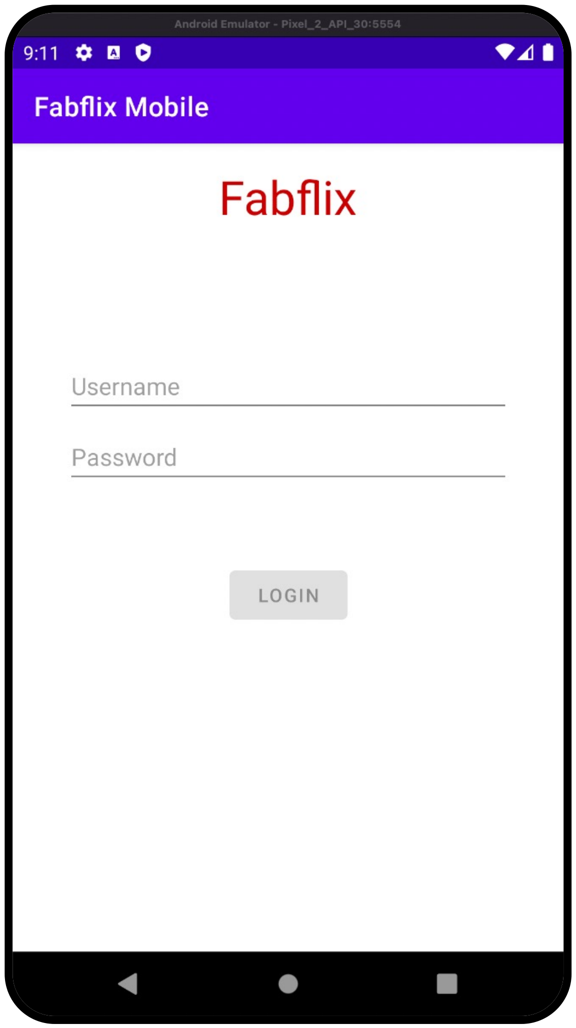
JDK location
The Java Development Kit (JDK) you want Gradle to use when building your project.
C:\Program Files\JetBrains\IntelliJ IDEA 2021.2.3\jbr

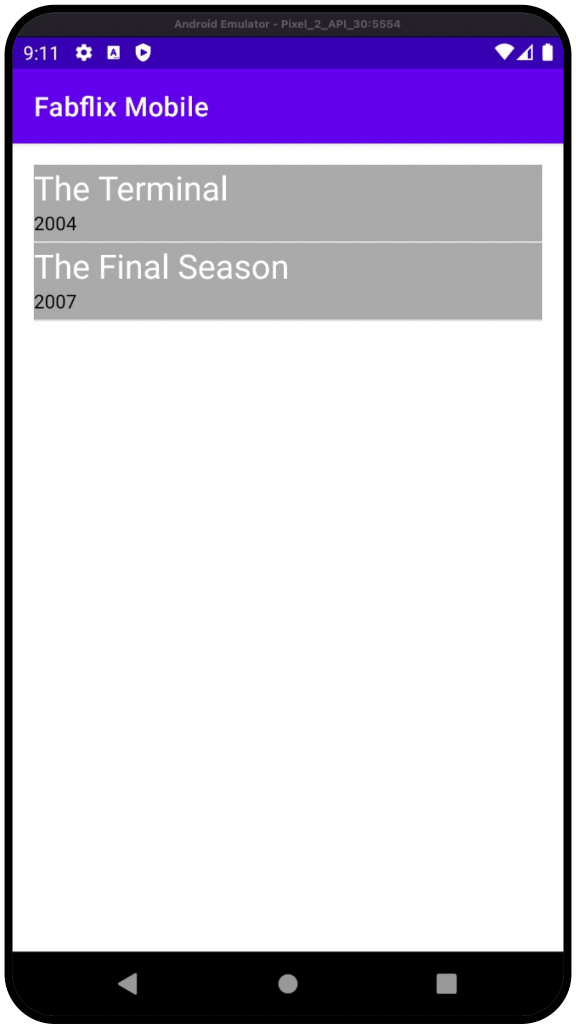
Step 5: Deploy the [cs122b-project2-login-cart-example (https://github.com/UCI-Chenli-teaching/cs122b-fall21-project2-login-cart-example)](https://github.com/UCI-Chenli-teaching/cs122b-fall21-project2-login-cart-example) on the Tomcat server.

