Second Iteration

https://github.com/LooseScruz/ada

I. User Stories

User Story 1: As someone who changes location often, I would like to be able to send messages to people even when we are not in the same room, so that I can always have the option of communication.

Title: Sending Messages Remotely
Actors: Users a, b, c...
Short Description: A user in the client can type any message into the chat box and wait for a response. They can type and wait in any order that they choose.

Flow:

1. User runs AdaClient (AdaServer is already running remotely)
    a. User “logs in” by entering their username
2. The client connects to the given port and a new NetworkSocketClient is created.
3. Account information is obtained (see User Story 2)
4. If a message is received, the client will check if it is present.
5. If the message is present, the system will print the sender and message (to all other clients connected to the server).
6. If the response is populated, the message will be played aloud (we may assume text-to-speech credentials are properly set up).
7. The client will insert the message into the database.
8. If the user types a message in response and presses enter, the message will be dispatched to the server.
9. If the server accepts it, it will be sent to the other clients and appear in their feed.
10. If the message is "exit," the client will close.

Alternative Flows:

1. If the text to speech client fails to be created, it is populated with null.
2. If the client cannot connect to a port, the client fails with an IllegalArgumentException.
3. If a null message is received, nothing will happen (Flow: step 8)

4. If the server fails, the message will not be received by any of the other clients.
5. If the postgres database is not running, the server will fail to start and no messages will be sent.

User Story 2: As someone who logs on and off frequently, I would like to have a persistent account, so I can log on and off.

Title: User Accounts
Actors: User
Description: A user is associated with an account, identified by a username of their choosing.

Flow: User runs AdaClient (see above).

1. The user is prompted to answer y/n if they have an account.
2. If they enter y they are prompted to enter a username.
3. The system checks the user against the database.
4. If the user is in the database the system prints "username validated."

Alternate Flows:

1. If the user answers "n" (to if they have an account), they are prompted to enter a new username.
2. If the username is not already in the system, the system prints "user created in database!" and they are brought to the text prompt (Flow: step 7).
3. If the username already exists or fails to be created, the system prints please try again and the user is prompted again. (Alternative Flows: step 4).
4. If the user typed "y" and the username is invalid the the system prints "username not in the system, try again" and they are brought back to the username prompt (Flow: step 4)
5. If the user enters something other than "y" or "n", the system prints "incorrect selection, please try again!" and they are brought back to the username prompt (Flow: step 4)

User Story 5: As someone who is vision-impaired, I want to be able to have text messages read aloud to me, so that I don't miss anything.

Title: Text to Speech
Actors: Users
Description: A user can hear received messages read aloud.

Flow:

1. User runs AdaClient (see above).
2. If the client can connect to the Google API, the textToSpeech client is created.
3. When a user receives a non-null message, if the text to speech client is non-null it will make a request to the server.
4. If the call succeeds, the message will be played to the user.

Alternative Flows:

1. If the text to speech client is null, a call will not be attempted, but the Client will print and store the message as usual.
2. If the message response is not populated, a text to speech call will not be made.
3. If the text to speech call fails, the audio will not be played.

II. Test Plan

test suite resides at `/ada_group/src/test/`

A. major subroutines

1. Remote-server sending messages

The remote sending of messages occurs between the server and several clients. The application acts like a traditional chat client.

a. equivalence partitions
   a. The major input here comes from the user's choice of string. In this way, we have the typical boundaries of no string, a large string, and a few middle-sized strings in between. A valid equivalence class would be a string of

length 1, length 50, and length 100. An invalid equivalence class would be a string of size 0, a null String, or a string beyond the database's ability to store data. These are tested in our large-string test, which tries a message of ~3,000 characters.

b. Additionally, the characters of a string may be valid or invalid (i.e., images are not valid while an emoji is).

b. boundary conditions

a. The boundary conditions here would be the limit between 0 and 1 string-length and the upper limit on the database's maximum input size.

2. Text-to-speech

The AdaTextToSpeech client calls synthesizeSpeech on the passed in String.

a. equivalence partitions

a. There are two equivalence partitions with regards to input-- valid strings and null (which is invalid).

b. GetAudio parses the synthesizeSpeechResponse and returns an AudioInputStream. A valid equivalence class is a valid response from the api. Invalid equivalence classes are an API failure, a null response, or an invalid response.

AudioUtil takes an AudioInputStream and plays it. If it catches an exception trying to play the audio it prints out the exception and does not play the audio.

a. equivalence partitions

i. There are two equivalence cases-- a valid audio stream (valid), and anything else (such as null, a corrupt AudioInputStream, or an AudioInputStream that has already been read).

3. Database

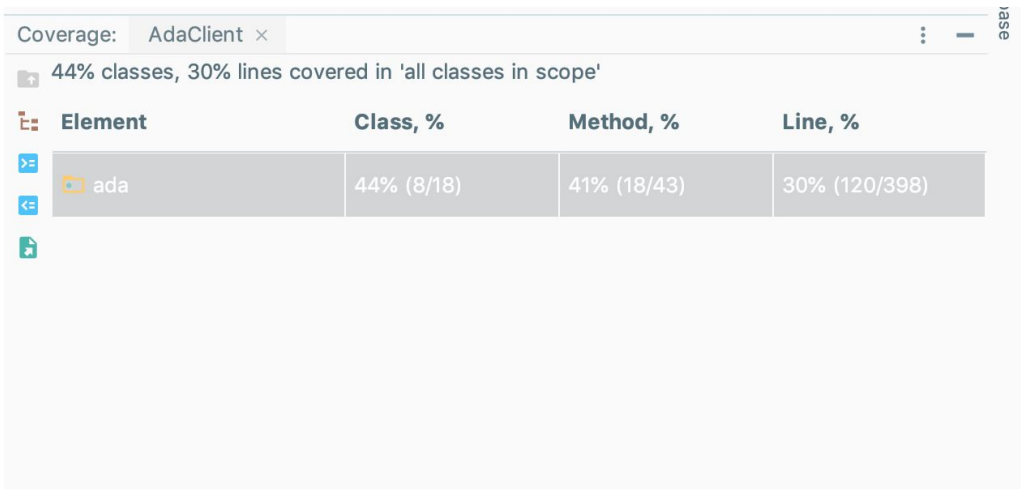
The database relies on postgres backend, and generally involves creation (accompanying starting the server) and insertion (accompanying the sending of messages). Because the creation commands are set (i.e., do not change with user input), only insertion is analyzed.

