

**San Jose State University**  
**CMPE 202**  
**Summer 2018**  
**T3am Project Report**

Professor Paul Nguyen  
Professor Gopinath Vinodh

## Table of Contents

<b>General Information</b>	<b>6</b>
Team Members	6
Demo URL / Accounts	6
Front End App	6
Back End / API App	6
Code Repository URLs	7
Back end (back end api / java server)	7
Set up Documentation	7
<b>Extra Credit Implementations</b>	<b>7</b>
Implement a "real" iOS or Android Mobile App calling the REST API	7
Implement Web "Front-End" Deployed to Heroku for Starbucks Payment Card management	7
Deploy API to AWS as Docker Containers in Amazon Containers	7
DockerFile	7
<b>Sprint Information</b>	<b>8</b>
Burndown Chart	9
Team Schedule / Task Allocation	9
Sprint Standup Journal	10
07/21/2018 to 07/27/2018	10
07/20/2018	11
07/19/2018	11
07/17/2018 to 07/18/2018	11
07/16/2018	11
07/15/2018	12
07/14/2018	12

Github Push Contributors Graph	12
<b>Technologies</b>	<b>14</b>
Back End	14
Apache Camel	14
Maven	14
Google Guice	14
JUnit/Python Unittest	14
iBATIS Data Mapping Framework	14
Front End (webapp)	14
jQuery, jQuery UI, bootstrap and moment.js	15
Deployment	15
<b>App Flow</b>	<b>15</b>
Sign up flow	15
Login flow	16
Dashboard	16
Cards List	17
Add New Card	18
Purchases List	19
Add New Purchase	20
<b>Cloud Infrastructure</b>	<b>21</b>
Back End Deployment	21
Docker on AWS ECS	22
Build Docker image	22
Push to ECS repository	22
Run from EC2	22
Run from ECS Fargate	23
Front End Heroku deployment	27
App Setup	27
Config Vars	28

Database Schema	29
sb_user	29
Schema	29
Results	29
sb_card	29
Schema	30
Results	30
sb_purchase	30
Schema	31
Results	31
<b>API Spec</b>	<b>32</b>
Sign up (Create User)	32
Curl Sample	32
Response	32
Sign in (Log in)	32
Curl Sample	32
Response	33
Get Cards Reload by UserId	33
Curl Sample	33
Response	33
Reload Card by UserId	33
Curl Sample	33
Response	34
Get Purchases by UserId	35
Curl Sample	35
Response	35
Add Purchase by UserId	35
Curl Sample	35
Response	35

<b>Class Design UML Diagrams</b>	<b>36</b>
<b>State Diagrams</b>	<b>37</b>
Sign In / Sign Up	37
Add Card	37
Add Purchase	38
<b>Unit Testings</b>	<b>38</b>
Java Unit Tests	38
Python End-To-End Tests	39

## General Information

### Team Members

Our team is called t3am and has the following members:

- Sy Le (006088940)
- Hyunwook Shin (012507417)
- Kevin Lai (008498282)
- Lin Cheng (012484459)

### Demo URL / Accounts

#### Front End App

Front End App can be accessed via

- <http://cmpe202-t3am-starbucks-webapp.herokuapp.com>

##### Demo Account

email: **syle@gmail.com**

password: **password**

### Back End / API App

Back end app can be accessed via

- <http://cmpe202-java-rest-api.herokuapp.com>
- <http://ec2-34-192-241-153.compute-1.amazonaws.com:8202>

## Code Repository URLs

Back end (back end api / java server)

<https://github.com/nguyensjsu/su18-202-t3am>

<https://github.com/nguyensjsu/su18-202-t3am/tree/master/web-app>

## Set up Documentation

Follow this doc to setup the environment locally.

<https://docs.google.com/document/d/1YKTlsn7VnxUjwjOXPiQigHKz-usWgbWOH0NESBkKrKk/edit?usp=sharing>

## Jenkins Server (CI/CD)

<https://ec2-18-222-125-85.us-east-2.compute.amazonaws.com/jenkins>

## Extra Credit Implementations

Our team chooses to implement the following things for extra credit

Implement a "real" iOS or Android Mobile App calling the REST API

Implement Web "Front-End" Deployed to Heroku for Starbucks Payment Card management

Refer to the heroku link above for front end app and the front end code

## Deploy API to AWS as Docker Containers in Amazon Containers

### DockerFile

```
FROM java
EXPOSE 8202
COPY . /opt/t3am
CMD nohup java -cp /opt/t3am/starbucks2-service-1.0.jar RestService &
```

