# Independent Study Report

Dr. Jianwei Niu  
Shawn Aten (hfl398)  
May 5, 2016

## Overview

This independent study comprised of two projects, the first being a modification and extension of the second. The first project is a RESTful API for a customer service ticket system for a bank, implemented in Node. The second project is an extension and modification of the first project. The idea was to convert it to a generic messaging service, not one for a bank customer support system. Further it was to have a mechanism to support real-time messages. It was also eventually used by students in Dr. Niu's Software Engineering course for a final project.

Each project was setup as a group of Docker containers. Docker is an implementation of Linux containers. Containers can be thought of as extremely lightweight virtual machines. Below are instructions on how deploy the services in the ways they were used during the study. They were setup locally, on the Digital Ocean cloud, and on a local OpenStack cloud hosted by ICS.

## Setup

### Create Mailgun account

1. Clone or download the repository for each project.
2. Create an account at <http://mailgun.com> and setup a sandbox domain.
3. Modify either the sample.env (project 1) or sample.common.env (project 2) file to include your private API key and sandbox domain. Rename these files to .env and common.env.

### Install Docker

Install Docker by following these [instructions](https://docs.docker.com/engine/installation/). If you're on Linux please also install [docker-compose](https://docs.docker.com/compose/install/) and [docker-machine](https://docs.docker.com/machine/install-machine/). If you're on OS X or Windows you'll get them as part of the Docker installation. If you're in a terminal and can successfully run docker ps and docker run --rm hello-world you can proceed.

### Run Service

1. For project 1 modify the .env file so that the ENV variable is set to "dev" or "prod" and the BASE\_URL variable is set to the ip of the machine where docker is running. For project 2 you just need to modify the BASE\_URL field in docker-compose.yml (for local deployment) or production.yml (for production deployment) to the appropriate ip. If you're using a local Linux machine, use local ip of your machine. If using anything else, just run docker-machine ls and use the ip listed there.
2. To start the service run docker-compose up -d from the project root directory. This will cause docker images to be downloaded and built and the project's containers to start running. For project 2 you also need to run docker-compose --rm dbload to populate the mock bank DB.
3. *Optional* You can view the logs output for each container (think service) by running docker-compose logs <SERVICE>. The possible services are hapi (webserver) and db.
4. *Optional* To access the DB directly you can run docker-compose run --rm db /bin/bash followed by mongo --host db in the new shell. You're now in a mongo client session.
5. Look at the endpoint specifications in each repo to see the endpoints available and example curl commands to test them. They're in a file called spec.md in the root directory. Make sure you use the ip from step 1. For project 1 you must create users with information matching what's in misc/bank\_db.json.

### Remote / Cloud Machines

1. To deploy on an OpenStack cloud we will use docker-machine to create a remote VM and then control it just like above. Download your OpenStack RC file from Horizon and run the script. Then run docker-machine create -d openstack my-machine and eval $(docker-machine env my-machine). Then run the commands as before. This time docker-machine will send the commands to the cloud VM and start the service there.
2. To deploy to Digital Ocean first get an API token for your account. Then it's nearly the same as for OpenStack. Run docker-machine create -d digitalocean --digitalocean-access-token <TOKEN> my-machine and eval $(docker-machine env my-machine).

## Project Details

Each project details section contains an outline of the project, results, any specific setup instructions needed that add to what's above, and a UML class diagram. The REST endpoints specification for each project is in a file called spec.md in the root directory of each repo.

