Project 4

- **Assigned: Friday October 28**
- **Task 1 Due: Sunday November 6, 11:59pm**
- **Task 2 Due: Friday November 18, 11:59pm**

Assigned by Joe Mertz
Please direct questions to Piazza, to a TA, or to Joe.

Working in Pairs option

For this project, you can work in teams of two; if you prefer, you can work alone. If you work as a pair, you can fully collaborate and turn in only one solution for Task 1 and 2. You cannot discuss your project with others beyond your teammate. If working as a team, you should only turn in the project to Canvas once under the ID of the teammate whose Andrew ID comes first sorted alphabetically. You should not turn it in twice.

The team can use grace days for Task 2 (only). Both students will be charged a grace day for each late day. If both students have not used all their grace days, then neither will be penalized if they turn in the assignment late. However, if a student has already used the allowance, they will be penalized 10% per day late, beyond their available grace days.

If you decide to work as a pair, you must do so for Task 1 and Task 2. There will be a short peer-review on Canvas that each team member must complete.

Project Topics: Mobile to Cloud application

This project has 2 tasks: - Task 1 involves researching, selecting, and demonstrating that you can successfully use the technologies you plan to use in your project. - Task 2 will build on *Lab 3 - Creating Containers and Deploying to the Cloud* and *Lab 8 - Android Lab*. You will design and build a distributed application consisting of a mobile application, a web service that communicates with a RESTful web service in the cloud, and a dashboard that displays logging and simple analytics about your application.

Task 1: Demonstrate Project Feasibility

The goal of this task is to develop an idea for your application, and demonstrate the feasibility of using a 3rd party API and a database-as-a-service.

Start by researching and choosing a 3rd party API to use in Task 2. Be sure to read Task 2 carefully to see how you will use the API, and to understand what APIs cannot be used.

To complete Task 1, you should satisfy the following requirements:

1. Fetch data from a 3rd party API
a. Create a simple Java application that will make a request to the API and receive structured JSON or XML data.
b. Extract some piece of data and print it to the console.

2. Write and read data to MongoDB Atlas
a. Read the Database section below about creating and using mongoDB with Atlas.
b. Create a mongoDB database on Atlas.
c. Create (another) simple Java application that will:
- Prompt the user for a string. - Write the string as part of a document to the mongoDB database.
- Read all documents currently stored in the database.
- Print all strings contained in these documents to the console.

3. Create a document that contains:
a. Your name and Andrew ID
b. The name of the API (e.g. Flickr)
c. The URL of the API documentation (e.g. https://www.flickr.com/services/api/).
d. A short description (1-3 sentences) of what your mobile application will do with the information from the API (e.g. My mobile app will prompt the user for a string and then search Flickr using their API to display an interesting picture tagged with that string.)
e. A screenshot of the console output from section 1b.
f. A screenshot of the console output from section 2c.

4. Submit your document (only) to Canvas as a pdf by the Task 1 deadline.
Do not submit your code. It will be integrated into Task 2.

Grading:
- Task 1 complete and submitted on time: Bonus 5 points - Task 2 does not use the API demonstrated in Task 1: Penalty 10 points

(Small print to head off lots of questions on Piazza: If you don't submit Task 1 on time, you get no bonus points. If you have not committed to an API by submitting Task 1 by the deadline, you can't be penalized for not using that API. If you do not use the API submitted in Task 1 on time, you still get the 5 bonus points, but are penalized 10 points, resulting in a net penalty of 5 points. Grace days cannot be used for Task 1. Grace days, if you have some remaining, can be used for Tasks 2. No purchase necessary. Limit one winner per household. Offer void in NJ, TX, and Jaynestown).

Task 2: Distributed Application and Dashboard

The Distributed Application

Your application must be of your own creative design. (We will use software similarity detection software to identify those who do not.) It can be simple, but should fetch information from a 3rd party source and do something of at least marginal value. For example, we have assigned projects that generate hash values, implement clickers, or manage a blockchain. Your application should do something similarly simple but useful (but you should not reuse our ideas or the ideas of your peers!).

The following is a diagram of the components for your this part of your application:

Your web service should be deployed to Heroku and provide a simple RESTful API similar to those you have developed in prior projects. You do NOT have to implement all HTTP methods, only those that make sense for your application. Your web service must fetch information from some 3rd party API. In Project 1 you experimented with screen scraping, therefore that is not allowed in this project. Rather, you must find an API that provides data via XML or JSON. It is easy and can be fun to search for APIs; ProgrammableWeb and the GitHub Public APIs repository are good resources.

