

# 95-702 Distributed Systems

## Project 4

Assigned: Friday October 18

Task 0 Due: Saturday October 26, 11:59pm

Task 1&2 Due: Friday November 8, 11:59pm

### Project Topics: Mobile to Cloud application

This project has 3 tasks.

Task 0 involves researching, selecting, and demonstrating using a 3<sup>rd</sup> party API that you will use in Task 1&2, and writing and reading from a nosql-database-as-a-service.

Task 1 will build on the Deploying to the Cloud Lab and the Android Lab. You will design and build a simple mobile application that will communicate with a RESTful web service in the cloud.

Task 2 will add an operations logging and analysis function to your Task 1 web service.

When completing these tasks, the student should reflect on synchronous and asynchronous calls, event handling, remote interfaces, mobile and cloud computing.

### Task 0 Demonstrate using a 3<sup>rd</sup> Party API and MongoDB-as-a-Service

Research and choose a 3<sup>rd</sup> party API to use in Task 1 and 2. Be sure to read Task 1 carefully to see how you will use the API, and to understand what APIs cannot be used.

In detail, you should satisfy the following requirements:

#### **1. Fetch data from a 3<sup>rd</sup> party API**

- 1.1. Create a simple Java application that will make a request to the API and receive structured JSON or XML data.
- 1.2. Extract some piece of data and print it to the console.

#### **2. Write and read data to MongoDB Atlas**

- 2.1. Read the Database section in Task 2 about creating and using mongoDB with Atlas.

- 2.2. Create a mongoDB database on Atlas.
- 2.3. Create (another) simple Java application that will:
  - 2.3.1. Prompt the user for a string.
  - 2.3.2. Write the string as part of a *document* to the mongoDB database.
  - 2.3.3. Read all *documents* currently stored in the database.
  - 2.3.4. Print all strings contained in these documents to the console.
3. **Create a document that contains:**
  - 3.1. Your name and Andrew ID
  - 3.2. The name of the API (e.g. Flickr)
  - 3.3. The URL of the API documentation (e.g. <https://www.flickr.com/services/api/>)
  - 3.4. 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.)
  - 3.5. A screenshot of the console output from section 1.2.
  - 3.6. A screenshot of the console output from section 2.3.4
4. Submit your document (only) to Canvas as a pdf by the Task 0 deadline.  
**Do not submit your code. It will be integrated into Task 2 and 3.**

Grading:

- Task 0 complete and submitted on time: Bonus 5 points
- Task 1 does not use the API demonstrated in Task 0: Penalty 10 points

(Small print to head off lots of questions on Piazza: If you don't submit Task 0 on time, you get no bonus points. If you have not committed to an API by submitting Task 0 by the deadline, you can't be penalized for not using that API. If you do not use the API submitted in Task 0 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 0. Grace days, if you have some remaining, can be used for Tasks 1 and 2. No purchase necessary. Limit one winner per household. Offer void in NJ, TX, and Jaynestown.)

## Task 1 Mobile to Cloud Application

Design and build a distributed application that works between a mobile phone and the cloud. Specifically, develop a native Android application that communicates with a web service that you deployed to Heroku.

The application must be of your own creative design. It can be simple, but should fetch information from a 3<sup>rd</sup> party source and do something of at least marginal value. For example, we have assigned projects that generate hash values, implement

clickers, and securely transmit sensor information. Your application should do something similarly simple but useful (but you should not reuse our ideas or the ideas of your peers!).

Your web service deployed to Heroku should be a simple REST *ful* API as 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 3<sup>rd</sup> party API. In Project 1 we 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. One good place to look for such APIs is ProgrammableWeb. (<http://www.programmableweb.com/apis/directory>)

***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.*** Make sure you do not base your project on an API built by a 7<sup>th</sup> 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:

- Any weather API
- Flickr (for we have already done that)
- Spotify
- Google Maps (unless you also use a 2<sup>nd</sup> API to get info to put on the map)
- OpenMovieDatabase
- These NYTimes APIs: top stories, news wires, popular, and books.
- Yahoo Finance API

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 Task 2 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 3<sup>rd</sup> party API).

In detail, your application should satisfy the following requirements:

## **1. Implement a native Android application**

- 1.1. Has at least three different kinds of views in your Layout (TextView, EditText, ImageView, etc.)

- 1.2. Requires input from the user
- 1.3. Makes an HTTP request (using an appropriate HTTP method) to your web service
- 1.4. Receives and parses an XML or JSON formatted reply from your web service
- 1.5. Displays new information to the user
- 1.6. Is repeatable (i.e. the user can repeatedly reuse the application without restarting it.)

## **2. Implement a web service, deployed to Heroku**

- 2.1. Using an HttpServlet to implement a simple (can be a single path) API. (It is recommended that you do **not** try to use JAX-RS / Jersey.)
- 2.2. Receives an HTTP request from the native Android application
- 2.3. Executes business logic appropriate to your application. This includes fetching XML or JSON information from some 3<sup>rd</sup> 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
- 2.4. Replies to the Android application with an XML or JSON formatted response. The schema of the response can be of your own design. Alternatively, you can adopt a standard schema that is appropriate to your application. (E.g. Common Alerting Protocol if your application deals with emergency alerts.)
  - -5 if information beyond what is needed is passed on to the Android app, forcing the mobile app to do more computing than is necessary.