More details about the docker file can be found here in our repo

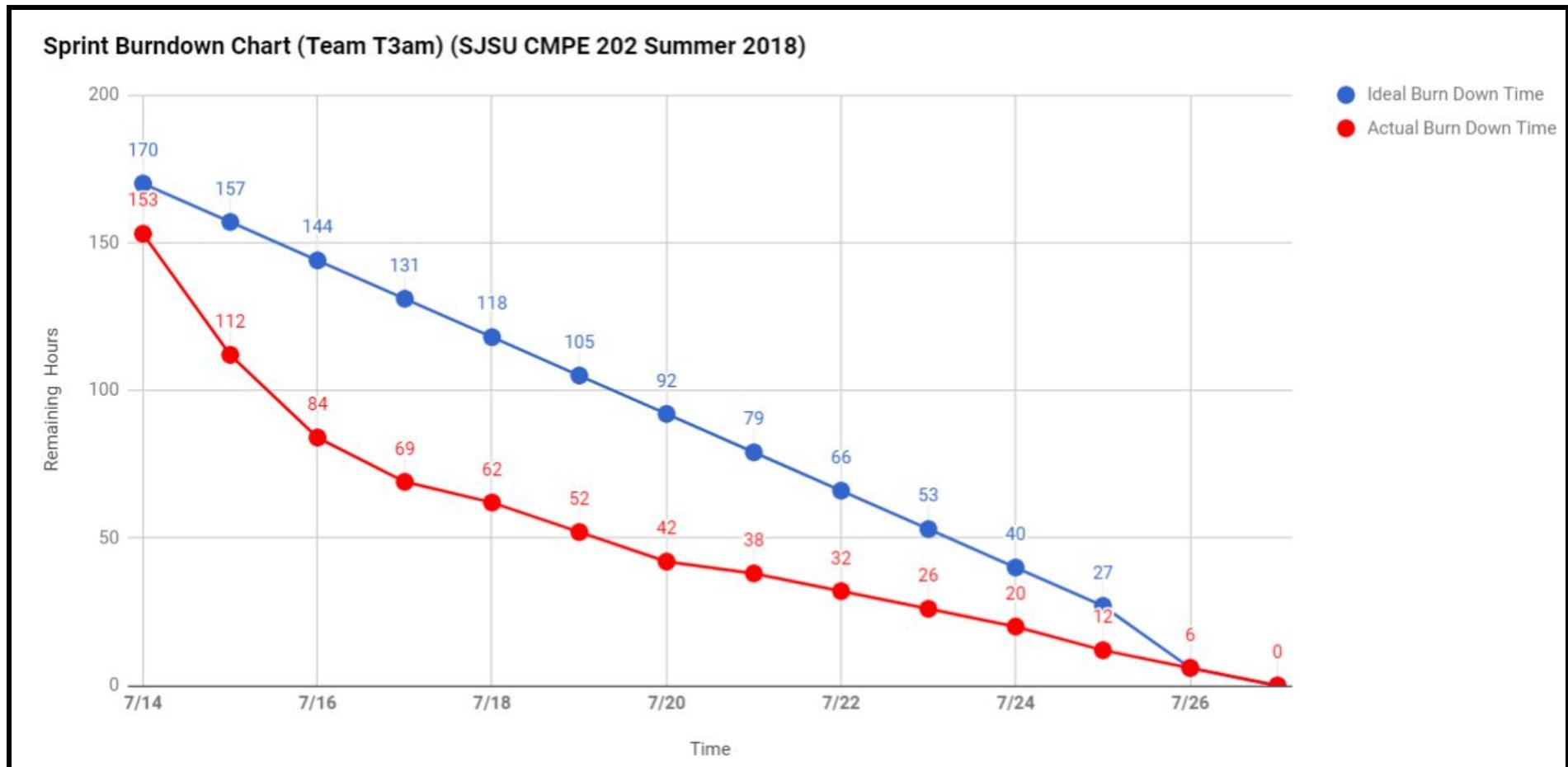
<https://github.com/nguyensjsu/su18-202-t3am/blob/master/Dockerfile>

## Sprint Information

Refer to this link for more information about our doc.