Use APIs that require authentication with caution. Many APIs will require you get a key (e.g. Flickr, which you used in the Android lab, required an API key). This is ok. But APIs that require authentication via OAuth or other schemes add a lot of work. Experiment ahead of time, but if you are brave, go ahead…

Be sure your API is from a reputable source. Your API still needs to be available when the TAs go to grade your project. Make sure you do not base your project on an API built by a 7th grade student…

Banned APIs: There are a number of APIs that have been used too often and are no longer interesting in this class. Therefore, you **cannot** use any of the following: - Agify.io - Alpha Vantage - Dog.ceo - Eventful - Flickr (for we have already done that) - Google Maps (unless you also use a 2nd API to get info to put on the map) - Last.fm - Merriam-Webster Dictionary - NASA Astronomy Picture of the Day - NYTimes APIs: specifically top stories, news wires, popular, and books. - OpenMovieDatabase - Pokemon API - Spoonacular - Spotify - Any weather API - Yahoo Finance API - Yelp - YouTube - Zomato

Users will access your application via a native Android application. **You do not need to have a browser-based interface for your application** (only for the Dashboard). The Android application should communicate with your web service deployed to Heroku. Your web service is where the business logic for your application should be implemented (including fetching information from the 3rd party API).

In detail, your distributed application should satisfy the following requirements:

Distributed Application Requirements

1. Implement a native Android application

a. Has at least three different kinds of Views in your Layout (TextView, EditText, ImageView, or anything that extends android.view.View). **In order to figure out if something is a View, find its API. If it extends android.view.View then it is a View.**
b. Requires input from the user
c. Makes an HTTP request (using an appropriate HTTP method) to your web service
d. Receives and parses an XML or JSON formatted reply from your web service
e. Displays new information to the user
f. Is repeatable (I.e. the user can repeatedly reuse the application without restarting it.)

2. Implement a web service, deployed to Heroku

a. Implement a simple (can be a single path) API.

b. Receives an HTTP request from the native Android application
c. Executes business logic appropriate to your application. This includes fetching XML or JSON information from some 3rd party API and processing the response. - -10 if you use a banned API - -10 if screen scrape instead of fetching XML or JSON via a published API

d. Replies to the Android application with an XML or JSON formatted response. The schema of the response can be of your own design.
- -5 if more information is returned to the Android app that is needed, forcing the mobile app to do more computing than is necessary. The web service should select and pass on only the information that the mobile app needs.

Use Servlets, not JAX-RS, for your web services. Students have had issues deploying web applications built with JAX-RS to Docker Containers and a solution has not yet been found.

Refer back to Lab 3 for instructions on pushing a web service to Heroku.

3. Handle error conditions

Your application should test for and handle gracefully: - Invalid mobile app input - Invalid server-side input (regardless of mobile app input validation) - Mobile app network failure, unable to reach server - Third-party API unavailable - Third-party API invalid data

Web Service Logging and Analysis Dashboard

Now enhance your web service to add logging, analysis, and reporting capabilities. In other words, create a web-based dashboard to your web service that will display information about how your service is being used. This will be web-page interface designed for laptop or desktop browser, not for mobile. In order to display logging and analytical data, you will have to first store it somewhere. For this task, you are required to store your data in a noSQL database, or more specifically a MongoDB, database hosted in the cloud.

The following is a diagram showing the dashboard components of your distributed application:

Logging data

Your web service should keep track (i.e. log) data regarding its use. You can decide what information would be useful to track for your web application, but you should track at least 6 pieces of information that would be useful for including in a dashboard for your application. It should include information about the request from the mobile phone, information about the request and reply to the 3rd party API, and information about the reply to the mobile phone. Information can include such parameters as what kind of model of phone has made the request, parameters included in the request specific to your application, timestamps for when requests are received, requests sent to the 3rd party API, and the data sent in the reply back to the phone. Be creative about what is useful for your application.

You should NOT log data from interaction with the operations dashboard, only from the mobile phone.

Storing logs

You should store your log data persistently so that it is available across restarts of the application. For this task you should use MongoDB to store your log data. MongoDB is a noSQL database that is easy to use. By incorporating it into your web service you will gain experience using a noSQL database, and experience doing CRUD operations programmatically from a Java program to a database.

See the MongoDB section below for more information on MongoDB, how to set it up, and hints on how to connect to it.