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

## **Writeup**

Because each student's mobile/cloud 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 (1.1 – 2.4) above. Your writeup will guide the TAs in grading your application. See the provided example (Project4Task1Writeup.pdf) for the content and style of this document.

## Task 2 Web Service Logging and Analysis Dashboard

For Task 2, you are to embellish your web service to add logging, analysis, and reporting capabilities. In other words, you are to 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.

**IMPORTANT NOTE:** Task 2 builds on Task 1, but for your own safety, you should not overwrite Task 1. Rather, once you have Task 1 working, you should create a separate Task 2 project. In this way you will never lose the working Task 1 that you are required to submit. When deploying to Heroku, you should deploy Task 1 and Task 2 separately. Heroku allows you do have multiple applications. In this way, if Task 2 does not work for some reason, we still have Task 1 to grade.

### 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 3<sup>rd</sup> 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 3<sup>rd</sup> party API, and the data sent in the reply back to the phone.

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

### Database

You should log your data persistently so that it is available across restarts of our application. For this task you should use MongoDB to store your logging 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.

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/cloud/atlas>) is recommended because it has a free level of service that is adequate for your project.

Here is some information about setting up MongoDB Atlas:

### **MongoDB Atlas**

In this project, you are going to use 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/cloud/atlas> and create your own account with a free "Starter Cluster"
- 2) When building a cluster, choose AWS as the cloud provider and choose N.Virginia (us-east-1) in the North American tab – you should see "Free tier available" there. Leave the Cluster Tier and Additional Settings as-is; you can give the cluster a name if you want (otherwise, it will be named something like "Cluster0"). Then click 'Create Cluster' in the bottom of the page; a Verify tab may pop up to make sure you're not a robot. The cluster takes a few minutes to create, so be patient.
- 3) Connect to the cluster.
  - a) Click on the 'Connect' button in the Sandbox section. Then click 'Add a Different IP Address'. Make the IP '0.0.0.0/0'. This means your DB will be open to the world, which is needed 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.
  - b) 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!
  - c) Click 'Choose a connection method'; choose 'Connect with your application'. Then choose the Driver as 'Java', use version 3.6 or later.
  - d) Click on the Full Driver Example (next to the Connection String Only tab). Click Copy to copy that code stub. For now, save that code in a file; later, you'll copy and paste that into your application to connect to your MongoDB instance, but don't forget to replace your <password> 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).

- e) Create your dashboard program that includes the code stub above; see the sample code in the Quick Start guide above to see how to access the database. Execute the code and you should see the log information from MongoDB. That means you have successfully connected to the Cluster. You may now create databases and collections needed for your dashboard.

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:

<http://mongodb.github.io/mongo-java-driver/3.11/driver/getting-started/quick-start/>

Bson info is available at:

<http://mongodb.github.io/mongo-java-driver/3.11/bson/documents/>

## 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:

- a) 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.
- b) 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. (They should not be 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).

## Please read carefully...

This task 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.

In detail, your solution should satisfy the following requirements:

### **1. 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 3<sup>rd</sup> party API, and information about the reply to the mobile phone. (You should NOT log data from interactions from the operations dashboard.)

### **2. Store the log information in a database**

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

### **3. Display operations analytics and full logs on a web-based dashboard**

- 3.1. A unique URL addresses a web interface dashboard for the web service.
- 3.2. The dashboard displays at least 3 interesting operations analytics.
- 3.3. The dashboard displays the full logs.

### **4. Deploy the web service to Heroku**

Deploy the web service to Heroku. This web service should have all the functionality of Task 1 but with the additional logging, database, and dashboard analytics functions.

In your Task 2 writeup be sure to include the dashboard URL!

### **Task 2 Writeup**

In the same style as Task 1, but in a separate document, describe how you have met these 4 requirements.

## **Prizes**

The TAs will identify exemplar projects that are unique in some way and nominate them to be demonstrated in class for small prizes!



## Questions

If you have questions, please post them to the course Piazza and the TAs and instructors will respond.

## Submission Summary

Follow the same submission procedures as in prior projects, except have writeup documents instead of screenshot directories.

You will have two server apps (Task 1, Task 2), and one Android app to submit. Depending on how you set up your work in IntelliJ IDEA, you may have one to three projects. (For example, you may have one project with 3 modules, or 3 projects, each with one module.) In any case, be sure the Modules include the strings “Task1”, “Task2”, and “Android” in their names.

You should also have two writeup documents that you should name:

Project4Task1Writeup.pdf

Project4Task2Writeup.pdf

You will have no separate screenshot directories for this project.

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

Now you should have one to three .zip files and two writeup documents

Create a new empty folder **named with your Andrew id (very important)**. Put all files mentioned above in to the new folder you created.

Zip that folder and submit it to Canvas. The submission should be a single zip file.

Now you should have only one .zip file named with your Andrew id.