## Project 1: Bank Support Ticket System

<https://github.com/shawnaten/node-bank-ticket-api>

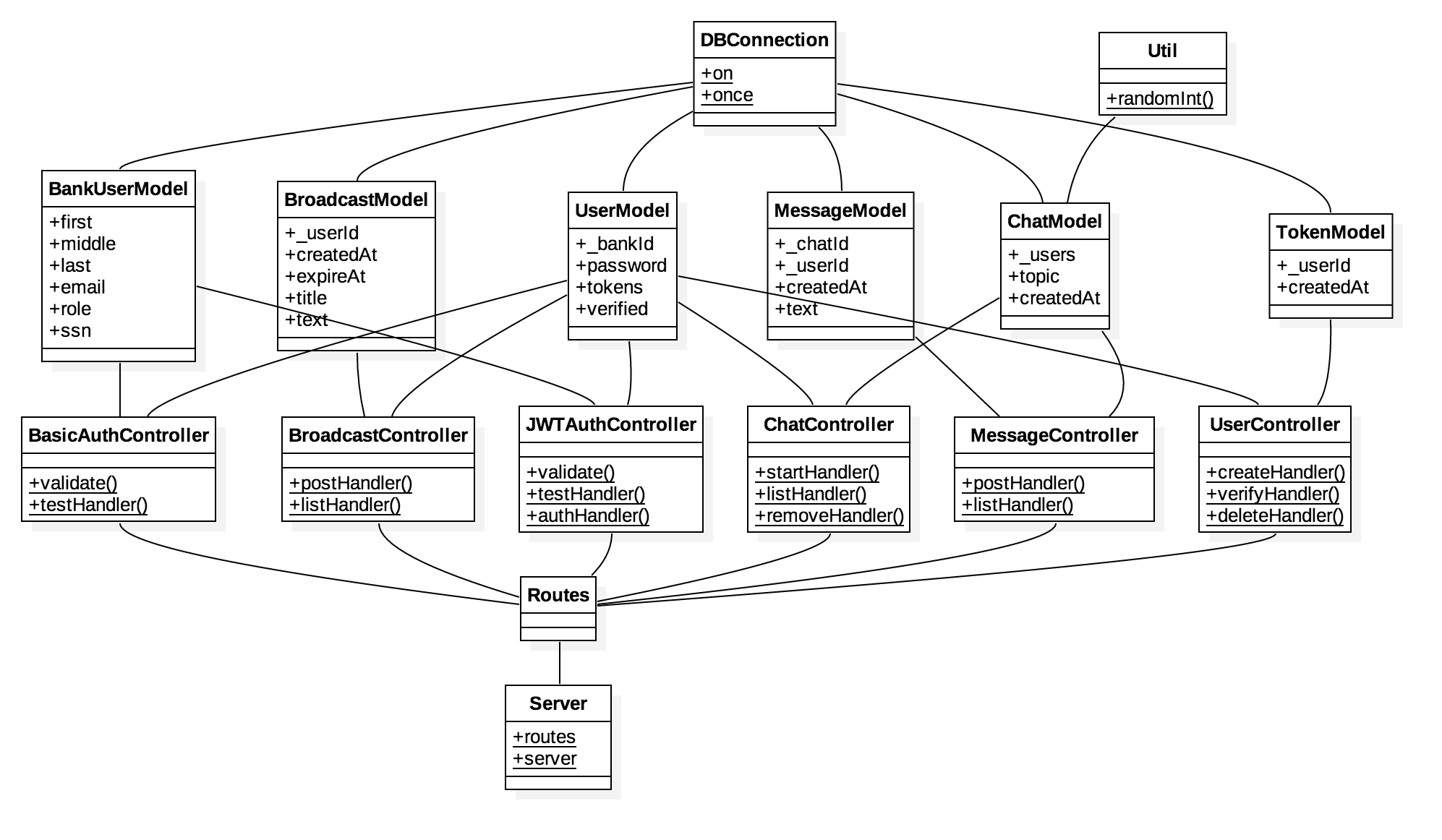
### Outline

The requested features for the service were that it: 1. Be a RESTful web service 2. Be implemented in Node 3. Must connect to a separate mock bank DB to restrict account creation to preexisting users in that DB 4. Have endpoints for basic account functionality, opening chats or tickets, and sending messages between users

### Results

All of the features in the outline were implemented by the end of the project and it was successfully used by one of the other group members working on a front-end app.