[https://docs.google.com/spreadsheets/d/1pfwGInHH9siD7P3Q7gQIALwi6-xfxB6CuSd0tBX\\_e3c/edit?usp=sharing](https://docs.google.com/spreadsheets/d/1pfwGInHH9siD7P3Q7gQIALwi6-xfxB6CuSd0tBX_e3c/edit?usp=sharing)

## Burndown Chart



## Team Schedule / Task Allocation

Task	Owner	ETA in Hour
Initial Planning work	Team	2

Set up RDS and DB tables	Lin Cheng	8
Set up the pipeline for webapp (AWS Beanstalk) includes the domain and basic hello world api.	Lin Cheng	2
Set up CI/CD pipeline for the above webapp/noSQL in the github repo...	Hyunwook	2
Initial code plumbing for the API server	Lin Cheng	12
Set up Response Header Content Type - Application/JSON	Sy Le	1
GET /api/v1/signin	Hyunwook	8
GET /api/v1/signout	Hyunwook	4
GET /api/v1/signup	Sy Le	10
GET /api/v1/user_profile return the information related to the user by user_id	Sy Le	4
GET /api/v1/transactions list all the transactions (card reloads vs purchase history) by user_id	Hyunwook & Kevin	8
POST /api/v1/reload add a new card to the list	Lin Cheng	6
POST /api/v1/purchase add a new purchase transaction to the list	Kevin Lai	8
Add Unit Tests	Hyunwook	4
Extra Credit Module 1 - Web Front End client side code development	Sy Le	15
Extra Credit Module 1 - Web Front End deployment on Heroku	Sy Le	10
MISC - Deploy BE code to Heroku	Sy Le	16
Docker image and deploy to AWS ECS	Lin Cheng	10
Do Project Report	TEAM	40
		170

## Sprint Standup Journal

07/21/2018 to 07/27/2018

- Team mainly works on project report

07/20/2018

- Done Front End and Deployed Front End App and API to Heroku
  - api : <http://cmpe202-java-rest-api.herokuapp.com>
  - webapp : <http://cmpe202-t3am-starbucks-webapp.herokuapp.com>
- Started and work on report

07/19/2018

- Complete last set of flow front end integration
- Start work on the deployment of front end heroku deployment
- Added Validation to API
- Wrap server response in ServerResponse
- Start work on project report
- Successfully run on AWS ECS Fargate

07/17/2018 to 07/18/2018

- Continue working on integrating Web FE in Add Purchase Flow
- Refactored and clean up code to make the Add Purchase API flow to work
- Clean up sql\_backup file
- Docker container for the API later
- WIP - Continued Add mode unit tests

07/16/2018

- Completed API to sign in using email and password
- Deployed our API Back End to heroku: <http://cmpe202-java-rest-api.herokuapp.com>
- Continued BE and FE (webapp)integration work: authentication (signin / signup) flow
- Parameterized Database Connection Strings, App Host, App Port into environment for deployment.
- Fixed UserProfile Fetch API to return real data instead of mocks

- WIP - CI/CD pipeline with jenkins
- WIP - Plumbing up unit tests

07/15/2018

- Initial Code complete, Server is now Up and Running
- Created 2 AWS EC instances for prod and dev
  - PROD: ec2-18-209-62-254.compute-1.amazonaws.com
  - DEV: ec2-34-192-241-153.compute-1.amazonaws.com
- Created AWS RDS for data store
- Functional getCards API
- Functional signUp API
- Basic plumbing works for POJO Objects such as Purchase, UserProfile, Card
- Created SQL Backup file for new deployment sql\_backup

07/14/2018

- Team meetup, draft out dependencies, task breakdown and initial work

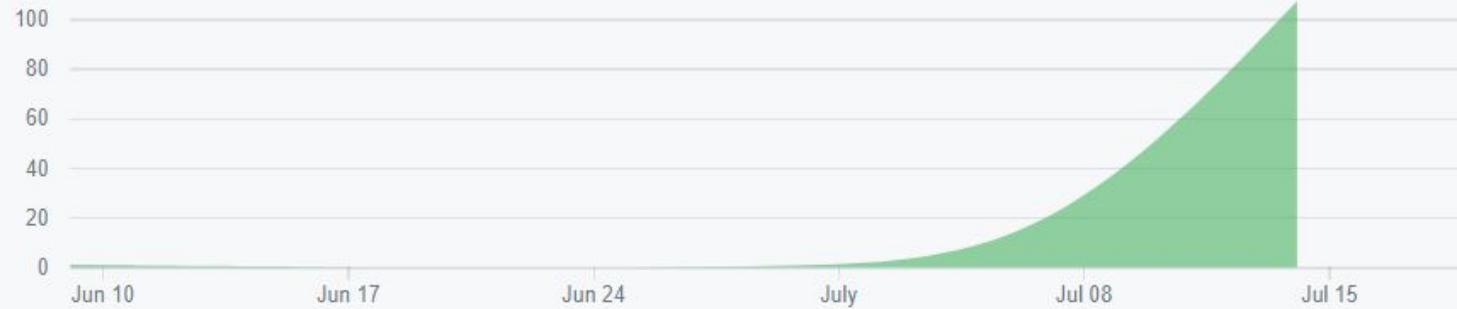
## Github Push Contributors Graph

<https://github.com/nguyensjsu/su18-202-t3am/graphs/contributors>

Jun 10, 2018 – Jul 20, 2018

Contributions: Commits ▾

Contributions to master, excluding merge commits



# Technologies

## Back End

### Apache Camel

The back end is built using Apache Camel which is used to serve up REST API endpoints.

