

GRADEBADGE
DEVELOPMENT OF A CLOUD-BASED REWARD APPLICATION

A Project
Presented to the
Faculty of
California State University,
San Bernardino

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Erwin Toni Soekianto
April 2013

GRADEBADGE
DEVELOPMENT OF A CLOUD-BASED REWARD APPLICATION

A Project
Presented to the
Faculty of
California State University,
San Bernardino

by
Erwin Toni Soekianto

April 2013

Approved by:

David Turner, Chair, Computer Science and
Engineering

Date

Richard J. Botting

Arturo I. Concepcion

© 2013 Erwin Toni Soekianto

ABSTRACT

The purpose of this project is to investigate the use of cloud-based services to deliver cutting-edge applications. For this purpose, a prototype of a reward application using badges was developed to illustrate and explore this emerging paradigm. The cloud services utilized by this application include Heroku (for the application server), MongoLab (for the database), and Facebook (for authentication and social network integration). The application server is written in Javascript and runs inside a Nodejs execution environment. The application is accessed through a web browser running on either desktop or mobile computers.

On the client side, this application makes use of numerous web technologies, including HTML5, CSS, Bootstrap, and jQuery. The project also made use of the git version control system to manage source code and deployment of the application server to Heroku. The source code repository was stored remotely through the cloud-based service called GitHub.

The purpose of the GradeBadge application is to help organizations interact with and motivate their members in a fun way. It keeps the members engaged by giving badges as rewards for their efforts or achievements. In order to facilitate adoption among users, GradeBadge is integrated with the social networking site Facebook.

ACKNOWLEDGEMENTS

I would like to thank all the people with whom I have worked while pursuing my master's degree at California State University, San Bernardino (CSUSB). Studying in the School of Computer Science and Engineering at CSUSB has been a great learning experience. I would like to thank the faculty of the School of Computer Science and Engineering who supported this project by serving on my committee: Dr. David Turner, Dr. Arturo Concepcion and Dr. Richard Botting.

TABLE OF CONTENTS

<i>Abstract</i>	iii
<i>Acknowledgements</i>	iv
<i>List of Tables</i>	ix
<i>List of Figures</i>	x
<i>1. Introduction</i>	1
1.1 Background	1
1.2 Technology Overview	3
1.2.1 Facebook	3
1.2.2 Heroku	3
1.2.3 MongoDB	3
1.2.4 MongoLab	4
1.2.5 Git	4
1.2.6 Bootstrap	4
1.2.7 JQuery	4
1.2.8 Node.js	4
1.3 Project Purpose	5
1.4 Project Scope	5
1.5 Related Work	5
1.6 Definitions, Acronyms, and Abbreviations	6

2. <i>Software Requirements Specification</i>	9
2.1 External Interfaces Requirement	9
2.1.1 Hardware Interfaces	9
2.1.2 Software Interfaces	9
2.1.3 Communication Interfaces	9
2.2 Functional Requirement	10
2.2.1 Create group	10
2.2.2 Create Badge	11
2.2.3 Add Member	11
2.2.4 Issue Badges to Members	11
2.2.5 View Badge Earned	11
2.2.6 Share Badge to Social Networking	11
2.3 Performance Requirement	11
2.4 Design Constraint	11
2.5 Software System Attributes	12
3. <i>System Architecture</i>	13
3.1 Overview	13
3.2 Deployment Workflow	14
3.2.1 Developers/Release Manager	16
3.2.2 Heroku	16
3.2.3 MongoLab	17
3.2.4 Facebook	17
3.2.5 Client Browser	17
4. <i>System Design</i>	18
4.1 Design Overview	18
4.2 Model View Controller Architecture	18

4.3	Bandwidth Reduction Strategy	19
4.3.1	File Compression	19
4.3.2	Content Distribution Network Resources	20
4.3.3	Caching	20
4.4	Server-side Architecture Design	22
4.4.1	Configuration	23
4.4.2	Nodejs Modules	25
4.5	Mapping of Model Classes to MongoDB	33
4.6	Request Handler Operation	37
4.7	Client-side Architectre Design	41
5.	<i>Database Design</i>	43
5.1	Overview	43
5.1.1	MongoDB	43
5.1.2	MongoLab	43
5.1.3	Documents	43
5.1.4	Collections	44
5.2	Data Model	44
5.2.1	User Collection	45
5.2.2	Group Collection	45
5.2.3	Badge Collection	45
5.2.4	Group Admin Links Collection	46
5.2.5	Group Member Links Collection	46
5.2.6	Badge User Links Collection	47
5.2.7	Group Badge Links Collection	47
6.	<i>Project Implementation</i>	49
6.1	Loading Screen	50

6.2	Login Screen	51
6.3	Group Page	53
6.4	Add New Group Page	54
6.5	Group Page	55
7.	<i>Conclusion and Future Direction</i>	56
7.1	Conclusion	56
7.2	Future Direction	56
	<i>APPENDIX A: Server Side Source Code</i>	58
	<i>APPENDIX B: Model Classes Source Code</i>	64
	<i>APPENDIX C: Request Handler Operation Source Code</i>	67
	<i>APPENDIX D: Client Side Source Code</i>	70
	<i>References</i>	89

LIST OF TABLES

5.1	User Collection	45
5.2	Group Collection	45
5.3	Badge Collection	46
5.4	Group Admin Links Collection	46
5.5	Group Member Links Collection	47
5.6	Badge User Links Collection	47
5.7	Group Badge Links Collection	48

LIST OF FIGURES

2.1	Use Case Diagram	10
3.1	Deployment Diagram	14
3.2	System Integration	15
6.1	GradeBadge Loading Screen	50
6.2	GradeBadge Login Screen	51
6.3	Facebook Login Screen	52
6.4	GradeBadge Group Page	53
6.5	GradeBadge Add New Group Page	54
6.6	GradeBadge Group Page Added	55

1. INTRODUCTION

1.1 Background

A long time ago, businesses used to produce their own electric power. Due to an engineering breakthrough in electric generator and transmission methods, it became easier to produce and transmit electricity to businesses that once produced their own electricity. As more businesses started buying electric power, the production of electricity become less expensive, encouraging more compnies to purchase rather than make their own electricity.

And today, just like the utilities, instead of buying servers to run websites and applications, businesses rent servers and various services from cloud computing providers. These cloud resources are provided to companies in a way that resembles people renting apartments in a single building; even though you are in the same building with other people, you still have your own space. As more people rent and buy computing service from large-scale providers, the cost of these services are decreasing.

Today there are many cloud computing providers that provide different types of services. Some provide application hosting, databases, code repositories, authentication services, social network integration, etc. Some famous providers are Amazon, Google, Microsoft, Facebook, GitHub and Heroku.

Google App Engine provides infrastructure to build web applications on the same scalable systems that power Google applications. Google App Engine supports Python, Java and the Go programming languages. Google App Engine also provides several options for storing data, including App Engine Datastore, Google Cloud SQL, and

Google Cloud Storage. Windows Azure provides similar services as Google App Engine but supports a different set of programming languages, including ASP.NET, VB.NET, Java, Nodejs and Python.

Amazon also provides scalable cloud computing services, which includes the popular EC2 virtual machine instances, where users choose the type of operating system and configuration they need. The virtual machine service provided by Amazon is more flexible than language-specific execution environments such as Google App Engine and Heroku, but require more time and expertise to set up and manage.

In this project, Heroku is used as a cloud application platform for running Javascript in the Nodejs execution environment. Heroku also provide alternative execution environments that support Scala, Ruby, PHP and others. One of the benefits of using Heroku (and other similar application hosting services) is that the bandwidth and CPU capacity can be scaled up or down almost instantly to accomodate rapidly changing demand. Also, by relying on Heroku fto manage the hardware and system and network administration, developers gain the reliability, performance and security that is provided by a larger company with staff dedicated to these purposes.

Among all the programming language execution environments supported by Heroku, this project uses Nodejs, which supports server-side programming in Javascript. On the client side, the application uses HTML5, Javascript, JQuery library and the Bootstrap framework. The applciation also relies on the MongoDB database as provided by the MongoLab service provider.

Node.js and MongoDB are often used together to build scalable web applications. The following describe the cloud computing service providers used in this project.

1.2 *Technology Overview*

1.2.1 *Facebook*

Facebook is a popular social networking website more than one billion users. It has proven to be a good platform to use for web applications that take advantage of its social networking features, such as friend lists, wall posting, etc [8]. Facebook also allows other applications to access user data with their authorization using its API.

1.2.2 *Heroku*

Application hosting is a form of cloud computing that enables developers to publish applications that require Internet connectivity. Heroku is one of the first companies to offer a remote Nodejs execution environment. Nodejs applications are deployed to Heroku using git [12]. To deploy, or redeploy, an applicaiton, the developer pushes a branch of a git repository to a remote git repository provided by Heroku. At deployment, Heroku extracts the application files from the git repository and runs the main executable, which is a server process. Heroku work is a partner of Facebook and so is designed to work easily with it.

1.2.3 *MongoDB*

There are many different types of cloud-based datastore services to choose from. For this project we used MongoDB because it works well with Node.js and Heroku. MongoDB is a scalable, high-performance, open source, NoSQL document-based database. MongoDB features include document-oriented storage, indexes, replication, high availability, auto-sharding, and querying [21].

1.2.4 *MongoLab*

MongoLab is the cloud computing provider for MongoDB database that was used in this project [22]. MongoLab has a free tier service that facilitates experimentation by developers, and thus was convenient for this project.

1.2.5 *Git*

Git is a distributed version control system [9]. This project uses GitHub to store and manage a remote git repository of application source code. Git was convenient for the project because Heroku uses git as a means to deploy web applications to its servers. Heroku's git-based deployment system allows easy creation of testing, staging, and production versions of the application.

1.2.6 *Bootstrap*

Bootstrap is a freely available CSS and Javascript library created by Twitter. It provides a responsive design framework that works well for applications that run inside browsers in desktop computers, tablets and smart phones [4]. The application's user interface was constructed using Bootstrap.

1.2.7 *Jquery*

The GradeBadge application uses Jquery for AJAX transactions and DOM manipulation [19]. Bootstrap also depends on Jquery.

1.2.8 *Node.js*

Cloud-based services support apps written in several different programming languages, such as Java, Python, PHP, Javascript, Ruby and more. For this project we used Javascript running in a Node.js context. Node.js is a platform built on

Chrome's JavaScript runtime. It's purpose is to allow construction of fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient when used for I/O intensive applications such as Web applications [23].

1.3 Project Purpose

The purpose of this project is to explore the current technologies that enable rapid development and deployment of desktop and mobile Web applications that can scale to accomodate any number of users.

1.4 Project Scope

Project does not include database sharding features of MongoDB, which would allow a greater degree of scalability.

The GradeBadge application provides the following functionalities:

- Create Group
- Create Badge
- Add Member
- Issue Badges to Members
- View Badges Earned
- Share Badges with Social Networking Contacts

1.5 Related Work

There are other similar mobile Web application and cloud computing demonstration projects that were built as master's degree projects at CSUSB. One of these projects

was completed by Manoj Kulkarni [32]. This project used Google App Engine for an application hosting provider and Google App Engine datastore and Java programming language. It also used JQuery Mobile as UI framework. The GradeBadge project uses different set of technologies, and is the first project at CSUSB that utilizes Nodejs and MongoDB. Most other similar projects at CSUSB have relied on either Java, .NET or PHP.

1.6 Definitions, Acronyms, and Abbreviations

The definitions, acronyms, and abbreviations used in the document are described in this section.

- GradeBadge: The name of this project.
- API: Application Programming Interface, which is a set of routines that an application uses to request and carry out low-level services performed by a computer's operating system; also, a set of calling conventions in programming that defines how a service is invoked through the application [10].
- Cloud computing: the use of computing resources (hardware and software) that are delivered as a service over a network (typically the Internet) [?].
- JQuery: A javascript library for building web based applications [19].
- DOM: Document Object Model, which is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents [5].
- item I/O: Input/Output.
- UI: User Interface.
- CSUSB: California State University, San Bernardino.