b. Equivalence partitions
    a. The input for the INSERT command is the user's selection of message, sender, and receiver. Focusing on the message, we see the same equivalence and boundary conditions noted above for strings. In addition to this, we have valid input as all keyboard-typeable characters and invalid input as non-keyboard characters. We also have a valid equivalence class as sending a message to an agent who is recorded in the database and an invalid class as sending a message to someone who is not. This is not specifically tested because all users who use the system must create an account, and messages are by default only sent to those users who are actively connected to the server. Invalid data would also include not selecting a sender or receiver or message--though, again, this is prevented in the code because otherwise no message would send.
    b. Another equivalence tests the existence of a valid postgres connection, which is assessed in the DB_Test file. We also tested the validity of usernames (must be unique) and the logic behind the account creation.[1] These tests look at valid user names, illegal characters like ‘, double insertions of a username, and an attempted (failed) sqlinjection attack.
c. Boundary conditions
    a. The boundary conditions here involve the use of typical string lengths. There is also the boundary between the existence of a postgres server, and there is the uniqueness of usernames (not unique, is unique).

[1] Due to an error with TravisCI, the tests in DB_Test were commented out because they would otherwise break the build.

III. Branch Coverage

A. Description: Jacoco from a POM plugin is used. This is part of our travisCI integration and is checked on pushes to GitHub.



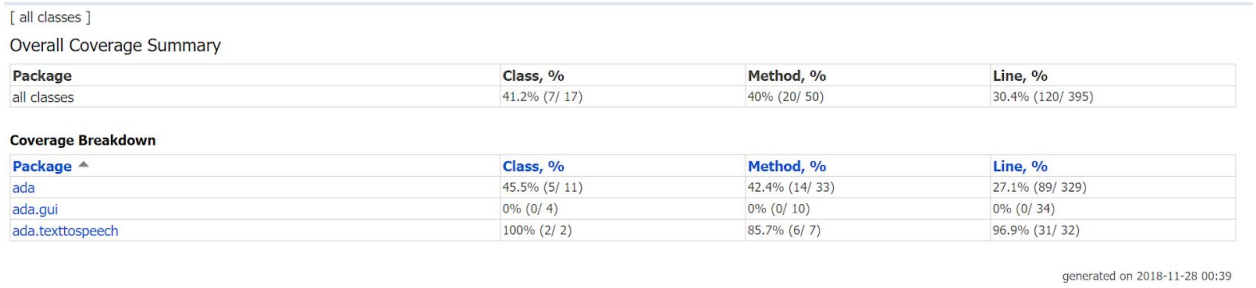
Coverage: jacoco.exec

33% classes, 26% lines covered in package 'ada'

Element	Class, %	Method, %	Line, %	Branch, %
gui	0% (0/4)	0% (0/8)	0% (0/38)	0% (0/4)
texttospeech	50% (1/2)	66% (4/6)	58% (18/31)	0% (0/2)
AdaClient	0% (0/1)	0% (0/2)	0% (0/61)	0% (0/20)
AdaServer	0% (0/1)	0% (0/2)	0% (0/12)	0% (0/2)
NetworkReader	100% (1/1)	75% (3/4)	63% (12/19)	66% (4/6)
NetworkSender	100% (1/1)	75% (3/4)	63% (14/22)	50% (5/10)
NetworkSocketClient	100% (1/1)	80% (4/5)	61% (24/39)	100% (6/6)
PostgreSQL_createUser	0% (0/1)	0% (0/2)	0% (0/54)	0% (0/12)
PostgreSQL_insertChat	0% (0/1)	0% (0/1)	0% (0/22)	100% (0/0)
PostgreSQL_queryHistory	0% (0/1)	0% (0/1)	0% (0/19)	0% (0/2)
PostgreSQLJDBC	0% (0/1)	0% (0/1)	0% (0/20)	100% (0/0)
TCPHost	66% (2/3)	80% (4/5)	62% (39/62)	79% (19/24)

Second Iteration

We had good coverage on the Text to Speech components and the core NetworkLogic (100%) , which were written a little earlier, and some coverage on the database logic, which was written in the late stages and has testing ongoing (travisCI had problems initially testing the postgres database). Jacoco is also slightly underreporting, as tests of the AudioUtil (which play a sound and are human-verified), are not run in Continuous Integration-- the local coverage can be viewed here:



[all classes]

Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	41.2% (7/ 17)	40% (20/ 50)	30.4% (120/ 395)

Coverage Breakdown

Package	Class, %	Method, %	Line, %
ada	45.5% (5/ 11)	42.4% (14/ 33)	27.1% (89/ 329)
ada.gui	0% (0/ 4)	0% (0/ 10)	0% (0/ 34)
ada.texttospeech	100% (2/ 2)	85.7% (6/ 7)	96.9% (31/ 32)

generated on 2018-11-28 00:39

Members: Mackenzie Glynn (mwg2126), Nathan Reitinger (nr2645), Riva Tropp (rtt2114), Brett Landau (bl2719)
Team: Church of CVN