### Maven

The back end project is bundled using maven.

### Google Guice

We used Guice to handle dependencies injection, namely @Inject

### iBATIS Data Mapping Framework

Internally we used iBATIS to map java code to SQL queries

## Front End (webapp)

The front end application in this project is a thin layer that handles integration with the back end API described above to handle basic flows of payments, purchases, reloads for drink ordering.

### jQuery, jQuery UI, bootstrap and moment.js

The front end is built using Bootstrap, jQuery, jQuery UI to construct the major piece of the UI and moment.js to handle date formatting

## Deployment

As for deployment, we used Heroku to deploy. You can find more information below.

## Testing and CI/CD

### JUnit/Python Unit Test

Junit library is used to run component-level unit tests in tools and models packages.

Python unittest library stress the end-to-end API testing on a local mock server.

### Jenkins

Jenkins job starts a test backend server locally by compiling the source code from GitHub using “make run”. As the Jenkins node does not have access to the git repository, we instead have created a recent copy of the source code (instead of git clone). After starting a service that listens to port 8082, Jenkins runs “make test” to run a series of API testing against the local server. This is ensure that there is no regressions or unexpected failures in our APIs. Once both JUnit and Python (end-to-end API) tests pass, the job will be marked as SUCCESS.

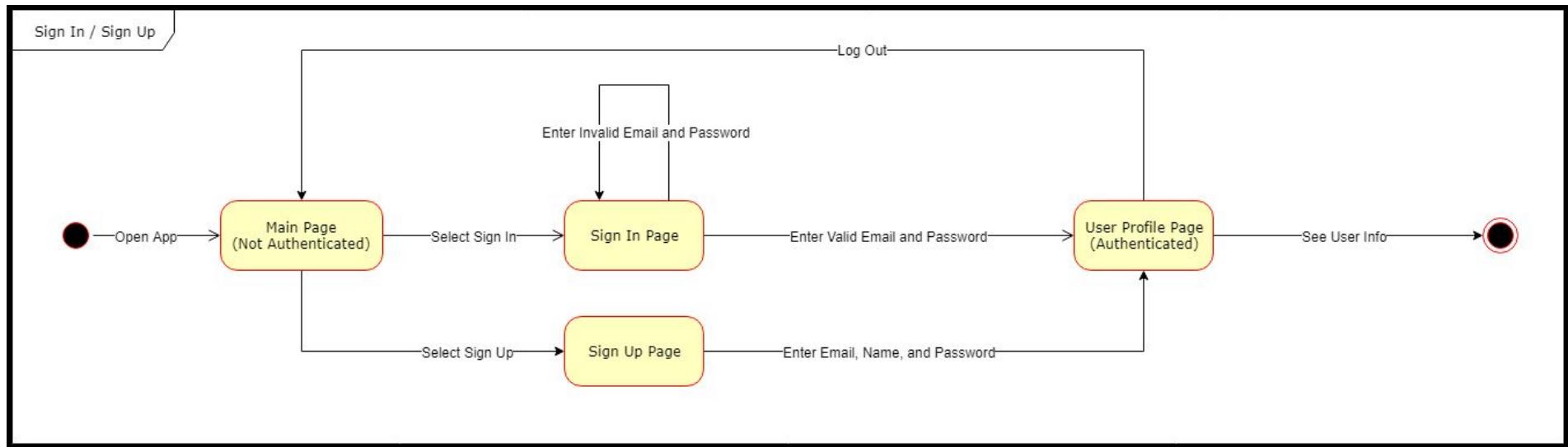
The screenshot shows the Jenkins web interface for a build. At the top, there's a navigation bar with the Jenkins logo, a search bar, and user information (Admin 202 | log out). Below the navigation, the page title is "Build #122 (Jul 20, 2018 11:34:15 PM)". To the left, a sidebar lists project management links: Back to Project, Status, Changes, Console Output, Edit Build Information, Delete Build, and Previous Build. The main content area displays the build status with a blue circular icon, the build number, date, and time. It also shows a note about "No changes." and the user who started it ("Started by user Admin 202"). At the bottom, a footer bar includes links for Page generated, REST API, and Jenkins version information.