- HTML: HyperText Markup Language, which is the authoring language used to create documents on the Web [29].
- HTTPS: Hyper-text Transfer Protocol Secure, which is a secure network protocol used to encrypt data transferred between server and client [14].
- MVC: Model-View-Controller is an architectural pattern used in software engineering to isolate business logic from user interface considerations [33].
- UML: The Unified Modeling Language, which is the industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems [28].
- Microsoft Azure: Cloud computing platform provided by Microsoft [31].
- Google App Engine: Cloud computing platform provided by Google.
- Amazon Web Services: Cloud computing platform provided by Amazon [2].
- Heroku: Cloud application platform provided by Heroku [12].
- Android: Mobile operating system provided by Google [3].
- IOS: Mobile operating system provided by Apple [16].
- NoSQL: Types of databases that use key-value pairs for storing data unlike traditional relational databases. [24].
- JSON: Javascript Object Notation, which is a data representation format using key-value pairs [20].
- Ajax: Asynchronous JavaScript and XML, which is a method for Web applications to communicate between client browsers and servers [1].
- OOP: Object Oriented Programming, which is a computer programming paradigm where application logic is organized into objects that provide an interface to encapsulated state. [30].

- CDN: Content Distribution Network.
- Slug: The binary deployed to Heroku.

2. SOFTWARE REQUIREMENTS SPECIFICATION

2.1 External Interfaces Requirement

2.1.1 Hardware Interfaces

The application is hosted in the Heroku application cloud service. The Web server communicates over HTTPS to ensure that data transferred between client and server is untampered and private. The system is a Web based application; users are required to use a high-speed Internet connection and use an up-to-date Web browser.

2.1.2 Software Interfaces

JavaScript will be implemented throughout the website in order to display the correct feature the user requested. And HTML5 may be implemented throughout the website in order to display the correct feature the user requested.

2.1.3 Communication Interfaces

This application is designed to be viewed on any Internet Web browser provided that JavaScript and image features are enabled and the browser is HTML5 compatible. Performance may vary slightly between browsers; however, the functionality of the site should not be impaired.

2.2 Functional Requirement

The functions specified in this section directly correspond to work that will be conducted in this project as shown in the use case diagram in Figure 2.1.

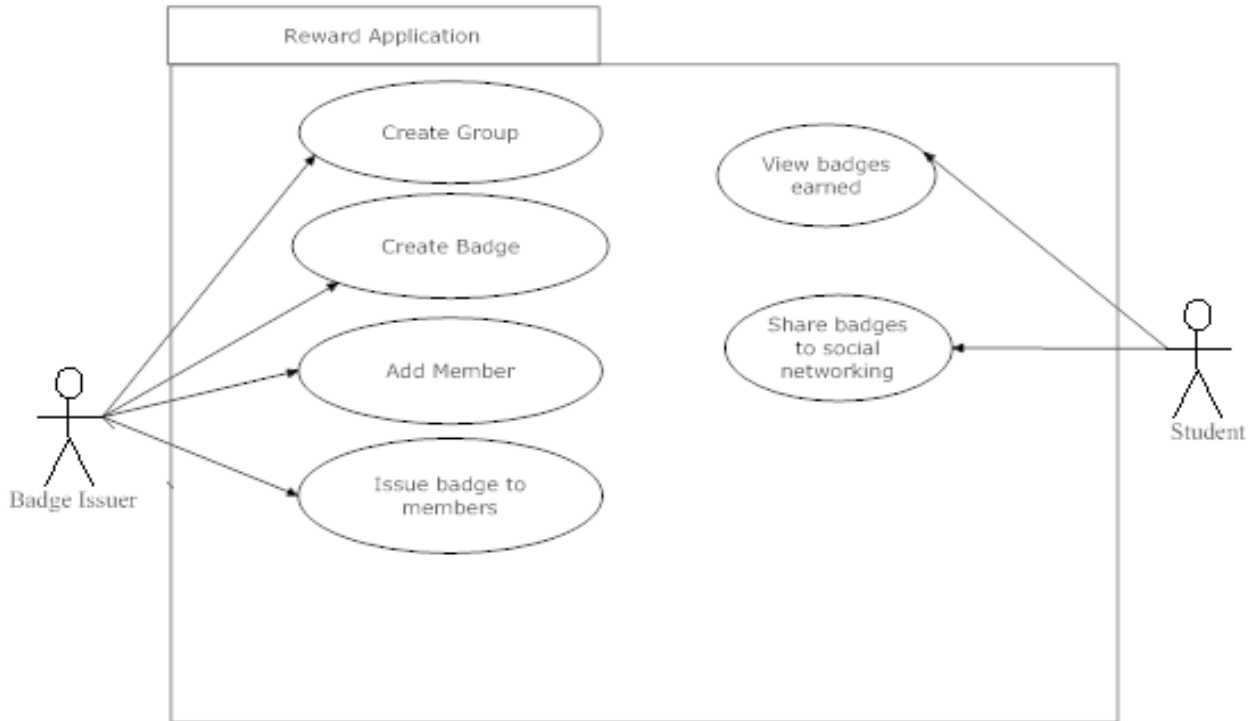


Fig. 2.1: Use Case Diagram

2.2.1 Create group

This functionality allows organizers or badge issuers to create groups, which will have a set badge collection.

2.2.2 Create Badge

This functionality allows badge issuers to create badge to be added to badge collection. A badge would have at least badge name and description.

2.2.3 Add Member

This functionality would allow organizers or badge issuers to add members into the groups as badge recipients. Member would have at least email address and name.

2.2.4 Issue Badges to Members

This functionality would allow badge issuers to issue badges to members.

2.2.5 View Badge Earned

This functionality would allow badge recipients to view all the badges that they have earned.

2.2.6 Share Badge to Social Networking

This functionality would allow badge recipients to share the badge to their social networking site such as Facebook

2.3 Performance Requirement

This application is going to be hosted in the Heroku cloud server, so the performance of this application would be high.

2.4 Design Constraint

This application requires internet-enabled devices and internet connection to perform. And every user must have Facebook account to be able to use this.

2.5 *Software System Attributes*

The author will keep coding standard with proper commenting and documentation.

3. SYSTEM ARCHITECTURE

3.1 *Overview*

There are three main server entities in this application, which are Heroku, Facebook, and MongoLab. When the client browser, either from desktop computer or tablet or smart phone, runs the application, it will hit the Heroku server, then the Heroku server will connect to Facebook server to check user authentication. And if the user is not logged-in, the user will be prompted with the Facebook login screen and submit the login data to Facebook server.

After the user is authenticated to use the application, the application server will connect to MongoLab server to read and write the data. The application server may occasionally go to Facebook server to check the users' friend list or post to user's wall or check whether the user is still logged-in.

This application uses HTTPS exclusively for security reasons, except in the local developer environment, where we use unencrypted HTTP, as shown in Figure 3.1.

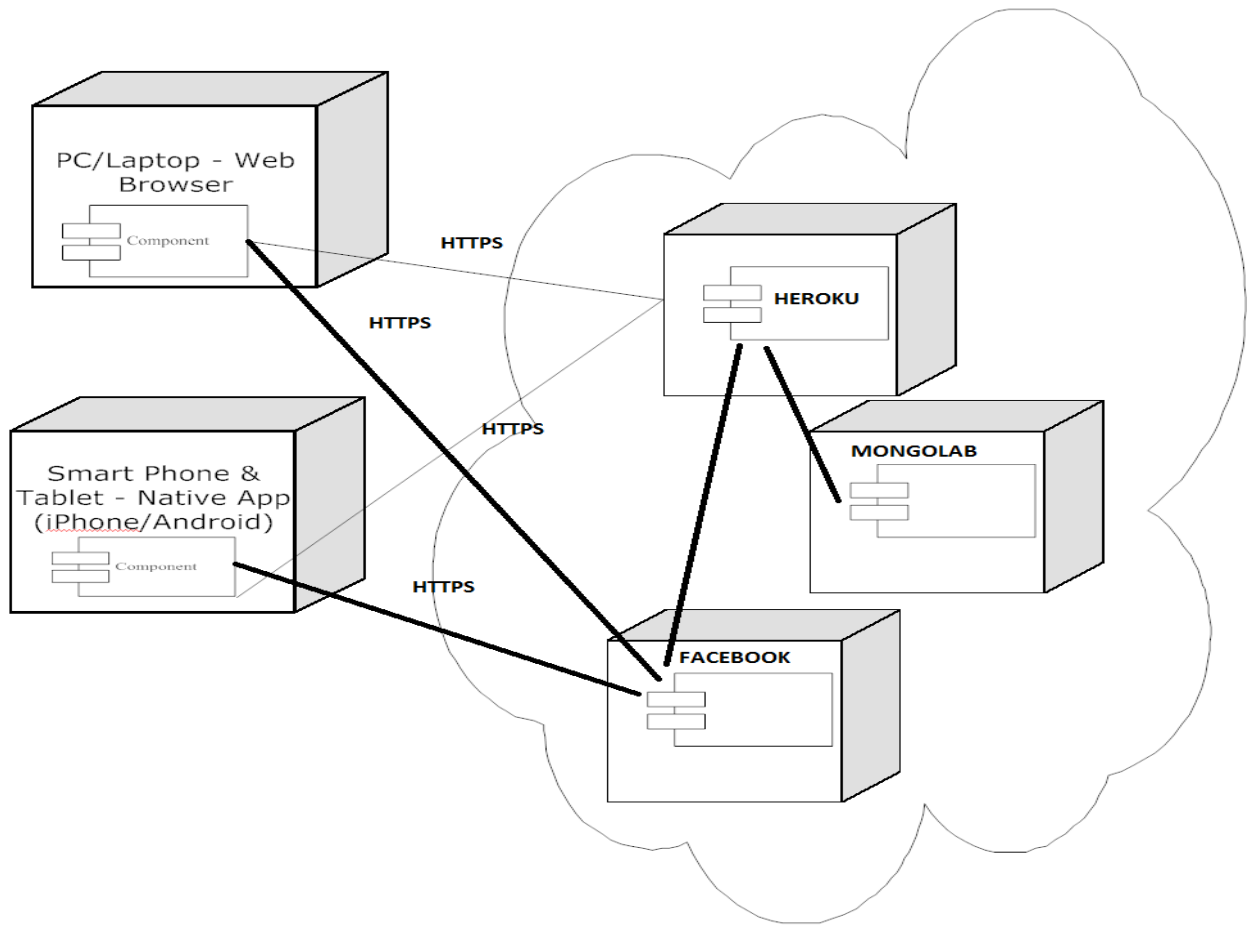


Fig. 3.1: Deployment Diagram

3.2 Deployment Workflow

There are three type of environments used in the deployment workflow: development, staging and production.

Developers work on new features or bug fixes in development branches then only minor updates are committed directly to the stable development branch. Once the features are implemented and/or set of bugs are fixed, they are merged in to staging branch and deployed to staging environment for testing and quality assurance. After testing is completed, the snapshot of staging branch is kept for production deploy-

ment, otherwise the process will repeat until the testing is completed. On the release date, the working staging branch is deployed to production environment.

Figure 3.2 illustrates the deployment workflow.