The Dashboard

The purpose of logging data to the database is to be able to create an operations dashboard for your web service. This dashboard should be web page interface for use from a desktop or laptop browser (not a mobile device).

The dashboard should display two types of data: 1. Operations analytics – display at least 3 interesting operations analytics from your web service. You should choose analytics that are relevant to your specific web service. Examples for InterestingPicture might be top 10 picture search terms, average Flickr search latency, or the top 5 Android phone models making requests. 2. Logs – display the data logs being stored for each mobile phone user interaction with your web service. The display of each log entry can be simply formatted and should be easily readable. **(Three points will be lost if they are displayed as JSON nor XML.)**

You will likely find HTML tables useful for formatting tabular information on a web page. And there are plenty of examples of embedding data in tables with JSP on the web. No frameworks are necessary for this, just < 20 lines of JSP (i.e. mixed HTML and Java). You may use a client-side framework if you like (e.g. Twitter Bootstrap).

In detail, your dashboard should satisfy the additional requirements:

Web Service Logging and Analysis Dashboard Requirements

4. Log useful information

At least 6 pieces of information is logged for each request/reply with the mobile phone. It should include information about the request from the mobile phone, information about the request and reply to the 3rd party API, and information about the reply to the mobile phone. (You should NOT log data from interactions from the operations dashboard.)

5. Store the log information in a database

The web service can connect, store, and retrieve information from a MongoDB database in the cloud.

6. Display operations analytics and full logs on a web-based dashboard

a. A unique URL addresses a web interface dashboard for the web service.
b. The dashboard displays at least 3 interesting operations analytics.
c. The dashboard displays **formatted** full logs.

7. Deploy the web service to Heroku

Writeup

Because each student's distributed application will be different, you are responsible for making it clear to the TAs how you have met these requirements, and it is in your best interest to do so. You will lose points if you don't make it clear how you have met the requirements. Therefore, you must create a document describing how you have met each of the requirements (1 – 7) above (you do **not** need to document *3. Handle error conditions.*) Your writeup will guide the TAs in grading your application. See the provided example (Project4Task1Writeup.pdf) for the content and style of this document.

Alternatively, instead of a document, you may submit a narrated screencast that includes the same information that would be in the writeup.

Demos

The TAs will identify exemplar projects that are unique in some way and nominate them to be demonstrated in class. Those who do so get a small prize!

Questions

If you have a question, please post them to the course Piazza, ask a TA, or direct to Joe.

Submission Summary

You will have one web service app, one Android app, and a WriteUp document to submit. Be sure each is named to be obvious what it is.

Be sure you meet the Documentation standards given in the first week of class, especially including an Author comment with your name and AndrewID. Failing to include your name and AndrewID in each source file will result in losing 5 points.

For each IntelliJ IDEA project, "File->Export To Zip File…". You must export in this way and NOT just zip the project folders.

Create a new empty folder **named with your Andrew id** (very important). Put your web service zip, your Android project zip, and your WriteUp pdf in that folder and zip it.

Now you should have only one .zip file named with your AndrewID.zip. Submit that single .zip file to Canvas.

MongoDB

The main MongoDB web site is https://www.mongodb.com. The site provides documentation, a downloadable version of the database manager application (*mongod*) that you can run on your laptop, and MongoDB drivers for many languages, including Java.

Mongod is the MongoDB database server. It listens by default on port 27017. Requests and responses with the database are made via a MongoDB protocol.

Mongo (without the DB) is a command line shell application for interacting with a MongoDB database. It is useful for doing simple operations on the database such as finding all the current contents or deleting them.

Because your web service will be running in the Heroku PaaS (or more specifically, Container-as-a-Service), you can't run your database on your laptop. Rather, you should use a MongoDB-as-a-Service to host your database in the cloud. Atlas (https://www.mongodb.com/atlas/database) is required because it has a free level of service that is adequate for your project.

Please read carefully… This project will challenge you to do a lot of research to understand enough MongoDB to create a simple database, add a collection, and insert, update, and find documents in that collection. This is very much like you will need to do regularly in industry. Code examples are provided on the MongoDB site, and elsewhere. **As long as you include comments as to their source, you can use them in your code.** If we search for a snippet of your code find it somewhere, and you have not attributed it to where you found it, that will be cheating and reason for receiving a failing grade in the course. Of course, the bulk of your code that is unique to your application should be your own and not copied from anywhere.