Step 6: Run the Android app by clicking the green play button.

If you can't see the "app" option under "Modules," it means Gradle was not built properly, please wait until the loading and building are finished.

If the Android app runs successfully, it will be loaded by the emulator and display the following login page. Type in "anteater" and "123456" as the username and password. As long as the backend is running, it does **not** check these two values. The second screenshot shows the interface after login.





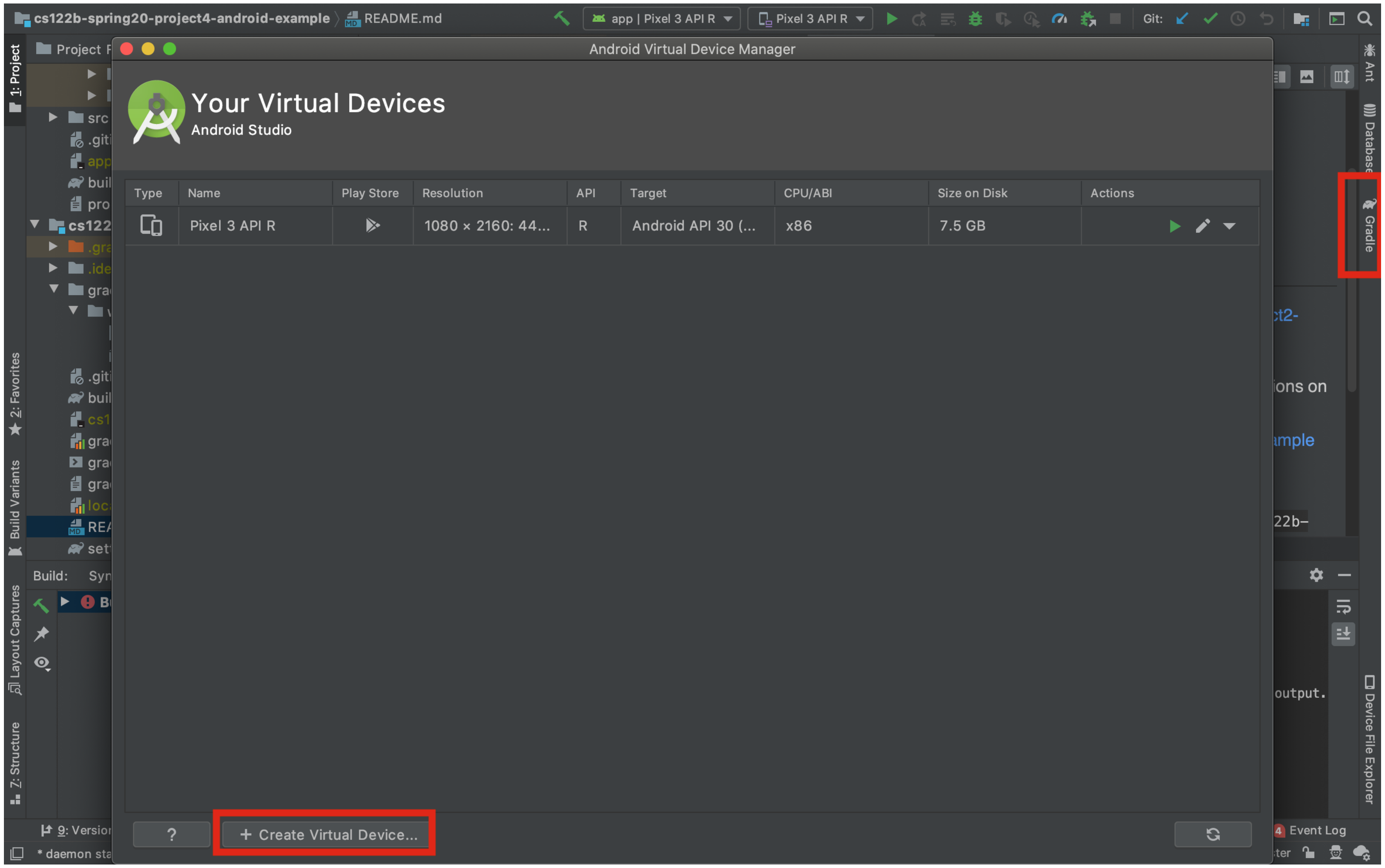
Requirements for your Android app

- Login page, which should behave like the website login page. (without `reCAPTCHA`).
  - Use HTTPS Connections to communicate with the backend server.
  - Self-signed HTTPS certificates check is disabled by the `NukeSSLCerts` class, which is invoked inside `NetworkManager`.
  - Sessions are maintained by the `CookieHandler` and `CookieManager` set in the `NetworkManager` class.

- Movie List Page
  - Displays a `ListView` of the movies searched. When a customer clicks on an item, it should show the corresponding Single Movie Page in a new activity.
  - Each item on the result list should contain the information of a movie: title, year, director, the first 3 genres (hyperlink is optional), the first 3 stars (hyperlink is optional), the same as Project 2.
  - Pagination on the search result list. `Previous` and `Next` buttons are required, and the page size is limited to 10.
- Single Movie Page
  - Single Movie Page should contain the movie title, year, director, all genres (hyperlink is optional), all stars (hyperlink is optional).
- Main Page
  - A search box that has the same behavior as the full-text search requirement in task 1 (searching in movie title). Autocomplete is optional.
- (Optional) Single Star Page
- It is **not allowed** to use the Android WebView.

Resources