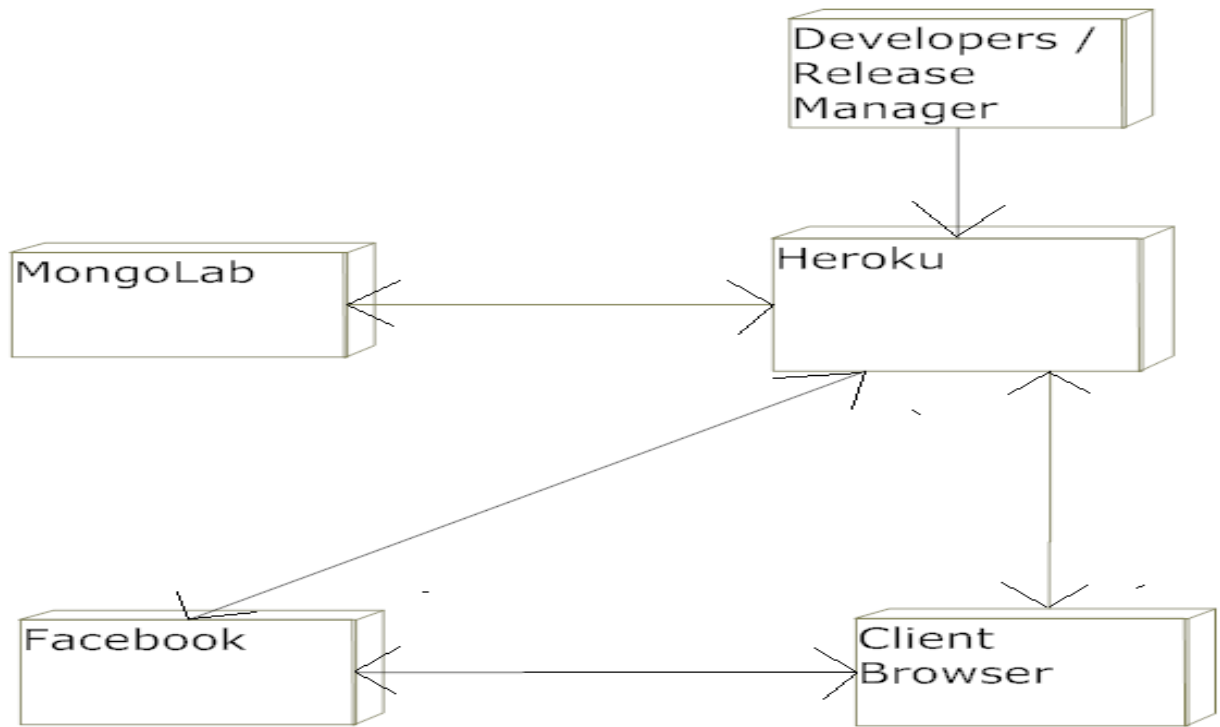


Fig. 3.2: System Integration

On this project, git is used as code repositories, to manage developments, staging and production branch. And Heroku toolbelt is also used to set the enviroment config variable for each deployment. Heroku allows users to use git to deploy automatically from local repositories.

3.2.1 *Developers/Release Manager*

In this project, developers first do unit testing in their local machines, then after the system reaches a certain point, the developer himself or assigned release manager will use git to push the changes to staging or production repositories that are connected to respective Heroku servers.

Below are the commands the developer/release manager uses to start the application in the staging environment.

```
$ foreman start
18:43:16 web.1 : started with pid 5540
18:43:16 web.1 : listening on 5000
```

Below are the commands for developer/release manager commits changes to master branch of the local repository, then followed by a push to master branch of the remote repository.

```
$ git status
$ git add .
$ git commit -m "message here"
$ git push origin master
```

Below are the commands for developer/release manager uses to push to staging environment.

```
$ git push staging master
```

3.2.2 *Heroku*

In this project there are two sets of heroku instances used: staging and production. The application running in Heroku connects to a database server running inside MongoDB using the MongoDB protocol to read and/or write data to database. Heroku also talks to Facebook servers via Facebook's Open Graph API.

Below is the sample command in Heroku to create staging environment

```
$ heroku create --remote staging
```

Below are the sample commands in Heroku to add environmental variables in remote environment

```
$ heroku config:set S3_KEY=XXX --remote staging
```

```
$ heroku config:set S3_SECRET=YYY --remote staging
```

3.2.3 *MongoLab*

In this project there are three sets of mongo databases used: development, staging and production. It is important to keep the versions of databases since new versions of changes may include changes in database structure, so rolling back or forward the application version would not cause any error.

Below is the command on how to connect to remote Mongo Database that is hosted in MongoLab

```
$ mongo <servername>.mongolab.com:<portno>/<dbname> -u <dbuser> -p <dbpassword>
```

3.2.4 *Facebook*

Facebook is playing an important role in this project. Facebook provides user authentication and social media integration. Facebook allows connection using the Facebook API and Open Graph API.

3.2.5 *Client Browser*

Client browser uses HTTPS GET for static content, and HTTPS POST for AJAX request to Heroku. And client browser also connects to Facebook server directly using Facebook API and Open Graph API in HTTPS.

4. SYSTEM DESIGN

4.1 *Design Overview*

The GradeBadge application uses a client-server architecture model, and it uses the Model View Controller (MVC) architecture model for both client and server. All communication between client and server is done through AJAX requests submitted through HTTP POST and containing data encoded using JSON.

4.2 *Model View Controller Architecture*

This application is based on the Model View Controller (MVC) architecture, which is a software design pattern for separating different components of a software application [33]. There are three main components in the MVC architecture as follows.

- Model: The model includes data in the application and the business rules that apply to it.
- View: The view includes the UI components in the application. The UI components are responsible for presenting the model and for collecting user input.
- Controller: The controller is responsible for updating the data in the model and notifying the view about changes in the model.

4.3 Bandwidth Reduction Strategy

The following sections describe the three strategies used in this project to reduce consumption of bandwidth. Bandwidth is one of the major costs to consider when building a scalable application using cloud computing services, so minimizing bandwidth consumption becomes critical.

4.3.1 File Compression

In this application, in order to reduce the bandwidth consumption, files that are transferred from the application server will be compressed in the gzip format if the client browser supports it, which will result in smaller files. The following code shows how browser requests for static resources are tested to see if the browser supports gzip compression. This code appears in the req_file module.

```
if (file.gzip !== undefined &&
    req.headers['accept-encoding'] !== undefined &&
    req.headers['accept-encoding'].indexOf('gzip') !== -1)
{
    return app_http.replyCached(res, file.gzip, file.type,
                                file.etag, 'gzip');
} else {
    return app_http.replyCached(res, file.data,
                                file.type, file.etag);
}
```

In addition to checking whether the browser supports gzip compression, the above code also checks for any etag value sent by the browser. The etag functions as a version identifier, which the browser sends to the server with the request message to see whether the file is different from what is in the browser cache and what the server

would return. If the etag indicates that the file is current, the the server doesn't need to return the file; it simply returns a code of indicating that the browser's cached version of the file is current. The following shows how this is done in code.

```
if (req.headers['if-none-match'] === file.etag) {  
    return app_http.replyNotModified(res);  
}
```

4.3.2 Content Distribution Network Resources

In this application, we rely on content distribution networks (CDNs) to deliver some static resources to browsers without consuming bandwidth to the application server. For example, the application relies on the Bootstrap UI framework. The files for this framework are available through a CDN provided for free to application developers. The following code shows how GradeBadge loads the CSS component of the Bootstrap system.

```
<link href="//netdna.bootstrapcdn.com/twitter-bootstrap/  
2.2.2/css/bootstrap.css" rel="stylesheet">
```

4.3.3 Caching

In this application, we implemented a caching strategy in which every request is checked to see whether a needed file has been modified or is cached in the client browser. This is done by examining an etag value that identifies a version of the requested file. If the browser's version of the file is current, then the server replies with a code of 304, which indicates that the file is current. If the etag does not identify the current version of the file, then the server return the current version of the file. This strategy is implemented to reduce unnecessary consumption of bandwidth.

The following describes the four possible responses from the server when the browser requests a file. The source code is taken from the `app_http` modul.

- Reply Not Found: This is the response when the requested file is not found; the server returns the code 404, as shown in the following code snippet.

```
res.writeHead(404, {});
```

- Reply Not Modified: This is the response when the requested file is not modified as determined by comparing the browser's etag with the server's etag. The following shows how this is done in the code. Note that the server takes this opportunity to reset the expiration time of the requested file to one year later.

```
res.writeHead(304, {
  'Connection'      : 'keep-alive',
  'Proxy-Connection': 'keep-alive',
  'Cache-Control'   : 'max-age=31536000',
  'Expires'         : new Date(Date.now() +
    31536000000)
});
```

- Reply Not Cached: This is the response when the requested file is not cached in the client browser. The following shows how this is done in the code, where the server responds by returning the requested file.

```
res.writeHead(200, {
  'Content-Type'      : 'text/html',
  'Content-Length'    : buffer.length,
  'Connection'       : 'keep-alive',
  'Proxy-Connection' : 'keep-alive',
  'Pragma'            : 'no-cache',
```

```

    'Cache-Control'      : 'no-cache, no-store'
  });

```

- Reply Cached: This is the response when the requested file is cached in the client browser, in which case the server will reset the expiration date of the file to one year later. The following shows how this is done in the code.

```

res.writeHead(200, {
  'Content-Type'      : contentType,
  'Content-Length'    : buffer.length,
  'Connection'       : 'keep-alive',
  'Proxy-Connection' : 'keep-alive',
  'Pragma'            : 'public',
  'Cache-Control'     : 'max-age=31536000',
  'Vary'              : 'Accept-Encoding',
  'Expires'           : new Date(Date.now() +
    31536000000),
  'ETag'              : etag,
  'Content-Encoding'  : contentEncoding
});

```

4.4 Server-side Architecture Design

In Nodejs, components are organized into modules, which can function as namespaces. In this project, all Nodejs modules that start with req_ are request handlers that get requested from the router module. All ajax requests will go through the req_op module, which verifies that the user is logged into Facebook and the application version is current. Every request passing through req_op must contain a Facebook

access token and application version identifier. If the user is not logged into Facebook, then `req_op` returns the JSON document `login:true`. If the application version is not current, then `req_op` returns the JSON document `ver:true`.

4.4.1 Configuration

The following are the list of configuration files required on the server side; the source-code of these files can be found in Appendix A.

- `.env`: This is the setup file that contains environmental variables. This file only exists in the developer local environment. The same environmental variables need to be present in the Heroku execution environment, however, they are specified through another mechanism, namely through executing the Heroku `config` command for both staging and production environments.

The following is an example of the environmental variable values that could be set in the development environment. In this example, `FB_APP_ID` is the Facebook application ID and `FB_SECRET` is the Facebook application secret. (For each environment, we are using a different Facebook application instance.) `PORT` is the HTTP port number that the server will listen to for connections. `APP_VER` is the current application version. `MONGO_URI` is the mongo database connection string. (We use different database instances for each environment.)

```
FB_APP_ID=466760923387961
FB_SECRET=75e7a042473a989a3b876d3ec8749920
PORT=5000
APP_VER=2
MONGO_URI=mongodb://app:gradebadge123@
          ds045757.mongolab.com:45757/gb-d
```

- `.gitignore`: This is the setup file that contains matching patterns that tell git which files should be ignored (not placed under version control) when committing to a branch of the local repository.
- `.slugignore`: This is the setup file that contains a list of files or folders that will be ignored when calculating the slug limit in Heroku.
- `package.json`: This is the setup file that contains a list of Nodejs dependencies and engine version to use when deploying the application. This file also contains the application name, version and description, as shown in the following.

```
{
  "name": "gradebadge",
  "version": "0.0.1",
  "description": "Grade Badge System",
  "dependencies": {
    "mongodb": "1.2.13"
  },
  "engines": {
    "node": "0.8.21",
    "npm": "1.2.12"
  }
}
```

- `Procfile`: This is the setup file that tells Heroku how to launch the application, as shown in the following.

```
web: node main.js
```

4.4.2 Nodejs Modules

Below are the list of files that are used when the server executes; the source-code of these files can be found in the Appendix A.

- `main.js`: This is the main module in Nodejs, which contains code that verifies all necessary environmental variables are set correctly. It also invokes initialization functions of other modules to start the application. After initialization is complete, the main module starts the HTTP request handling loop.