### UML Class Diagram



UML Class Diagram

## Project 2: Generic Messaging System

<https://github.com/shawnaten/node-message-api>

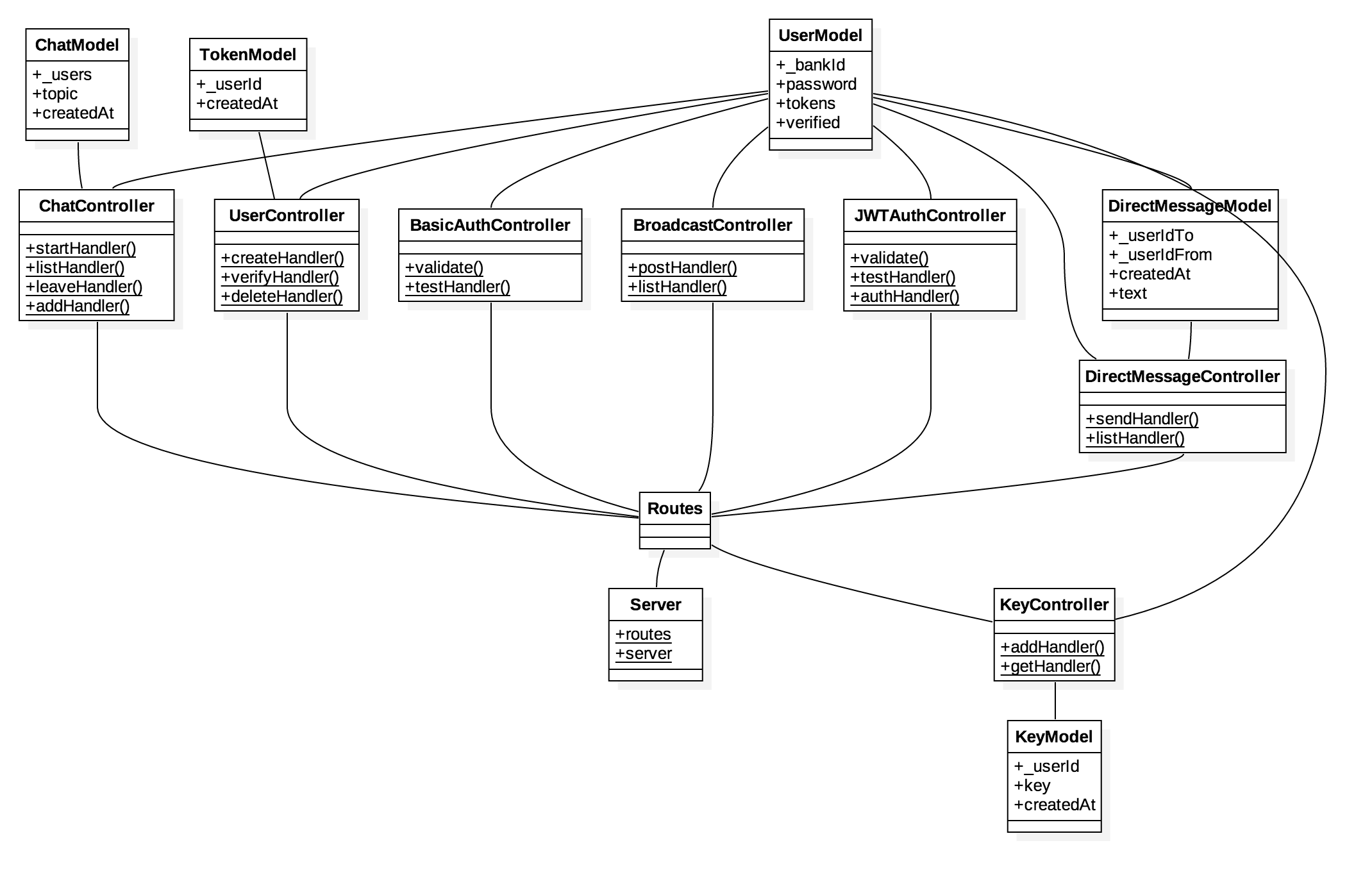
### Outline

The requested features for the service were that it: 1. Build upon the code from the first project 2. Remove the integration with the bank DB 3. Fully host account information in its own DB 4. Have endpoints for basic account functionality, group chat, and sending messages between users 5. Have some mechanism to support real-time messages

### Results

The group chat functionality was mostly implemented but abandoned because of changes to what the students in Dr. Niu's Software Engineering class were working on. Instead direct messages were implemented. A plan was developed for a real-time mechanism but not implemented by the end of the semester. The idea for the real-time functionality was to have an XMPP server that the Node server would send requests to. Android devices would be connected to the XMPP server and would trigger a notification when they received input from the XMPP server. So essentially the service worked as REST service with the addition of a push notification mechanism which could facilitate real-time message updates without polling.

### UML Class Diagram



UML Class Diagram