- This is a good note about how different events trigger specific Android lifecycle callback functions: **https://gist.github.com/grantland/1096474/65d8dd9dd774463c2afc14764242c220fe92e051 (https://gist.github.com/grantland/1096474/65d8dd9dd774463c2afc14764242c220fe92e051)**
- If you have an Android phone, you can run the app directly on your phone, which is faster.
- If you run into an error as "Emulator: emulator: ERROR: unknown skin name 'nexus_6" then download and configure a new emulator and start it using the AVD.


cs122b-spring20-project4-android-example
README.md
app | Pixel 3 API R
Pixel 3 API R
Git:
Android Virtual Device Manager
Your Virtual Devices
Android Studio
Gradle
+ Create Virtual Device...
?
Event Log

Type	Name	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	Pixel 3 API R		1080 × 2160: 44...	R	Android API 30 (...	x86	7.5 GB	

▾ Extra Credit: Fuzzy Search Using User Defined Functions

Currently, when a FabFlix customer performs a search action, the customer must spell keywords correctly: otherwise, the desired movie will not be retrieved. For example, some star names, such as "Schwarzenegger," are sufficiently difficult to spell. Customers are very likely to spell them incorrectly, and perhaps believe that the Terminator movies are not in the stock, when, in fact, they are!

To make it easier for customers to find the movies they desire--and thus increase our likelihood of making sales--the exact string matching in Fabflix for searches is to be replaced with fuzzy search, which returns a movie whose value is close enough to the query.

Develop the fuzzy search

The designers usually take the approach that consists of SQL LIKE query and a Levenshtein (Edit Distance) Algorithm (LEDA) to implement the fuzzy search.

SQL LIKE

Here is the official link about the pattern matching: **SQL pattern matching, (https://dev.mysql.com/doc/refman/8.0/en/pattern-matching.html)** most of you have implemented this in Project 2: substring matching.

