Titicaca InfluencerBoard Final Report

Part 1: User Stories

1. As a user, I want an entry page so that I can login to my own home page. My conditions of satisfactions are:
    a. I should know clearly that the entry page is for the influencer board app.
    b. I should be able to login without spending time creating a new account.
2. As a user, I want a home page so that I can see my personalized information and recommendations. My conditions of satisfactions are:
    a. I should be able to see some of my basic information, such as my name, my subscriptions, my interests, etc.
    b. I should be able to see the influencers recommended for me based on my own interests or randomly if I'm new to the app.
3. As a user, I want to see some basic information of the recommended influencers on my home page so that I can choose the one I'm interested in more directly. My conditions of satisfactions are:
    a. I should be able to see a cover photo of each influencer that best represents their content and style.
    b. I should be able to see the number of followers of that influencer to know how popular they are.
    c. I should be able to see a precise description of each influencer. For example, a list of their popular channel tags would be great.
    d. I should be able to follow an influencer easily by clicking on a follow button.
    e. I should be able to see more information about that influencer by clicking on a button similar to a "more info" button.
4. As a user, I want to have a profile page of my own so that I can keep track of my personal info and following channels. My conditions of satisfactions are:
    a. I should be able to go to my own profile page from the home page easily by clicking on a button similar to a "my profile" button.
    b. I should be able to see my personal information on my profile page, such as my name.
    c. I should be able to see all the channels that I'm currently following on the profile page and be directed to those channels easily.
    d. I should be able to go back to my homepage easily by clicking on a "back to home" button.
5. As a user, I want to see a more detailed influencer page so that I can have more information about them. My conditions of satisfactions are:
    a. I should be able to access an influencer's home page easily from my own home page.

b. I should be able to see that influencer’s name, country, number of followers, tags, and a more detailed description on their home page.
c. I should be able to see some popular videos under their channel and should be able to play those videos on the influencer’s page.

Part 2: Test Plan and Execution

In this part, we list all the major classes we’ve tested and the equivalence classes (some with boundary conditions) associated with them.
Link to our git repo: https://github.com/LucassLin/CS4156TeamProject/tree/main/src/test/java

GetChannelAnalytics

getInfluencers

This class is supposed to be able to return an array of Influencer objects given the provided number requirement, which should be equal to or greater than zero.
Prototype: `public ArrayList<InfluencerProfile> getInfluencers(int num)`
Equivalence class A (invalid) → num < 0 → `getInfluencersTestA()`
    Normal case: num = -1
Equivalence class B (valid) → num >= 0 → `getInfluencersTestB()`
    At the boundary case: num = 0
    Immediately outside the boundary case: num = -1
    Normal case: num = 6

InfluencerBoardResource

This class contains most of the methods that return the frontend view for our endpoints. We’ll test each endpoint thoroughly.

getWelcome

This method returns the welcome page view for our signin endpoint.
Prototype: `public LoginView getWelcome()`
Test: `getWelcomeTest()`

getHomeForUser

This method returns the UserHomeView object for our user home endpoint.

Prototype: `public UserHomeView getHomeForUser(@PathParam("name") String name, @PathParam("email") String email, final @Context ResourceContext resourceContext)`
Equivalence class A (invalid/ outside the class) → the profile has a null email → `getHomeForUserTestA()`
Equivalence class B (valid/ inside the class) → the profile has both valid name and email value → `getHomeForUserTestB()`

addLikeRecord

This method adds a new record to the LikeRecord table through a POST request.
Prototype: `public void addLikeRecord(@PathParam("channelId") String channelId, @PathParam("email") String email)`
Equivalence class A (invalid/ outside the class) → either channelId or email has a null value → `addLikeRecordTestA`
Equivalence class B (valid/ inside the class) → valid channelId and email value → `addLikeRecordTestB`

deleteLikeRecord

This method deletes a record from the LikeRecord table through a POST request.
Prototype: `public void deleteLikeRecord(@PathParam("channelId") String channelId, @PathParam("email") String email)`
Equivalence class A (invalid) → either channelId or email has a null value → `deleteLikeRecordTestA`
Equivalence class B (valid) → try to delete a record that does not exist → `deleteLikeRecordTestB`
Equivalence class C (valid) → try to delete a record that exists → `deleteLikeRecordTestC`

getFollowing

This method returns a FollowingView object that returns the frontend view of our userProfile endpoint.
Prototype: `public FollowingView getFollowing(@PathParam("name") String name, @PathParam("email") String email)`
Equivalence class A (invalid) → either name or email has null value → `getFollowingTestA`
Equivalence class B (valid) → both parameters are valid → `getFollowingTestB()`

getInfluencerForUser

This method fetches detailed information of influencers and corresponding most popular video and returns to the front end for rendering.
Prototype: `public InfluencerProfileView getInfluencerForUser(@PathParam("channelId") String channelId)`
Equivalence class A(valid) → valid channelID → `getInfluencerForUserTestA`
Equivalence class B(invalid) → invalid channelID → `getInfluencerForUserTestB`

LikeRecordDAO

This class represents the interface to access the LikeRecord table. We'll test its insertion and selection methods.

