HiLingual Server

Team 2

Garrett Davidson, Noah Maxey, Nate Ohlson, Riley Shaw

Problem Statement

Our team previously built an iOS app which relied on a backend written by a previous team member. This app is meant to facilitate learning new languages through talking to native speakers. This server has many limitations, including platform compatibility, speed, and scalability. We would like to scrap that server and start over using different technologies to address these issues. We will not be reusing any of the code because we are transitioning from a Dropwizard based Java server to a fully Swift server. We do not intend to use the app client for any part of this course, the client will help guide the high level design of the new server. Because we do not plan to use the client for any part of this project, we will only be submitting tests for the server. Creating this new server will also allow us to design from the ground up with testing in mind to prevent regressions as we add more features in the future.

Background

With our previous server implementation, we had the server hosted remotely, running Java, and relying heavily on the Dropwizard framework. The remote hosting, combined with the resource intensive nature of Java, made keeping the server up an expensive task. Using Java and Dropwizard also limited our flexibility when using our Swift-based iOS client. We would like to convert this server to use Swift, and add some more features to it, to run more efficiently, on our own machines, and be more compatible and extensible with our iOS client.

Environment

This server will be written entirely in Swift 3.0. It will be compiled and run on a Linux host. For storage, we will be a MySQL database for long-term and native Swift storage for short-term storage.

Requirements

Functional:

Backlog ID	**Functional Requirements**	**Hours**
1.	As a user, I would like to send and received photos and audio messages on the server.	**15**
2.	As a user, I would like to translates messages to my native language.	**16**
3.	As a user, I would like to store my sent and received Unicode compliant text messages on the server.	**20**
4.	As a user, I would like to send and receive my flashcards from the server.	**20**
5.	As a user, I would like to receive and send edits of my profile to the server.	**15**
6.	As a user, I would like to be matched with users similar to me.	**30**
7.	As a user, I would like to create a new profile.	**14**
8.	As a user, I would like to receive push notifications.	**20**
9.	As a user, I would like to authenticate using Facebook or Google account.	**10**
	Total:	**160**

Non-Functional:

1. We must have the server run on Linux machines.
2. Messaging must be as fast and reliable as possible.
3. Messages should be stored securely on the server.
4. Personal/account information must be kept secure.
5. Server must be able to support all target languages.
6. Server must be able to authenticate users via OAuth 2.0.
7. Server must be able to integrate with various support APIs (translation, etc).

8. Server must support a large number of concurrent users.
9. Server must be able to be run locally for testing/development.
10. Server must be very stable and unlikely to crash.

Use Cases

Case: Send a message (text, audio, image)

Action	System Response
1. Send text, audio or picture message	2. Validate message
	3. Store message in database
	4. Send successful response code

Case: Request a translation of a received text message

Action	System Response
1. Request a translation of a text message.	2. Send original message text to translation server
	3. Receive and translated text
	4. Store translated text in database
	5. Send translated text back to user
6. Receive translated message.	

Case: Retrieve messages (text, audio, image)

Action	System Response
1. Request message from server	2. Look up user id
	3. Read messages sent to that user id from database
	4. Respond with messages

Case: Upload a flashcard set

Action	**System Response**
1. Send flashcards to server	2. Parse flashcards
	3. Store flashcards in database
	4. Return successful response code

Case: Retrieve a flashcard set

Action	**System Response**
1. Request flashcards	2. Look up user id
	3. Retrieve flashcards from database
	4. Send flashcards to user

Case: Update profile information

Action	**System Response**
1. Request current profile information	2. Look up user id
	3. Retrieve user profile information from database
	3. Send profile to user
4. Change profile info and send differences to server	5. Look up user id
	6. Store changes to profile in database

Case: Request list of matches

Action	**System Response**
1. Request list of matches	2. Retrieve profile info for user id from database
	3. Run matching algorithm against user id profile information
	4. Compile list of matching users
	5. Respond with matching users

Case: Create new profile

Action	**System Response**
1. Create new profile	
2. Send new profile information to server	3. Create new user id
	4. Store profile information in database
	4. Return successful response code

Case: Generate push notification

Action	**System Response**
1. Send message to another user	2. Validate message
	3. Store message in database
	4. Generate push notification for recipient

Case: Authenticate using Facebook/Google

Action	**System Response**
1. Send server authentication id and authority token	2. Verify token with authority
	3. Create application session token
	4. Send application session token to user