LEDA algorithm

The LEDA algorithm, which is implemented as a dynamic function in C or C++, can be accessed using the interface edth. edth is a user defined function from the **flamingo library (https://flamingo.ics.uci.edu/toolkit/)**. edth takes three parameters:

1. The first parameter is the key as a string.
2. The second parameter is the target string to be compared.
3. The third parameter is the maximum edit distance that the key string and target string can differ and still be considered similar to each other.

edth returns a Boolean value indicating whether the key string and the target string are considered similar within the edit distance.

The fuzzy search should take the union of the results from LIKE and edth.

e.g. `<name LIKE '%Schwarseneger%' />` OR `<edth(name, 'Arnold Schwrzneggar', 2) />`

Remarks for the 5 points extra credits

- Successfully implement and integrate fuzzy search into the searching functionality in Task 1 (main search and Autocomplete) and Task 2 (Android search).

Resources

- Feel free to use the User Defined Functions (described above) from this **example. (http://flamingo.ics.uci.edu/toolkit/)**
- You should carefully tune the fuzzy search edit distance threshold to make the fuzzy search result meaningful. Setting the edit distance too high might give you results irrelevant to your search query. For example, you could normalize the threshold based on the length of the query.
- Your search results should combine the full-text search and fuzzy search.

▾ Rubric

General requirements: Same as project 3.

Feature	Item	Details	Points
Full-text Search	Full-text search on the movie title	Use Full-text search on the movie title field, Jump to the corresponding Movie List Page and show correct results.	10
Autocomplete	Autocomplete UI	Up and Down keys navigation in dropdown list. Item is highlighted.	2
		Input box text is updated along with Up and Down keys navigation.	2
		Show suggestion list in 1 category: Movie (autocomplete search is on movie title).	5
		No more than 10 items in total in the suggestion list.	3
		Autocomplete delay (300ms).	2
		Enable Autocomplete only for >= 3 chars.	2
	Search	Full-text search should be implemented using AJAX (RESTful API).	10
		Cache the suggestion lists in Front-end for past queries and reuse it when possible.	5
	Jump Action	Press "Enter" Key directly or click search button without choosing any item should do a normal Search (Full-text search).	3
		Click on a suggession entry will jump to corresponding Single Movie Page.	3
		Use the Up/Down arrow key and press the "Enter" Key on a suggestion entry to jump to the corresponding Single Movie Page.	3
	Javascript Console Log	Output a javascript console log when an Autocomplete query is initiated.	1
		Output a javascript console log to differenciate if the suggestion list is coming from Front-end cache or Backend server.	2

		Output a javascript console log of the used suggestion list (in a javascript array).	2
	Reasonable speed	Autocomplete should finish within a reasonable amount of time (delay + query time <= 2 secs).	5
		reCAPTCHA still works properly on website.	3
	Login	Display login error messages.	3
		Login with correct credentials.	4
	Search	Full-text search on movie title field (Autocomplete is optional).	10
Android		Show search results: a list view of movies with title, year, director, first 3 genres, first 3 stars (see Movie List Page), sorting is optional.	5
	Movie List Page	Tap on a movie should jump to the corresponding Single Movie Page.	2
		Pagination of search results, 10 items per page.	2
		Next/Prev button to nevigate between result pages.	6
	Single Movie Page	Display information of a movie (title, year, director, all stars, all genres).	5
	Support fuzzy search using UDF	Fuzzy search works for main web search box.	2
Extra credit		Fuzzy search works for web autocomplete.	1
		Fuzzy search works for Android search.	2
Total			100+5=105

▾ Demonstration

In summary, we require 2 things for the demo:

1. A screen recording demo video showing your web application working on AWS and

Android in a local emulator (version: Android version 33, emulator Pixel XL) connected to the AWS backend.

2. Leave an AWS instance running for a week.

We will test your Android App with the same emulator mentioned above against your AWS backend.