Create

This method inserts a new record into the LikeRecord table if the record doesn't exist.
Prototype: `public LikeRecord create(LikeRecord record)`
Equivalence class A (invalid) → try to insert a record that already exists → `createLikeRecordTestA()`
Equivalence class B (valid) → try to insert a new record → `createLikeRecordTestB()`

findAll

This method finds all the records corresponding to one email address from the LikeRecord table.
Prototype: `public List<LikeRecord> findAll(String email)`
Equivalence class A (invalid) → a null email is passed in → `findAllTestA()`
Equivalence class B (valid) → the email passed in does not exist in the table → `findAllTestB()`
Equivalence class C (valid) → the email passed in exists in the table → `findAllTestC()`

deleteRecord

This method deletes a record from the LikeRecord table.
Prototype: `public int deleteRecord(String email, String channelID)`
Equivalence class A (invalid) → null parameter is passed in → `deleteRecordTestA()`
Equivalence class B (valid) → a record that doesn't exist is passed in → `deleteRecordTestB()`
Equivalence class C (valid) → a record that exists is passed in → `deleteRecordTestC()`

UserProfileDAO

This class represents the interface to access the UserProfile table. We'll test its insertion and selection methods.

createUser

This method allows us to insert a user into the user profile table, where email and name are required.
Prototype: `public UserProfile createUser(UserProfile profile)`
Equivalence class A (invalid) → user with null email or name → `createUserTestA()`
Equivalence class B (valid) → user with valid email and name value → `createUserTestB()`

getAllUsers

This method allows us to get all the users stored in the table, no arguments are needed
Prototype: `public List<UserProfile> getAllUsers()`
Test: `getAllUsersTest()`

Search

This class contains methods calling youtube api and retrieve data we need.

getInfluencerProfile

This method retrieves details of a specific youtube channel.
Prototype: `public InfluencerProfile getInfluencerProfileByID`
Equivalence class A (invalid) → invalid channel id → `public void getInfluencerProfileInvalidTest`
Equivalence class B (valid) → valid channel id →`public void getInfluencerProfileValidTest`