The following is the part of the code in the main module where it initializes the model, router and fb modules and runs them asynchronously.

```
var n = 3;
function done() {
  if (--n === 0) {
    router.start();
  }
}
model .init(done);
router .init(done);
fb .init(done);
```

- `router.js`: This module routes incoming requests to the appropriate module. The following is the part of the code in the router module where it checks for pathname of the incoming request and routes the request to a corresponding module request handler.

```
function route(req, res) {
  var pathname = url.parse(req.url).pathname;
  if (pathname === '/') req_root.handle(req, res)
```

```

else if (pathname === verpath) req_app.handle(req, res)
else if (pathname === issuerpath) req_issuer.handle(req
    , res)
else if (pathname === '/mem') req_mem.handle(req, res)
else if (pathname === '/counters') req_counters.handle(
    req, res)
else req_rootdir.handle(req, res);
}

```

- app_ajax.js: This module contains the application-wide AJAX handling routines. The following is a sample function of the Ajax module, which is used to send data back to the browser in JSON format in the utf8 character encoding.

```

exports.data = function(res, data) {
    if (data === undefined) {
        data = {};
    }
    var buf = new Buffer(JSON.stringify({'data': data}), '
        utf8 ');
    res.writeHead(200, {
        'Content-Type': 'application/json; charset=UTF-8',
        'Content-Length': buf.length,
        'Pragma': 'no-cache',
        'Cache-Control': 'no-cache, no-store'
    });
    res.end(buf);
};

```

- app_http.js: This module contains all of the HTTP protocol routines for the

application. The use of caching headers, compression headers and other HTTP based optimizations are implemented in this module. This file was discussed in the caching strategy section.

- fb.js: This module contains all code that interacts with Facebook. The following is a sample initialization function, which will check the Facebook App ID and secret.

```
exports.init = function(cb) {
  var options = {
    hostname: 'graph.facebook.com',
    path: '/oauth/access_token?' +
          'client_id=' + process.env.FB_APP_ID +
          '&client_secret=' + process.env.FB_SECRET +
          '&grant_type=client_credentials ',
    method: 'GET'
  };
  send(options, function(data) {
    if (data instanceof Error) {
      throw data;
    }
    if (data.access_token === undefined) {
      throw new Error(
        'fb.init: access_token not returned by facebook.'
        +
        '\nfb.init: Facebook returned: ' + JSON.stringify
          (data)
      );
    }
  });
}
```

```

    }
    appToken = data.access_token;
    cb(appToken);
  });
};

```

- `logger.js`: This module contains application-wide logging functionalities. The following is a sample of error logging in this module. The application limits the number of error messages printed to avoid server slowdown in the case when there are many errors. A single error is cause for concern and study; recording a continuous stream of errors would be counter-productive and may expose the application more easily to denial of service attacks.

```

exports.errors = function(msg, opt_msg) {
  if (errorsPrinted < process.env.LOGGER_MAX_ERRORS) {
    ++errorsPrinted;
    print('ERROR', msg, opt_msg);
    if (errorsPrinted == process.env.LOGGER_MAX_ERROR) {
      console.log('MAX ERROR HIT');
    }
  }
  ++exports.errorsReceived;
};

```

- `model.js`: This module initializes the database connection pool during server start up. The following is the code used to establish this connection pool. Note that it uses the `MONGO_URI` value from an environmental variable. It also uses the connection options that were set in the `model` module.


```

MongoClient.connect(process.env.MONGO_URI, connectOptions
, function(err, db) {
    if (err) throw err;
    exports.db = db;
    cb();
});

```

- req_app.js: This module handles requests for the application's HTML template for badge earners. The following is the initialization function of the req_app module, where it returns the app.html template, replacing FB_APP_ID with the Facebook application ID provided through an environmental variable. To conserve bandwidth, the code uses a gzip compressed version of the file and sets the etag value.

```

exports.init = function(cb) {
    fs.readFile('app.html', 'utf8', function(err, file) {
        if (err) throw err;
        html = new Buffer(file.replace(/FB_APP_ID/g,
            process.env.FB_APP_ID), 'utf8');
        etag = app_http.etag(html);
        zlib.gzip(html, function(err, result) {
            if (err) throw err;
            ghtml = result;
            cb();
        });
    });
};

```

- req_counter.js: This module handles request for the logging counter. The follow-

ing is the request handler function of the req_counter module, where it constructs the page with logging information and sends it back to the browser in the utf8 character encoding and not cached.

```
exports.handle = function(req, res) {  
    var page =  
        '<p>logger errors: ' + logger.errorsReceived  
        + '</p>' +  
        '<p>logger warnings: ' + logger.  
        warningsReceived + '</p>' +  
        '<p>logger info: ' + logger.infoReceived  
        + '</p>' +  
        '<p></p>';  
    page = new Buffer(page, 'utf8');  
    app_http.replyNotCached(res, page);  
}
```

- req_file.js: This module handles requests for static content. The request handler function of this module is discussed in the Bandwidth Reduction Strategy chapter where it handles the etag and gzip file compression.

The following is the sample function where it calculates and displays memory consumption resulting from loading all static resources and compressing them. These resource are kept in memory at all times to eliminate access to the disk drive.

```
function displayStats(files) {  
    var uncompressed = 0, compressed = 0;  
    for (var i = 0; i < files.length; ++i) {  
        uncompressed += files[i].data.length;
```

```

        if (files[i].gzip !== undefined) compressed += files[i].gzip.length;
    }
    console.log('memfile bytes , uncompressed: ' +
        Math.ceil(uncompressed / 1024 / 1024) + ' MB');
    console.log('memfile bytes , compressed: ' +
        Math.ceil(compressed / 1024 / 1024) + ' MB');
}

```

- req.issuer.js: This module handles request for application HTML template for badge issuers. The following is the initialization function of the req.issuer module, where it returns the issuer.html template, replacing FB_APP_ID with the Facebook application ID provided through an environmental variable. To conserve bandwidth, the code uses a gzip compressed version of the file and sets the etag value.

```

exports.init = function(cb) {
    fs.readFile('issuer.html', 'utf8', function(err, file)
    {
        if (err) throw err;
        html = new Buffer(file.replace(/FB_APP_ID/g,
            process.env.FB_APP_ID), 'utf8');
        etag = app_http.etag(html);
        zlib.gzip(html, function(err, result) {
            if (err) throw err;
            ghtml = result;
            cb();
        });
    });
}

```

```
});
};
```

- req_mem.js: This module handles request for memory usage. The following is the request handler function of req_mem module where it construct the page with memory usage information and send it back to client in utf8 format and not cached.

```
exports.handle = function(req, res) {
  var usage = process.memoryUsage(),
      page = '<p>Heroku limit = 512 MB</p>' +
            '<p>rss = ' + Math.ceil(usage.rss
                                   / 1024 / 1024) + ' MB</p>' +
            '<p>heapTotal = ' + Math.ceil(usage.
                                           heapTotal / 1024 / 1024) + ' MB</p>' +
            '<p>heapUsed = ' + Math.ceil(usage.heapUsed
                                          / 1024 / 1024) + ' MB</p>';
  page = new Buffer(page, 'utf8');
  app.http.replyNotCached(res, page);
}
```

- req_root.js: This module handles request for static content under the root URL. The following is the request handler function of req_root module where it construct the html page that will redirect the page to the correct path with current version number.

```
var html = new Buffer('<script>location.replace("/' +
                     process.env.APP_VER + '/')</script>', 'utf8');
exports.handle = function(req, res) {
```

```

    app_http.replyNotCached(res , html);
};

```

- req_op.js : This module handles all AJAX request from client and routes to appropriate modules. The following is the part of request handler function where it checks the pathname of incoming request and call the appropriate module handler.

```

exports.handle = function(req , res) {
    app_ajax.parse(req , function(data) {
        var pathname = url.parse(req.url).pathname;
        if (pathname === '/op/save-group') {
            op_save_group.handle(data , res);
        } else if (pathname === '/op/read-groups-by-admin')
        {
            op_read_groups_by_admin.handle(data , res);
        });
    });
};

```

4.5 Mapping of Model Classes to MongoDB

There will be one node js module to represent the mongoDB collection, named model_(collection_name).js. And many-to-many relationships are represented by linking documents, named (a)_(b)_links

The following list below are the list of files that are used to Model to represent MongoDB, and the source-code of these files can be found in the Appendix B.

- model_group.js: this node.js module represents Groups collection. The following is one of functions in the model_group to get group document by given id.

```

exports.getByIds = function(group_ids , cb){
  model.db.collection('groups').find({'_id' : {$in:
    group_ids} }).toArray(function(err , groups){
    model.db.close();
    if (err) return cb(err);
    cb(groups);
  });
};

```

HERE

- model_badge.js: this node.js module represents Badges collection. The following is one of functions in the model_badge module to create badge document in the badge collection from given document.

```

exports.create = function(badge , cb) {
  model.db.collection('badges').insert(
    badge ,
    function(err) {
      model.db.close();
      if (err) return cb(err);
      cb();
    }
  );
};

```

- model_user.js: this node.js module represents Users collection. The following is one the functions in the model_user module to record the login activity of the user by updating the last login time stamp in the user document.

```

exports.login = function(user , cb) {
  model.db.collection( 'users ' ).save(
    user ,
    function(err) {
      model.db.close();
      if (err) return cb(err);
      cb();
    }
  );
};

```

- `model_group_admin.js`: this node.js module represents `group_admin_links` collection. The following is one of the function in the `model_group_admin` module to get the array of groups ids from given admin (user) id.

```

exports.getGroupIdsByAdminId = function(admin, cb) {
  model.db.collection( 'group_admin_links ', { 'gid' : true } )
    .find(admin).toArray(function(err , group_admin_links)
    {
      model.db.close();
      if (err) return cb(err);
      var group_ids = group_admin_links.map(function(
        group_admin_link) {return group_admin_link.gid;});
      cb(group_ids);
    });
};

```

- `model_group_member.js`: this node.js module represents `group_member_links` collection. The following is one of the function in the `model_group_member` module

to get the array member (user) ids from given group id.

```
exports.getMemberIdsByGroupId = function(group, cb) {
  model.db.collection('group-admin-links',{ 'uid' : true})
    .find(group).toArray(function(err, group-admin-links)
    {
      model.db.close();
      if (err) return cb(err);
      console.log('model-group-admin getMemberIdsByGroupId
        group-admin-links = '+ JSON.stringify(
        group-admin-links));
      var member_ids = group-admin-links.map(function(
        group-admin-link) {return group-admin-link.uid;});
      cb(member_ids);
    });
};
```

- model_user_badge.js: this node.js module represents user_badge_links collection. The following is one of the functions in model_user_badge module to create the link document.

```
exports.create = function(user_badge, cb) {
  model.db.collection('user_badge_links').insert(
    user_badge,
    function(err) {
      model.db.close();
      if (err) return cb(err);
      cb();
    }
  );
};
```



```
);
};
```

- `model_group_badge.js`: this node.js module represents `group_badge_links` collection. The following is one of the functions in `model_group_badge` module to create the link document.

```
exports.create = function(group_badge, cb) {
  model.db.collection('group_badge_links').insert(
    group_badge,
    function(err) {
      model.db.close();
      if (err) return cb(err);
      cb();
    }
  );
};
```

4.6 Request Handler Operation

There will be one node.js module to handle ajax request from client, named `op_(request).js`.