Screen recording demo video

- **Please make sure your video is readable. Enlarge your font (terminal, browser) and make sure your video is not compressed.**
- The entire demo should be on AWS. **NO** local demo allowed.
- You need a terminal connects to AWS, and a web browser open to tomcat manager page on AWS (https://<AWS public IP>:8443/manager/html).
- Use screen recording tools (e.g, QuickTime Player for macOS and Open Broadcaster Software for Windows, or other equivalents)
- Please keep the length of the video within 15 minutes. However, do **NOT** edit the video in any way. Otherwise it could be treated as cheating.
- Upload the video to Youtube or other public media hosting website, then include your video URL (publicly accessible) in README.md, submit on GitHub.
- Preparation before the demo:
    - ssh onto AWS instance.
    - on AWS instance, prepare the MySQL database `moviedb`, load data (movie-data.sql) properly, you can remove any data you added before in P3 (from XML or from dashboard).
    - open your IntelliJ locally, with an Android emulator started.
    - remove your git repo from AWS instance, demo should start with a fresh cloned repo.
    - make sure MySQL and Tomcat are running on AWS instance.
- Demo on AWS instance:
    - **Commit check**:
        1. git clone your git repo to AWS instance: `git clone <repo url>`. This step is to make sure you have a clean repo.
        2. cd into your repo and run `git log`, show your latest commit.
        3. fail to show commit check will result in an instant 0 of project.

- **Deploy your web application on AWS instance**:
  1. inside your repo, where the pom.xml file locates, build the war file:
     ```
     mvn clean package
     ```
  2. show tomcat web apps, it should NOT have the war file yet:
     ```
     ls -lah /home/ubuntu/tomcat/webapps
     ```
  3. copy your newly built war file:
     ```
     cp ./target/*.war /home/ubuntu/tomcat/webapps
     ```
  4. show tomcat web apps, it should now have the new war file:
     ```
     ls -lah /home/ubuntu/tomcat/webapps
     ```
- **Show your website in your web browser:**
  - open browser Devs Tools **Console Panel** (https://developers.google.com/web/tools/chrome-devtools/console) .
  - login as a customer into the Main Page, which has access to the main search bar.
  - enter **"lo"** in the main search bar, hit Enter key directly, a normal full-text search should perform and jump to Movie-List Page, the autocomplete should not be triggered since only two characters are provided.
  - enter **"s lov"** in the main search bar, a dropdown suggestion list should show your autocomplete suggestions, **use up/down arrow keys to navigate to the third suggestion**, hit Enter key, the corresponding Single Movie Page should be redirected.
  - enter **"s lov"** in the main search bar, a dropdown suggestion list should show your autocomplete suggestions, **use your mouse to click the last suggestion**, the corresponding Single Movie Page should be redirected.
  - quickly enter **"love"** in the main search bar (this is to test your autocomplete delay), only one query should be sent to the backend, then delete **"e"** from the main search bar, then reenter **"e"**.
  - demo your fuzzy search result if you have implemented it.
- **Show your Android in the local emulator:**
  - connect your Android App to AWS address instead of localhost.
  - show `git status` and `git log` on your local report to ensure the same version is used.
  - build and deploy the APK on your local emulator (version: Android version 33, emulator Pixel XL).
  - go to the login page
    - enter email/account **"a@email.com"** with a wrong password. click login, an error message should display.
    - enter the correct password **"a2"** for the same email/account. click login, it should succeed and redirect to Main Page.
  - in Main Page, enter **"s lov"** in the main search bar, hit Enter key, a Movie-List Page should display.
  - on the Movie List Page, navigate to the second page, select the first movie and jump to the Single Movie Page.

    - demo your fuzzy search result if you have implemented it.
- Here is a **sample (https://youtu.be/lVLmsHpFekk)** demo video.

▾ Submission

Same as Project 3, but please update README with the Project 3 content.

If you select to do the extra credits, include a brief explanation of the design and the implementation of your fuzzy Search in README.