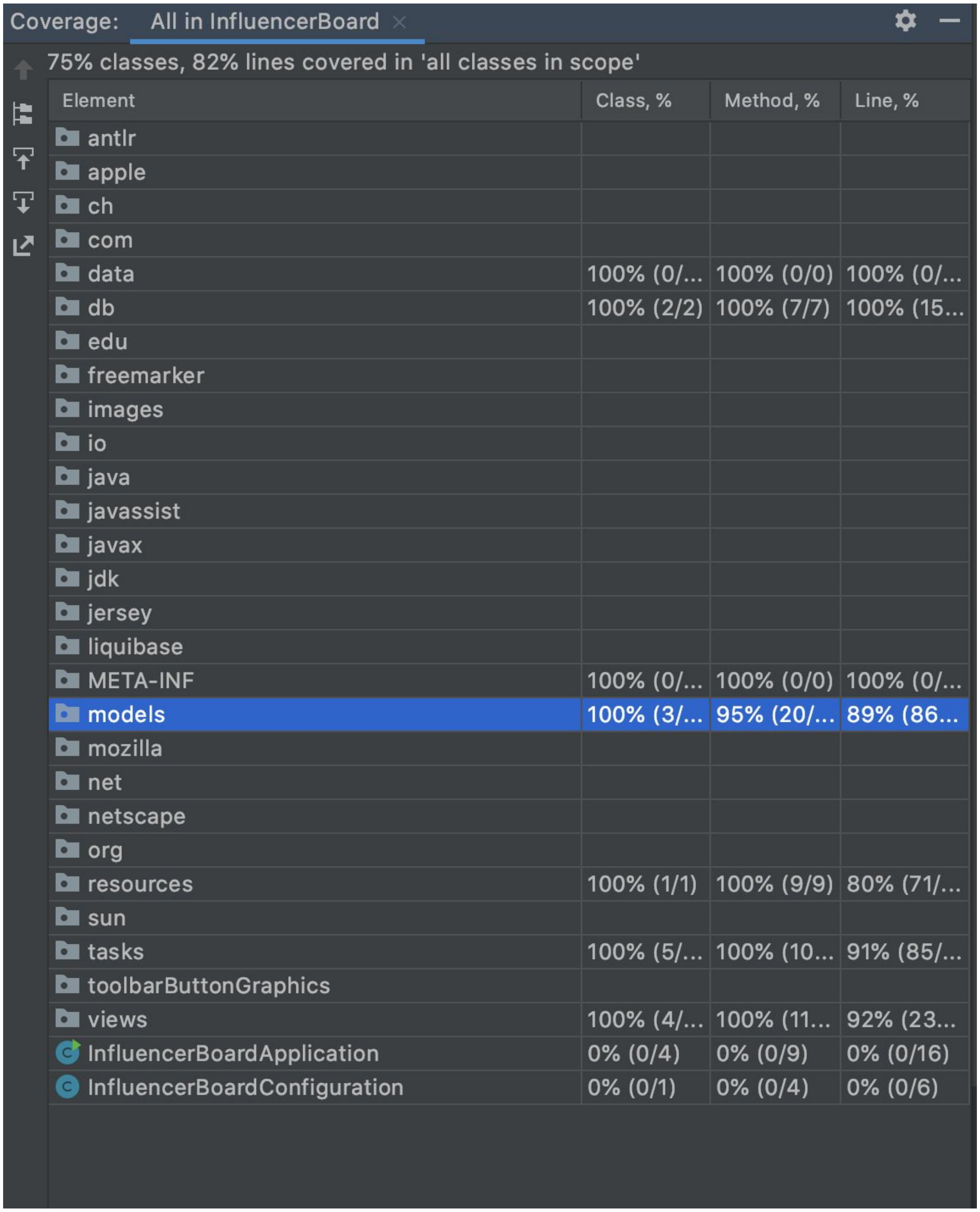
getPopularVideoList

This method retrieves a list of most popular videos for a given channel ID.
Prototype: `public ArrayList<String> getPopularVideoList()`
Equivalence class A (invalid) → invalid channel id → `public void getPopularVideoListInvalidIdTest`
Equivalence class B (valid) → valid channel id →`public void getPopularVideoListValidIdTest`

Boundary Analysis:
We have boundary analysis for method getChannelAnalytics, which gets channel details from a file. The input is the number of channels we want to get. We have three test cases which are negative number(-1), 0 and positive number(6). They cover the boundary, valid and invalid class.

The rest of the methods do not have any boundary because the arguments for the rest of the methods are non-numerical. Those arguments can basically only be separated into valid and invalid classes. For example, for method getHomeForUser which takes two strings(email, userName) as arguments, we only have valid vs. invalid equivalence class, which corresponds to valid email and invalid email. Similarly, for all the database operations methods such as addLikeRecord, deleteLikeRecord, there are only valid vs. invalid cases, which corresponds to inserting data with null parameter(invalid), inserting legal data(valid), inserting legal but duplicate data(valid but would be rejected).

Part 3: Branch Coverage



75% classes, 82% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %
antlr			
apple			
ch			
com			
data	100% (0/...	100% (0/0)	100% (0/...
db	100% (2/2)	100% (7/7)	100% (15...
edu			
freemarker			
images			
io			
java			
javassist			
javax			
jdk			
jersey			
liquibase			
META-INF	100% (0/...	100% (0/0)	100% (0/...
models	100% (3/...	95% (20/...	89% (86...
mozilla			
net			
netscape			
org			
resources	100% (1/1)	100% (9/9)	80% (71/...
sun			
tasks	100% (5/...	100% (10...	91% (85/...
toolbarButtonGraphics			
views	100% (4/...	100% (11...	92% (23...
InfluencerBoardApplication	0% (0/4)	0% (0/9)	0% (0/16)
InfluencerBoardConfiguration	0% (0/1)	0% (0/4)	0% (0/6)

We archive 82% percent coverage. We are not able to test some of the codes because those are configuration codes as we can see at the bottom of the image. Those configuration classes would only be called when our real database is used. During the test, we mock a database and test our functions and thus those configuration classes and functions are not called.

Part 4: Continuous Integration

We use .travis.yml for CI configuration:
https://github.com/LucassLin/CS4156TeamProject/blob/main/.travis.yml

CI reports:
https://travis-ci.com/github/LucassLin/CS4156TeamProject/jobs/455162394