Every request may read, write or update to and from more than one collection

The following are the list of files that are used in request handler operation, and the source-code of these files can be found in the Appendix C

- `op_read_badges_by_group.js`: this operation is for request for all badges in the given group. The following shows how this is done in the code, where it gets the array of badge ids with given group id from `group_badge_links` document, and after that get the badges document based on the badge ids array result from

badge collection.

```
exports.handle = function (data, res) {  
  var group = { gid: data.gid };  
  model_group_badge.getBadgeIdsByGroupId(group, function(  
    bids) {  
    if (bids instanceof Error) {  
      return app_ajax.error(res);  
    }  
    model_badge.getByIds(bids, function(badges){  
      if (badges instanceof Error) {  
        return app_ajax.error(res);  
      }  
      return app_ajax.data(res, badges);  
    });  
  });  
};
```

- `op_read_groups_by_admin.js`: this operation is for request for all groups in the given admin. The following shows how this is done in the code, where it gets the array of group ids with given admin id from `group_admin` links document, and after that get the groups document based on the group ids array result from group collection.

```
exports.handle = function (data, res) {  
  var admin = { uid: data.uid };  
  model_group_admin.getGroupIdsByAdminId(admin, function(  
    gids) {  
    if (gids instanceof Error) {
```

```

        return app_ajax.error(res);
    }
    model_group.getByIds(gids, function(groups){
        if (groups instanceof Error) {
            return app_ajax.error(res);
        }
        return app_ajax.data(res, groups);
    });
});
};

```

- op_save_badge.js: this operation is for request for saving given badge details. The following shows how this is done in the code, where it adds badge document to badge collection, and get the newly created badge id together with group id then add to group_badge linking document.

```

exports.handle = function (data, res) {
    var badge = { name: data.name, desc: data.desc, pict:
        data.pict, gid: data.gid };
    model_badge.create(badge, function(err) {
        if (err) {
            return app_ajax.error(res);
        }
        var group_badge = { bid : badge._id, gid : badge.gid
            };
        model_group_badge.create(group_badge, function(err) {
            if (err) {
                return app_ajax.error(res);
            }
        });
    });
};

```

```

    }
  });
});
return app_ajax.data(res , {gid : badge._id} );
};

```

- op_save_group.js: this operation is for request for saving given group details. The following shows how this is done in the code, where it adds group document to group collection, and get the newly created group id together with user id then add to group_admin linking document.

```

exports.handle = function (data, res) {
  var group = { name: data.name, desc: data.desc , uid:
    data.uid };
  model_group.create(group , function(err) {
    if (err) {
      return app_ajax.error(res);
    }
    var group_admin = { gid : group._id , uid : group.uid
      };
    model_group_admin.create(group_admin , function(err) {
      if (err) {
        return app_ajax.error(res);
      }
    });
    return app_ajax.data(res , {gid : group._id} );
  });
};

```

4.7 Client-side Architecture Design

The following list below are the list of files that are used in client-side execution, and the source-code of these files can be found in the Appendix D.

- app.html : This is the html template for badge earners page. This file contains the HTML skeleton of the html page for badge earners. The following is the sample HTML code where in the template provides the List element then later JQuery will insert list item element into that.

```
<ul id="groups_list" class="thumbnails">
    <!-- auto generated -->
</ul>
```

- issuer.html : This is the html template for badge issuer page. This file contains the HTML skeleton of the html page for badge earners. The following is the sample HTML code where in the template provides the list element without any content, then later JQuery will insert the name of the user logged in.

```
<div class="nav-collapse collapse">
    <ul class="nav">
        <li id="name"></li>
        <li id="login" style="display:none;"></li>
    </ul>
</div>
```

- public_root/channel.html : This is the static content required by Facebook. The following is the content of channel.html, which refer to java-script file required by Facebook.

```
<script src="//connect.facebook.net/en_US/all.js"></
script>
```

- `public_root/favicon.ico` : This is the static content for icon use in the browser.
- `public_ver/app.js` : This is the client side java-script. The following is a sample function, which performs screen transition and initialize the new screen in this application.

```
a.screen = function(screenName, speed){
    if (speed === undefined) speed = 300;
    var newScreen = screens[screenName];
    if (newScreen.init) newScreen.init();
    currentScreen.transitionTo(speed, newScreen);
    currentScreen = newScreen;
};
```

5. DATABASE DESIGN

5.1 Overview

5.1.1 MongoDB

MongoDB is a scalable, high-performance, open source, NoSQL document-based database. MongoDB features include document-oriented storage, indexes, replication, high availability, auto-sharding, and querying. [21]

5.1.2 MongoLab

MongoLab is the cloud computing provider for MongoDB database that was used in this project [22]. MongoLab has a free tier service that facilitates experimentation by developers, and thus was convenient for this project.

5.1.3 Documents

Data in MongoDB is stored in documents, and every document must have a primary key named `_id`. Like regular SQL-based database, documents are like row in a table, where in MongoDB, documents have flexible schema, but every document must have `_id` field, even if you don't specify the `_id` field, mongoDB will add that automatically. [21]

Although document structure is not enforced in MongoDB, different data model and structure may have significant impacts on MongoDB and application performance. So it is good to keep some kind of structure or pattern in data model.

5.1.4 Collections

Documents in MongoDB are organized in collection, and basic database operations are performed based on collection. Indexes can be assigned in any field or subfield contained in documents within a MongoDB collection, and they are defined on per-collection level. [21]

5.2 Data Model

In this project, there are three main collections, User, Group and Badge Collection. And there are four linkings collections to represent the many-to-many relationship between the main collections.

- user : this collection contains the user information
- group : this collection contains the group information
- badge : this collection contains the badge information
- group_admin_links : this linking collection contains group_id and user_id, which represent the admin of a group
- group_member_links : this linking collection contains group_id and user_id, which represent the member of a group
- badge_user_links : this linking collection contains badge_id and user_id, which represent badges that user earned
- group_badge_links : this linking collection contains group_id and badge_id, which represent badges that belong to a group

5.2.1 User Collection

This collection contains of user documents, contain the following information, Table 5.1

Tab. 5.1: User Collection

Document Attribute	Sample Data	Description
_id	34728373	Facebook user ID
last_login	12/1/13 12:34:56 PM	User last login timestamp

5.2.2 Group Collection

This collection contains of group documents, Table 5.2

Tab. 5.2: Group Collection

Document Attribute	Sample Data	Description
_id	51466591b95b2b5023000001	Auto-generated Group ID
name	Yoga CSUSB	Group name
desc	This is Yoga group in CSUSB	Group description

5.2.3 Badge Collection

This collection contains of badge documents, Table 5.3

Tab. 5.3: Badge Collection

Document Attribute	Sample Data	Description
_id	514e34507075ae0c0f000001	Auto-generated Badge ID
name	Attended 10 classes	Badge name
desc	This is badge description	Badge description
pict	pict1.png	Badge Picture Reference

5.2.4 Group Admin Links Collection

This collection represent many-to-many relationship between group and user. User in this collection means the admin of the group. Table 5.4

Tab. 5.4: Group Admin Links Collection

Document Attribute	Sample Data	Description
_id	514e34507075ae0c0f000001	Auto-generated Document ID
gid	514e34507075ae0c0f000123	Group ID
uid	514e34507075ae0c0f000456	User ID as Admin

5.2.5 Group Member Links Collection

This collection represent many-to-many relationship between group and user. User in this collection means the member of the group. Table 5.5

Tab. 5.5: Group Member Links Collection

Document Attribute	Sample Data	Description
_id	514e34507075ae0c0f000002	Auto-generated Document ID
gid	514e34507075ae0c0f000123	Group ID
uid	514e34507075ae0c0f000675	User ID as Member

5.2.6 Badge User Links Collection

This collection represent many-to-many relationship between badge and user. User in this collection means the user who earned the badge. Table 5.6

Tab. 5.6: Badge User Links Collection

Document Attribute	Sample Data	Description
_id	514e34507075ae0c0f000005	Auto-generated Document ID
bid	514e34507075ae0c0f000975	Badge ID
uid	514e34507075ae0c0f000456	User ID as Earner

5.2.7 Group Badge Links Collection

This collection represent many-to-many relationship between group and badge. Badge in this collection means the badge in the group. Table 5.7

Tab. 5.7: Group Badge Links Collection

Document Attribute	Sample Data	Description
_id	514e34507075ae0c0f000004	Auto-generated Document ID
gid	514e34507075ae0c0f000158	Group ID
bid	514e34507075ae0c0f000953	Badge ID

6. PROJECT IMPLEMENTATION

The GradeBadge application is designed to work on mobile, tablet devices and desktop computers. The UI of the application is developed using Bootstrap. When a page requires the data to be loaded from server or modified or deleted, a request is sent to the Web server over HTTPS. The requests are sent to the Web server using Ajax. For handling Ajax requests and responses, this application uses JQuery Ajax API. All UI components are dynamically created or initialized in response to the data received from the Web server.

6.1 Loading Screen

When the GradeBoard application is loaded, a loading screen is presented to the user as shown in the Figure 6.1. The loading screen shows application logo and loading progress bar, and the screen is automatically redirected after the loading completed.

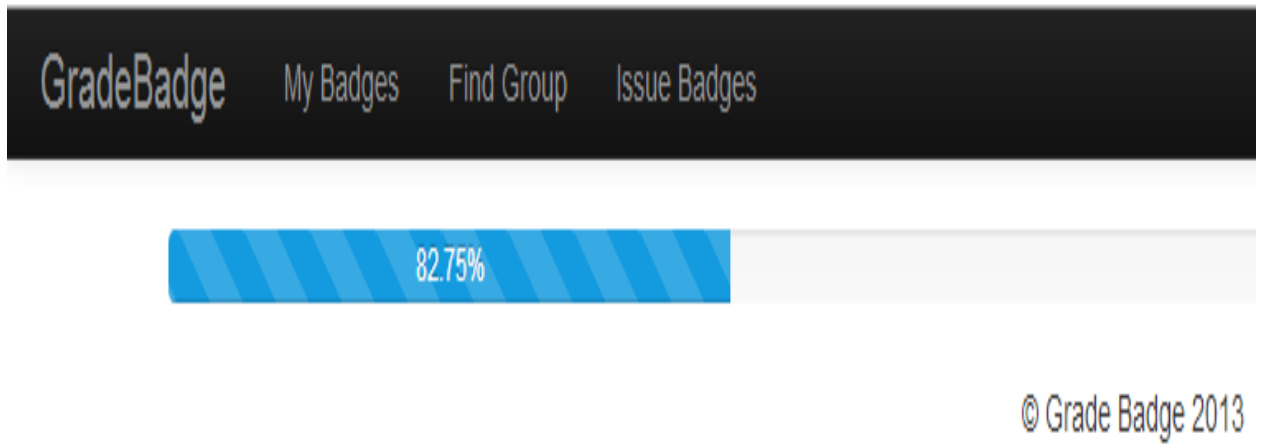


Fig. 6.1: GradeBadge Loading Screen

6.2 Login Screen

GradeBadge uses Facebook account for users to login. When the user is not logged-in to Facebook, the screen is automatically redirected to the Facebook login screen as shown in Figure 6.2. Every user in the system can be badge issuer and badge earner.