Setting up MongoDB Atlas

In this project, you are going to us nosql-database-as-a-service with MongoDB Atlas. Information about MongoDB can be found here: https://www.mongodb.com/what-is-mongodb

Getting started: 1. Create your account. Go to https://www.mongodb.com/atlas/database and create your own free account. 2. Answer the "Tell us a few things..." questions however you like, but include Java as the preferred language. 3. Choose to create a FREE shared cluster. 4. Accept the default settings and Create Cluster. 5. In the Security Quickstart: - *How would you like to authenticate your connection?*
Authenticate using Username and Password. Create a MongoDB user name and password (**only use letters and numbers to save yourself some hassle for encoding it later**) - don't forget these. The cluster takes a few minutes to provision, so be patient. - *Where would you like to connect from?*
Choose My Local Environment, add the IP address `0.0.0.0/0`, and Add Entry. What this IP address means is that your DB will be open to the world, which is required for the grading purposes. (You can check this later on the Security tab, IP Whitelist. If it doesn't have that IP address, click on Edit.) - Then Finish and Close. - If a popup window invites you to set up "Termination Protection", just "Close". (You don't need that for this project.) 6. Connect to the cluster.
a) Click on the Connect button.
b) Select the MongoDB Drivers option, then choose the Driver as Java, use version 4.3 or later.
c) Select Include full driver code example. Click Copy to copy that code stub. For now, save that code in a file; later, you'll edit and paste into your application to connect to your MongoDB instance, but don't forget to replace your with your database user's credentials (Note that when entering your password, any special characters are URL encoded; that's why a simple password is better here).
d) You will access this database in two ways:
For Task 1: Create a simple Java application to demonstrate reading and writing to the database as described in Task 1 above.
For Task 2: In your Web Service Logging and Analysis Dashboard

The sample code in the Quick Start guide shows how to access the database. You can access this cloud-based MongoDB database from your laptop as well as from Heroku.

Info about the MongoDB Java driver and sample code can be found here:
https://docs.mongodb.com/drivers/java/sync/v4.3/quick-start/

You can easily add the MongoDB Java Drivers to a project with Maven:

```
<dependency>
   <groupId>org.mongodb</groupId>
   <artifactId>mongodb-driver-sync</artifactId>
   <version>4.3.4</version>
</dependency>
```

Hints for connecting to MongoDB Atlas

Use a password that uses only letters and numbers so you don't have to deal with encoding it.

When Heroku communicates with MongoDB Atlas, it requires TLSv1.2. To enable this, edit your Dockerfile and add the following lines near the top of the file with the other ENV commands. (But not the first line in the file.)

```
 # Use TLSv1.2 for communication between Heroku and MongoDB
ENV JAVA_OPTS="-Djdk.tls.client.protocols=TLSv1.2"
```

The MongoDB connection string that Atlas provides is of the form:

```
mongodb+srv://USER:PASSWD@CLUSTER.mongodb.net/mydb?retryWrites=true&w=majority
```

But the `+srv` will not work with a number of DNS servers, and TLS and an authentication mechanism needs to be defined. Therefore: 1. On the MongoDB Atlas dashboard where you created the database, click on *Database* and then on *Cluster* link (it might be Cluster0, Cluster1, etc.) Under the REGION label you will see three shard servers listed. Click on the name of each to display the full server URL, and copy the full URL. It will look something like `cluster0-shard-00-02.cbkkm.mongodb.net:27017` 2. Find the URLs for the other two shard servers and copy them also. 3. Create your own connection string: `mongodb://USER:PASSWD@SERVER1,SERVER2,SERVER3/test?w=majority&retryWrites=true&tls=true&authMechanism=SCRAM-SHA-1`
Be sure to substitute your own values for USER, PASSWD, and SERVER1-3

The resulting connection string should look similar to:

```
mongodb://myuser:mysecretpassword@cluster0-shard-00-02.cbkkm.mongodb.net:27017,cluster0-shard-00-01.cb
```

When running your application, you will see the following warning:
`WARNING: SLF4J not found on the classpath. Logging is disabled for the 'org.mongodb.driver' component`

If you would like to see the messages logged from the MongoDB driver, add the following dependencies to the pom.xml file:

```
<!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-api -->
 <dependency>
         <groupId>org.slf4j</groupId>
         <artifactId>slf4j-api</artifactId>
         <version>1.7.36</version>
 </dependency>
 <!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-simple -->
 <dependency>
         <groupId>org.slf4j</groupId>
         <artifactId>slf4j-simple</artifactId>
         <version>1.7.36</version>
 </dependency>
```

Here is a link for more information about BSON.