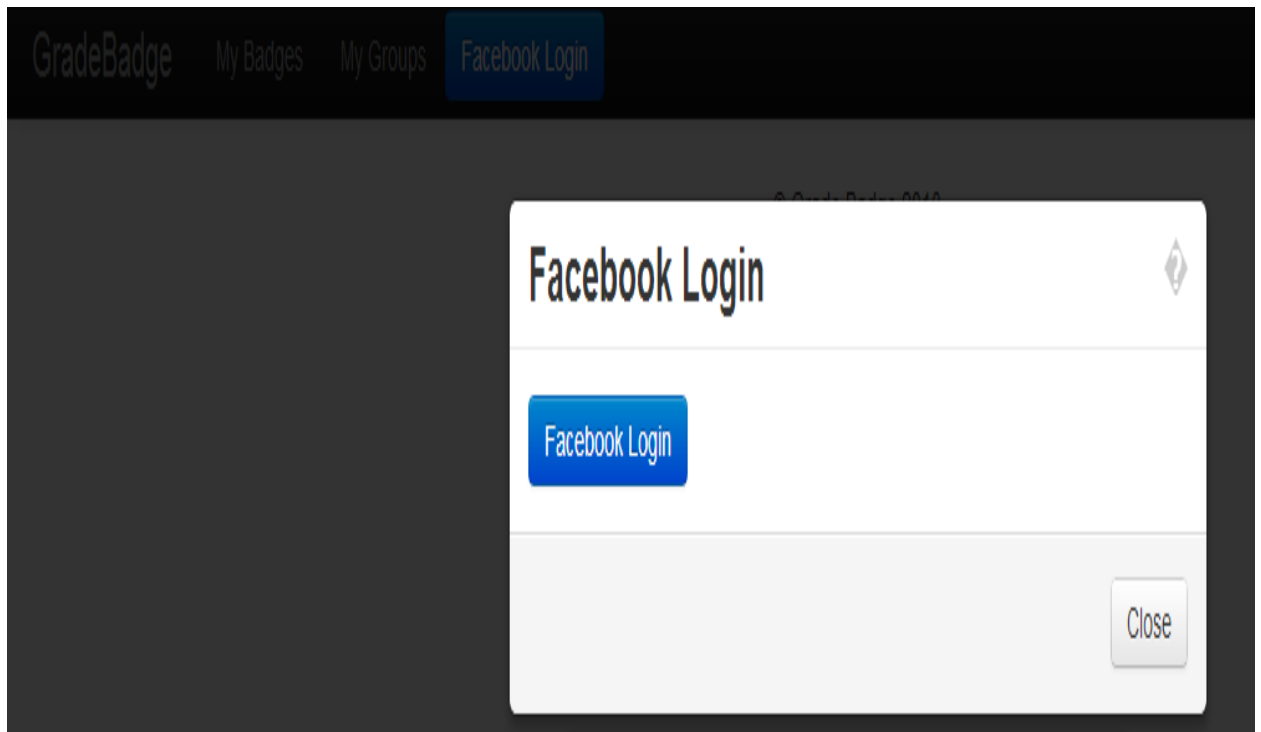


Fig. 6.2: GradeBadge Login Screen

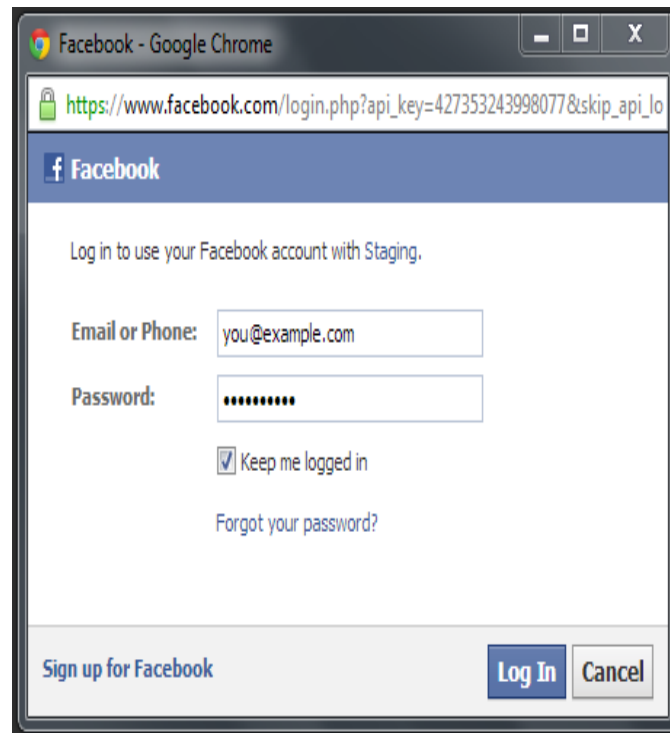


Fig. 6.3: Facebook Login Screen

6.3 Group Page

as shown in Figure 6.4.

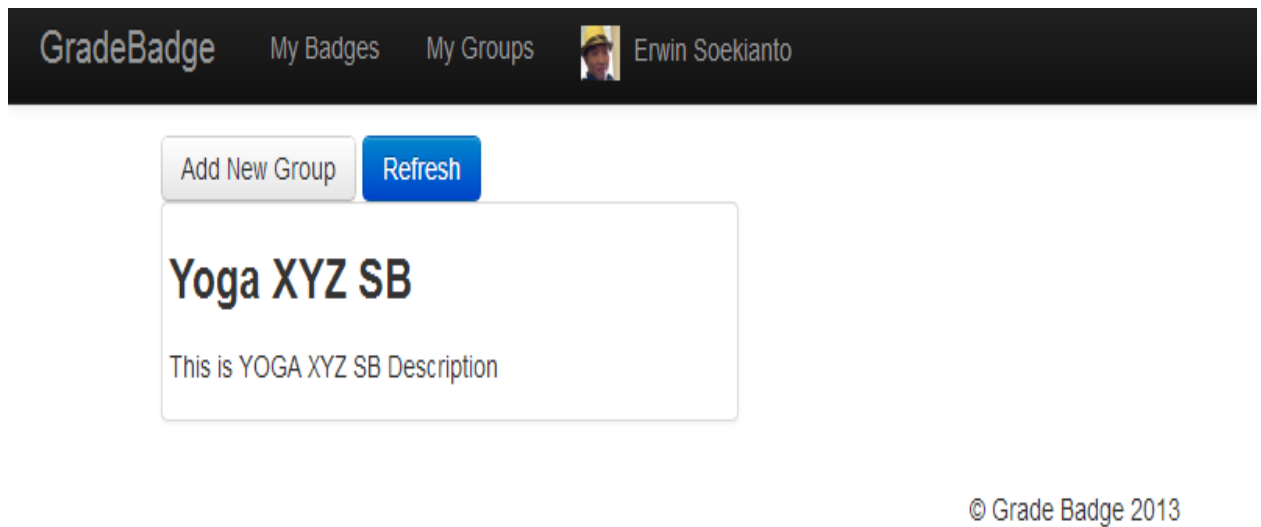


Fig. 6.4: GradeBadge Group Page

6.4 Add New Group Page

as shown in Figure 6.5.

The screenshot displays the GradeBadge application interface. The top navigation bar includes the 'GradeBadge' logo, links to 'My Badges' and 'My Groups', a user profile for 'Erwin Soekianto', and a 'Logout' button. The main content area features a sidebar with 'Add New Group' and 'Refresh' buttons. A group card for 'Yoga XYZ SB' is visible, with the description 'This is YOGA XYZ SB Description'. A modal window titled 'Add New Group' is open, showing input fields for 'Group Name' (filled with 'Yoga CSUSB') and 'Group Description' (filled with 'This is Yoga CSUSB'). The modal also includes 'Close' and 'Save changes' buttons.

Fig. 6.5: GradeBadge Add New Group Page

6.5 Group Page

as shown in Figure 6.6.

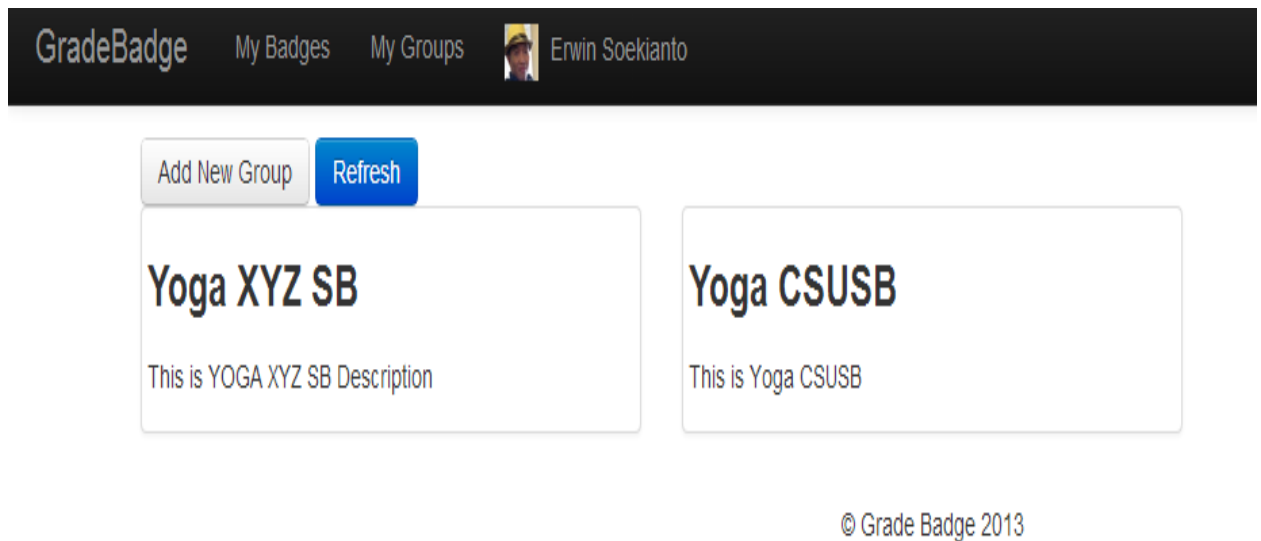


Fig. 6.6: GradeBadge Group Page Added

7. CONCLUSION AND FUTURE DIRECTION

7.1 *Conclusion*

GradeBadge is cloud-based web application that is hosted in Heroku, uses Nodejs and MongoDB, that provide responsive design that works well to be run inside browsers in desktop computers, tablets and smart phones. GradeBadge also enables users to login with their Facebook account to simplify authentication, and integrated with their Facebook.

Cloud computing provides significant cost-saving to developers to build application that can be scaled up or down almost instantly to accomodate rapidly changing demand.

7.2 *Future Direction*

GradeBadge application can be used as sample to showcase the development of cloud-based cross-platform application. This application can also be extended and enhanced in the future, as described in the following

- Auto-sharding: Implementation of auto-sharding in MongoDB to support greater scalability.
- Other social networking: Implementation of authentication and integration with other social networking applications such as Twitter, Tumblr, Google+, etc.
- Native App: Development of native IOS , Android and Windows Mobile Appli-

cation.

- API: Development of API to enable other applications to integrate a reward system module to their applications.

APPENDIX A

SERVER SIDE SOURCE CODE

```

//.env

FB_APP_ID=466760923387961

FB_SECRET=75e7a042473a989a3b876d3ec8749920

PORT=5000

APP_VER=2

MONGO_URI=mongodb://app:gradebadge123@ds045757.mongolab.com:45757/gb-d

//.gitignore

.DS_Store

node_modules

.env

//main.js

var http                = require('http');
var router              = require('./router');
var model               = require('./model');
var group               = require('./model_group');
var fb                  = require('./fb');
var logger              = require('./logger');

// TODO: minify js and css as part of deployment process.
// IDEA: minify at startup rather than as a build step.

// TODO: Check concepts against the following article.
// https://devcenter.heroku.com/articles/increasing-application-performance-with
-    http-cache-headers

// Check for required environmental variables.
if (process.env.PORT     === undefined) throw new Error('PORT not defined');
```

```

if (process.env.MONGO_URI === undefined) throw new Error('MONGO_URI not
    defined');

if (process.env.FB_APP_ID === undefined) throw new Error('FB_APP_ID not defined
    ');

if (process.env.FB_SECRET === undefined) throw new Error('FB_SECRET not defined
    ');

if (process.env.APP_VER === undefined) throw new Error('APP_VER not defined')
    ;

//check for max log, otherwise set to default value
if (process.env.LOGGER_MAX_WARNINGS === undefined) {
    process.env.LOGGER_MAX_WARNINGS = 16;
    logger.warning('LOGGER_MAX_WARNINGS not defined; defaulting to 16.');
```

}

```

if (process.env.LOGGER_MAX_ERRORS === undefined) {
    process.env.LOGGER_MAX_ERRORS = 16;
    logger.warning('LOGGER_MAX_ERRORS not defined; defaulting to 16.');
```

}

```

if (process.env.LOGGER_MAX_INFO === undefined) {
    process.env.LOGGER_MAX_INFO = 16;
    logger.warning('LOGGER_MAX_INFO not defined; defaulting to 16.');
```

}

```

// Trim for foreman.

process.env.PORT = process.env.PORT .replace(' ', '');
process.env.MONGO_URI = process.env.MONGO_URI .replace(' ', '');
process.env.FB_APP_ID = process.env.FB_APP_ID .replace(' ', ');

```



```

process.env.FB_SECRET = process.env.FB_SECRET .replace(' ', '');
process.env.APP_VER   = process.env.APP_VER   .replace(' ', '');

var n = 3;

function done() {

    if (--n === 0) {

        router.start();

    }

}

model .init(done);
router .init(done);
fb     .init(done);

//router.js

var http      = require('http');
var url       = require('url');
var app_http  = require('./app_http');
var req_root  = require('./req_root');
var req_mem   = require('./req_mem');
var req_app   = require('./req_app');
var req_issuer = require('./req_issuer');
var req_file  = require('./req_file');
var req_op    = require('./req_op');
var req_counters = require('./req_counters');

var verpath = '/' + process.env.APP_VER + '/';
var issuerpath = verpath + 'issuer';

var req_verdir = req_file.create('public_ver', verpath.length);
var req_rootdir = req_file.create('public_root', 1);

```

```

exports.init = function(cb) {

    var n = 4;

    function done() {

        if (--n === 0) cb();

    }

    req_verdir .init(done);
    req_rootdir .init(done);
    req_app     .init(done);
    req_issuer  .init(done);

};

function route(req, res) {

    var pathname = url.parse(req.url).pathname;

    if      (pathname === '/') req_root .
        handle(req, res)
    else if (pathname === verpath) req_app .
        handle(req, res)
    else if (pathname === issuerpath) req_issuer .
        handle(req, res)
    else if (pathname.substr(0, verpath.length) === verpath) req_verdir .
        handle(req, res)
    else if (pathname.substr(0, 4) === '/op/') req_op .
        handle(req, res)
    else if (pathname === '/mem') req_mem .
        handle(req, res)
    else if (pathname === '/counters') req_counters.
        handle(req, res)
}

```

```

else
    req_rootdir .

    handle(req, res);
}

function requestHandler(req, res) {

    // Make sure messages are sent over https when deployed through Heroku.
    // See https://devcenter.heroku.com/articles/http-routing
    if (req.headers['x-forwarded-proto'] === 'https' ||    // common case
        req.headers['x-forwarded-proto'] === undefined) { // local deployment
        route(req, res);
    } else {
        res.writeHead(302, { 'Location': "https://" + req.headers.host + req.url });
        res.end();
    }
}

exports.start = function() {

    http.createServer(requestHandler).listen(process.env.PORT, function(err) {

        if (err) console.log(err);

        else console.log("listening on " + process.env.PORT);

    });
};

```

APPENDIX B

MODEL CLASSES SOURCE CODE

```

// model_group.js

var assert = require('assert');
var model = require('./model');

exports.create = function(group, cb) {
  model.db.collection('groups').insert(
    group,
    function(err) {
      model.db.close();
      if (err) return cb(err);
      cb();
    }
  );
};

exports.getByIds = function(group_ids, cb){
  model.db.collection('groups').find({'_id' : {$in: group_ids} }).toArray(
    function(err, groups){
      model.db.close();
      if (err) return cb(err);
      console.log('model_group getByIds groups array = ' + JSON.stringify(groups));
      cb(groups);
    }
  );
};

exports.getAll = function(cb){
  model.db.collection('groups').find().toArray(function(err, groups){
    model.db.close();
    if (err) return cb(err);
  });
};

```

```

        console.log('model_group getByIds groups array = ' + JSON.stringify(groups));

        cb(groups);

    });

};

exports.readGroupMembers = function(group, cb) {

    model.db.collection('group_member_links',{'uid' : true}).find(group).toArray(

        function(err, group_member_links){

            if (err) {model.db.close(); return cb(err);}

            console.log('model_group readGroupMembers group_member_links = ' + JSON.

                stringify(group_member_links));

            var member_ids = group_member_links.map(function(group_member_link) {return

                group_member_link.gid;});

            model.db.collection('users').find({'_id' : {$in: member_ids} }).toArray(

                function(err, members){

                    model.db.close();

                    if (err) return cb(err);

                    console.log('model_group readGroupMembers members array = ' + JSON.

                        stringify(members));

                    group.members = members;

                    cb();

                });

        });

};

};

```

APPENDIX C

REQUEST HANDLER OPERATION SOURCE CODE

```

//op_read_badges_by_group.js

var model_badge      = require('./model_badge');
var model_group_badge = require('./model_group_badge');
var app_ajax         = require('./app_ajax');

exports.handle = function (data, res) {

  console.log('op_read_badges_by_group  input = ' + JSON.stringify(data));

  var group = { gid: data.gid };

  model_group_badge.getBadgeIdsByGroupId(group, function(bids) {

    if (bids instanceof Error) {

      logger.error(__filename + ' : model_group_badge.getBadgeIdsByGroupId : ' +
        bids.message);

      return app_ajax.error(res);

    }

    console.log('group_badges is read = ' + JSON.stringify(bids));

    model_badge.getByIds(bids, function(badges){

      if (badges instanceof Error) {

        logger.error(__filename + ' : model_badge.getByIds : ' + badges.message)

          ;

        return app_ajax.error(res);

      }

      console.log('badges is read = ' + JSON.stringify(badges));

      return app_ajax.data(res, badges);

    });

  });

};

//op_save_badges.js

var model_badge      = require('./model_badge');

```



```

var model_group_badge = require('./model_group_badge');

var app_ajax          = require('./app_ajax');

var logger            = require('./logger');

exports.handle = function (data, res) {

  //console.log('op_save_badge input = ' + JSON.stringify(data));

  var badge = { name: data.name, desc: data.desc, pict: data.pict, gid: data.gid

    };

  model_badge.create(badge, function(err) {

    if (err) {

      logger.error(__filename + ' : save_badge model_badge : ' + err.message);

      return app_ajax.error(res);

    }

    console.log('group created with id = ' + badge._id);

    var group_badge = { bid : badge._id, gid : badge.gid };

    model_group_badge.create(group_badge, function(err) {

      if (err) {

        logger.error(__filename + ' : save_badge model_group_badge : ' + err.

          message);

        return app_ajax.error(res);

      }

      console.log('group_badge created with id = ' + group_badge._id);

    });

  });

  return app_ajax.data(res, {gid : badge._id} );

};

```

APPENDIX D

CLIENT SIDE SOURCE CODE

```

//public_root\channel.html

<script src="//connect.facebook.net/en_US/all.js"></script>

//public_ver\app.js

window.a = {};

a.creds = {};

a.m = {};

a.c = {};

a.v = {};

/* -----SCREENS----- */

$(function() {

    var screens = {};

    (function () {

        Screen = function(name) {

            this.name = name;

            this.$mainDiv = $('#'+name);

        };

        Screen.prototype.transitionTo = function(speed, newScreen) {

            this.$mainDiv.fadeOut(speed, function() {

                newScreen.$mainDiv.fadeIn(speed);

            });

        };

        screens.loading      = new Screen('loading');

        screens.groups       = new Screen('groups');

        screens.title        = new Screen('title');

        screens.badges       = new Screen('badges');

```

```

    screens.myBadges    = new Screen('myBadges');
    screens.profile     = new Screen('profile');
    screens.login       = new Screen('login');
    screens.groupBadges = new Screen('groupBadges');
    screens.badgeMembers= new Screen('badgeMembers');
    screens.findGroups  = new Screen('findGroups');
  }());

```

```

var currentScreen = screens.loading;

```

```

a.screen = function(screenName, speed){
  if (speed === undefined) speed = 300;
  var newScreen = screens[screenName];
  if (newScreen.init) newScreen.init();
  currentScreen.transitionTo(speed, newScreen);
  currentScreen = newScreen;
};

```

```

a.refresh = function(){
  currentScreen.refresh();
};

```

```

a.rebuild = function(){
  currentScreen.rebuild();
};

```

```

screens.findGroups.rebuild = function(){
  $('#groups_list > li').remove();
  a.m.anyGroups.forEach(function(group, i){

```

```

    $('#groups_list').append(
        '<li class="span4">' +
            '<div class="thumbnail">' +
                '<h3>' + group.name + '</h3>' +
                '<p>' + group.desc + '</p>' +
                '<button id="group_edit" class="btn btn-primary" onclick="screens.
                    findGroups.joinGroupBtn('+i+')" type="button">' +
                    '<i class="icon-certificate icon-white"></i> Join</button>' +
                '</div>' +
            '</li>'
    );
});
};

```

```

screens.findGroups.joinGroupBtn = function(i){
    a.m.joinGroup(a.m.anyGroups[i]._id, function(){
        if (err)
            alert('Err' + err.message);
        else
            a.screens('badges');
    });
};

```

```

screens.findGroups.refresh = function(){
    a.m.readAnyGroups(function() {
        screens.findGroups.rebuild();
    });
};

```

```

screens.findGroups.init = function(){
    console.log('findGroups init');
    if (a.m.anyGroups === undefined) screens.findGroups.refresh();
};

screens.title.init = function(){
    FB.api('/me', function(response) {
        console.log('Screen init title');
        $('#name').html('<a href="#" onclick="a.screen(\'profile\')"> '+response.name+'</a>');
    });
};

screens.login.init = function(){
    screens.login.$mainDiv.html('<button class="btn btn-primary" type="button"
        onclick="a.fbLogin()">Facebook Login</button>');
    $('#loginModal').modal('show');
};

screens.groupBadges.init = function(){
    console.log('Curr Group = ' + JSON.stringify(a.v.currGroup));
    $('#groupname').html('<h2>'+ a.v.currGroup.name +'</h2>');
    if (a.v.currGroup.badges === undefined) screens.groupBadges.refresh();
};

screens.groupBadges.refresh = function(){
    a.m.readGroupBadges(function() {
        screens.groupBadges.rebuild();
    });
};

```

```

    });
};

screens.badgeMembers.init = function(){
    console.log('Curr Badge = ' +JSON.stringify(a.v.currBadge));
    $('#badgename').html('<h2>'+ a.v.currBadge.name + '</h2>');
    if (a.v.currBadge.members === undefined) screens.badgeMembers.refresh();
};

screens.badgeMembers.refresh = function(){
    a.m.readBadgeMembers(function() {
        screens.badgeMembers.rebuild();
    });
};

screens.groups.refresh = function(){
    a.m.readGroups(function() {
        screens.groups.rebuild();
    });
};

screens.badgeMembers.rebuild = function(){
    $('#badge_members_list > li').remove();
    a.v.currBadge.members.forEach(function(member, i){
        $('#badge_members_list').append(
            '<li class="span4">' +
            '<div class="thumbnail">' +
            '<h3>'+ member.name + '</h3>' +
            '<p>'+ member.uid + '</p>' +

```

```

        '<p>Status earned</p>' +

        '<p>photo, # badge earned</p>' +

        '<button id="group_edit" class="btn btn-primary" onclick="" type="

            button">' +

            '<i class="icon-certificate icon-white"></i> Assign (not done)</

            button>' +

        '</div>' +

        '</li>'

    );

});

};

screens.groupBadges.rebuild = function(){

    $('#group_badges_list > li').remove();

    a.v.currGroup.badges.forEach(function(badge, i){

        $('#group_badges_list').append(

            '<li class="span4">' +

            '<div class="thumbnail">' +

            '<h3>'+ badge.name + '</h3>' +

            '<p>'+ badge.desc + '</p>' +

            '<p>'+ badge.pict + '</p>' +

            '<button id="assign_badge" class="btn btn-primary" onclick="

                screens.groupBadges.selectBadgeBtn('+i+') " type="button">' +

            '<i class="icon-user icon-white"></i> Assign Badge</button>' +

            '</div>' +

            '</li>'

        );

    });

};

```



```

screens.groupBadges.selectBadgeBtn = function(i) {

    a.v.currBadge = a.v.currGroup.badges[i];

    screens.groupBadges.rebuild();

    a.screen('badgeMembers');

};

screens.groups.rebuild = function(){

    $('#groups_list > li').remove();

    a.m.groups.forEach(function(group, i){

        $('#groups_list').append(

            '<li class="span4">' +

                '<div class="thumbnail">' +

                    '<h3>'+ group.name + '</h3>' +

                    '<p>'+ group.desc + '</p>' +

                    '<button id="group_edit" class="btn btn-primary" onclick="screens.

                        groups.selectGroupBtn('+i+') " type="button">' +

                        '<i class="icon-certificate icon-white"></i> Badges</button>' +

                    '</div>' +

                '</li>'

        );

    });

};

screens.groups.selectGroupBtn = function(i) {

    a.v.currGroup = a.m.groups[i];

    screens.groupBadges.rebuild();

    a.screen('groupBadges');

};

```

```

screens.groups.init = function(){
    if (a.m.groups === undefined) screens.groups.refresh();
};

});

/*-----  END OF SCREENS ----- */

/*-----  MODEL ----- */
a.m.readGroups = function(cb) {
    $.ajax({
        url: '/op/read-groups-by-admin',
        type: 'post',
        dataType: 'json',
        cache: false,
        data: JSON.stringify( { 'accessToken': a.creds.accessToken } )
    })
    .done(function(data) {
        console.log('app.js data = ' + JSON.stringify(data));
        //if (data.login !== undefined) {
            //a.relogin(function() { $saveBtn.click(); });
        //} else
        if (data.error !== undefined) {
            console.log('error = ' + data.error);
            cb(data.error);
        } else {
            console.log('group is read');
            a.m.groups = data.data;
            a.m.groups.sort(function(a,b) { return a.name < b.name ? -1 : 1});

```

```

        cb();
    }
})

.fail(function(jqxhr, textStatus, errorThrown) {
    if (errorThrown) console.log('Error: ' + errorThrown);
    else console.log('Error: ' + textStatus);
});
};

a.m.readAnyGroups = function(cb) {
    $.ajax({
        url: '/op/read-any-groups',
        type: 'post',
        dataType: 'json',
        cache: false,
        data: JSON.stringify( { 'accessToken': a.creds.accessToken } )
    })

    .done(function(data) {
        console.log('app.js data = ' + JSON.stringify(data));
        //if (data.login !== undefined) {
            //a.relogin(function() { $saveBtn.click(); });
        //} else
        if (data.error !== undefined) {
            console.log('error = ' + data.error);
            cb(data.error);
        } else {
            console.log('group is read');
            a.m.anyGroups = data.data;
            a.m.anyGroups.sort(function(a,b) { return a.name < b.name ? -1 : 1});
        }
    })
};

```

```

        cb();
    }
})

.fail(function(jqxhr, textStatus, errorThrown) {
    if (errorThrown) console.log('Error: ' + errorThrown);
    else console.log('Error: ' + textStatus);
});
};

a.m.joinGroup = function(gid, cb) {
    $.ajax({
        url: '/op/join-group',
        type: 'post',
        dataType: 'json',
        cache: false,
        data: JSON.stringify( { 'accessToken': a.creds.accessToken, 'gid': gid } )
    })

    .done(function(data) {
        console.log('app.js data = ' + JSON.stringify(data));
        //if (data.login !== undefined) {
            //a.relogin(function() { $saveBtn.click(); });
        //} else
        if (data.error !== undefined) {
            console.log('error = ' + data.error);
            cb(data.error);
        } else {
            console.log('group_member linked');
            cb();
        }
    })
}

```

```

    })

    .fail(function(jqxhr, textStatus, errorThrown) {

        if (errorThrown) console.log('Error: ' + errorThrown);

        else console.log('Error: ' + textStatus);

    });

};

a.m.readGroupBadges = function(cb) {

    $.ajax({

        url: '/op/read-badges-by-group',

        type: 'post',

        dataType: 'json',

        cache: false,

        data: JSON.stringify( { 'accessToken': a.creds.accessToken, 'gid' : a.v.

            currGroup._id } )

    })

    .done(function(data) {

        console.log('app.js data = ' + JSON.stringify(data));

        //if (data.login !== undefined) {

            //a.relogin(function() { $saveBtn.click(); });

        //} else

        if (data.error !== undefined) {

            console.log('error = ' + data.error);

            cb(data.error);

        } else {

            console.log('badges are read');

            a.v.currGroup.badges = data.data;

            a.v.currGroup.badges.sort(function(a,b) { return a.name < b.name ? -1 :

                1});

        }

    });

};

```

```

        cb();
    }
})

.fail(function(jqxhr, textStatus, errorThrown) {
    if (errorThrown) console.log('Error: ' + errorThrown);
    else console.log('Error: ' + textStatus);
});
};

a.m.readBadgeMembers = function(cb) {
    $.ajax({
        url: '/op/read-badge-members',
        type: 'post',
        dataType: 'json',
        cache: false,
        data: JSON.stringify( { 'accessToken': a.creds.accessToken, 'gid' : a.v.
            currGroup._id, 'bid' : a.v.currBadge._id } )
    })
    .done(function(data) {
        console.log('app.js data = ' + JSON.stringify(data));
        //if (data.login !== undefined) {
            //a.relogin(function() { $saveBtn.click(); });
        //} else
        if (data.error !== undefined) {
            console.log('error = ' + data.error);
            cb(data.error);
        } else {
            console.log('members are read');
            a.v.currBadge.members = data.data;
        }
    })
};

```

```

        a.v.currBadge.members.sort(function(a,b) { return a.name < b.name ? -1 :
            1});

        cb();

    }

    })

    .fail(function(jqxhr, textStatus, errorThrown) {

        if (errorThrown) console.log('Error: ' + errorThrown);

        else console.log('Error: ' + textStatus);

    });

};

/*----- END OF MODEL ----- */

/*----- CONTROLLER ----- */

a.c.saveNewGroup = function() {

    //console.log(JSON.stringify( { 'name': $('#gname').val(), 'desc': $('#gdesc')
        .val(), 'accessToken': a.creds.accessToken } ));

    $.ajax({

        url: '/op/save-group',

        type: 'post',

        dataType: 'json',

        cache: false,

        data: JSON.stringify( { 'name': $('#gname').val(), 'desc': $('#gdesc').val()
            , 'accessToken': a.creds.accessToken } )

    })

    .done(function(data) {

        console.log(JSON.stringify(data));

        if (data.login !== undefined) {

```

```

        a.fbReLogin(function() { $saveBtn.click(); });
    } else
    if (data.error !== undefined) {
        console.log('error = ' + data.error);
        //$numDiv.html(data.error);
    }else{
        console.log('group is saved, id = ' + JSON.stringify(data.data));
        $('#addGroupModal').hide();
        //a.screen('groups');
        a.m.groups.push({ '_id': data.data.gid, 'name': $('#gname').val(), 'desc':
            $('#gdesc').val() });
        a.rebuild();
    }
})
.fail(function(jqxhr, textStatus, errorThrown) {
    if (errorThrown) console.log('Error: ' + errorThrown);
    else console.log('Error: ' + textStatus);
});
};

a.c.saveNewBadge = function() {
    $.ajax({
        url: '/op/save-badge',
        type: 'post',
        dataType: 'json',
        cache: false,
        data: JSON.stringify( { 'name': $('#bname').val(), 'desc': $('#bdesc').val()
            , 'pict': $('#bpict').val(), 'gid': a.v.currGroup._id, 'accessToken': a.
            creds.accessToken } )
    });
}

```



```

    })

    .done(function(data) {

        console.log(JSON.stringify(data));

        if (data.login !== undefined) {

            a.fbReLogin(function() { $saveBtn.click(); });

        } else if (data.error !== undefined) {

            console.log('error = ' + data.error);

            //$numDiv.html(data.error);

        } else {

            console.log('badge is saved, id = ' + JSON.stringify(data.data));

            $('#addBadgeModal').hide();

            a.screen('groupBadges');

            a.v.currGroup.badges.push({'_id': data.data.bid, 'name': $('#bname').val(),

                , 'desc': $('#bdesc').val(), 'pict': $('#bpict').val(), 'gid': a.v.

                currGroup._id });

            a.rebuild();

        }

    })

    .fail(function(jqxhr, textStatus, errorThrown) {

        if (errorThrown) console.log('Error: ' + errorThrown);

        else console.log('Error: ' + textStatus);

    });

};

/* ----- END OF CONTROLLER ----- */

/*----- VIEW ----- */

```

```

a.v.currGroup = null;

a.v.currBadge = null;

/* ----- END OF VIEW ----- */

/* -----FB Business----- */
a.fbRelogin = function(cb) {
console.log('a.relogin()');
    FB.getLoginStatus(function(response) {
        if (response.status === 'connected') {
            a.creds.uid = response.authResponse.userID;
            a.creds.accessToken = response.authResponse.accessToken;
            cb();
        } else if (response.status === 'not_authorized') {
            //a.screen.next('login');
        } else {
            //var $relogin = $('<button>Login to Facebook</button>');
            //$('.screen').hide();
            //$('.body').append($relogin);
            //$relogin.click(function() {
            //    $relogin.remove();
            //    $('.screen').show();
            //    a.login(cb);
            //});
        }
    });
};

```

```

a.fbLogin = function(cb) {
    FB.login(function(response) {
        if (response.authResponse) {
            a.creds.uid = response.authResponse.userID;
            a.creds.accessToken = response.authResponse.accessToken;
            $('#loginModal').modal('hide');
            a.screen('title');
            //cb('');
        } else {
            a.screen('login');
            //cb('Login failed.');
```

```

        }
    });
};

a.fbInit = function (fbAppId) {
    FB.init({
        appId      : fbAppId,
        channelUrl  : '://' + window.location.host + '/channel.html',
        status      : false, // check the login status upon init?
        cookie      : false, // set sessions cookies?
        xfbml      : false
    });
    FB.Canvas.setAutoGrow();
    FB.getLoginStatus(function(response) {
        if (response.status === 'connected') {
            a.creds.uid = response.authResponse.userID;
            a.creds.accessToken = response.authResponse.accessToken;
```

```
        a.screen('title');
    } else {
        a.screen('login');
    }
});
};

/* ----- */

fbAsyncInit();
```

REFERENCES

- [1] Ajax (programming). [http://en.wikipedia.org/wiki/Ajax_\(programming\)](http://en.wikipedia.org/wiki/Ajax_(programming)).
- [2] Amazon Web Services. <http://aws.amazon.com/>.
- [3] Android Mobile Computing Platform.
<http://developer.android.com/about/index.html>.
- [4] Bootstrap. <http://twitter.github.com/bootstrap/>.
- [5] Dom Document Object Model. <http://www.w3.org/DOM>.
- [6] Eclipse Founcation. <http://www.eclipse.org/>.
- [7] Entities, Properties, and Keys.
<https://developers.google.com/appengine/docs/java/datastore/entities>.
- [8] Facebook Developers. <https://developers.facebook.com>.
- [9] Github. <https://github.com/>.
- [10] Glossary. <http://technet.microsoft.com/en-us/library/bb742416.aspx>.
- [11] Google Developers Academy.
<https://developers.google.com/appengine/training/intro/whatisgae/>.
- [12] Heroku How It Works. <http://www.heroku.com/how>.
- [13] How Entities and Indexes are Stored.
https://developers.google.com/appengine/articles/storage_breakdown.
- [14] Http secure. <http://en.wikipedia.org/wiki/HTTPS>.
- [15] Index Definition and Structure.
<https://developers.google.com/appengine/docs/python/datastore/indexes>.

- [16] IOS Mobile Computing Platform. <http://en.wikipedia.org/wiki/IOS>.
- [17] Jetty (web server). [http://en.wikipedia.org/wiki/Jetty_\(web_server\)](http://en.wikipedia.org/wiki/Jetty_(web_server)).
- [18] JQuery Mobile 1.2 Reference Document.
<http://www.jquerymobile.com/demos/1.2.0/docs/about/features.html>.
- [19] JQuery Official Document. <http://www.jquery.com/>.
- [20] Json. <http://en.wikipedia.org/wiki/Json>.
- [21] Mongodb Agile and Scalable. <http://www.mongodb.org/>.
- [22] Mongolab. <http://www.mongodb.org/>.
- [23] Node.js Manual and Documentation. <http://nodejs.org/api/>.
- [24] Nosql. <http://en.wikipedia.org/wiki/NoSQL/>.
- [25] Official Documentation of Google App Engine Java Datastore.
<https://developers.google.com/appengine/docs/java/datastore/>.
- [26] Official Java Technology Document.
http://www.java.com/en/download/faq/whatis_java.xml.
- [27] PolyModel.
<https://developers.google.com/appengine/docs/python/ndb/polymodelclass>.
- [28] Unified Modeling Language. <http://www-01.ibm.com/software/rational/uml/>.
- [29] W3C community. <http://www.w3.org/TR/REC-html40/>.
- [30] What is an Object?
<http://docs.oracle.com/javase/tutorial/java/concepts/object.html>.
- [31] Windows Azure. <http://www.windowsazure.com/en-us/develop/overview/>.
- [32] Manoj Kulkarni. *GradeBoard: A Cloud-Based Solution for a Student Grading System*. CSUSB - Masters Project, 2013.
- [33] Ph.D Steve Burbeck. Mvc How to use Model-View-Controller.
<http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html>.