Page generated: Jul 20, 2018 11:44:25 PM UTC REST API Jenkins ver. 2.121.1

## Ngnix

Nginx is used to route requests to Jenkins HTTPS server <https://ec2-18-222-125-85.us-east-2.compute.amazonaws.com:8081/login>

## App Flow



### Sign up flow

New users used this to sign up for new account

### Login flow

Existing users log in with their existing account

# CMPE202 T3AM Starbucks Store

Sign In    Sign Up

Email address

syle@gmail.com

Password

\*\*\*\*\*

Sign In

## Dashboard

This is where users check their remaining balance as well as some information about their account

# CMPE202 T3AM Starbucks Store

User	Reloads History	Purchases History
New Card	New Purchase	Log Out
<b>User ID</b> 958bbc20-cda2-4f57-9fe6-e1ecb6e6e913		
<b>Email address</b> syle@gmail.com		
<b>Full Name</b> Sy Le		
<b>Current Balance</b> \$36.00		

## Cards List

This is where user look for their previous reloaded gift cards

# CMPE202 T3AM Starbucks Store

User   Reloads History   Purchases History

New Card   Refresh

Number	Code	Balance	Date Added
1111111111	111	\$10.00	July 18th 2018, 3:20:57 pm
2222222222	222	\$20.00	July 18th 2018, 3:21:13 pm
3333333333	333	\$30.00	July 19th 2018, 2:41:04 pm

## Add New Card

This is where user add a new gift card

**Add New Card**

**User ID**  
958bbc20-cda2-4f57-9fe6-e1ecb6e6e913

**Email address**  
syle@gmail.com

**Card ID**  
4444444444

**Card Code**  
444

**Card Balance**  
40

**Save** **Cancel**

## Purchases List

This is where users can view their previous purchases

# CMPE202 T3AM Starbucks Store

User	Reloads History	Purchases History
<a href="#">New Purchase</a>	<a href="#">Refresh</a>	
Balance	Note	Date Added
\$3.00	Iced Coffee	July 18th 2018, 3:26:33 pm
\$3.00	Iced Coffee	July 18th 2018, 3:26:34 pm
\$3.00	Iced Coffee	July 18th 2018, 3:26:38 pm
\$4.00	Frap	July 18th 2018, 3:29:54 pm
\$5.00	Chai	July 18th 2018, 3:49:18 pm
\$6.00	Thai Tea	July 19th 2018, 2:41:31 pm

## Add New Purchase

This is where the user add a new purchase

**Add New Purchase**

**User ID**  
958bbc20-cda2-4f57-9fe6-e1ecb6e6e913

**Email address**  
syle@gmail.com

**Charge Amount**  
5

**Note**  
Cold Brew

**Save** **Cancel**

This is a screenshot of a modal dialog box titled "Add New Purchase". It contains four input fields: "User ID" with the value "958bbc20-cda2-4f57-9fe6-e1ecb6e6e913", "Email address" with the value "syle@gmail.com", "Charge Amount" with the value "5", and a "Note" field containing "Cold Brew". At the bottom right are two buttons: "Save" and "Cancel".

## Cloud Infrastructure

### Back End Deployment

Docker on AWS ECS

Build Docker image

- Check out code
- Go to source root(su18-202-t3am) and do a "mvn install"

- Run “java -cp starbucks2-service/target/starbucks2-service-1.0.jar RestService”
- Copy starbucks2-service/target/starbucks2-service-1.0.jar and starbucks2-service/target/lib/ to EC2 server’s /opt/t3am
- From /opt/t3am, “docker build -t t3am .”

## Push to ECS repository

- Login to AWS ECR: “aws ecr get-login --no-include-email”
- Create repository: “aws ecr create-repository --repository-name t3am”
- Tag image: “docker tag t3am 914381644891.dkr.ecr.us-east-1.amazonaws.com/t3am”
- Push: “docker push 914381644891.dkr.ecr.us-east-1.amazonaws.com/t3am”

```
[root@ip-172-31-3-105 t3am]# docker tag t3am 914381644891.dkr.ecr.us-east-1.amazonaws.com/t3am
[root@ip-172-31-3-105 t3am]# docker push 914381644891.dkr.ecr.us-east-1.amazonaws.com/t3am
The push refers to a repository [914381644891.dkr.ecr.us-east-1.amazonaws.com/t3am]
a76d79f03ef5: Pushed
35c20f26d188: Layer already exists
c3fe59dd9556: Layer already exists
6ed1a81ba5b6: Layer already exists
a3483ce177ce: Layer already exists
ce6c8756685b: Layer already exists
30339f20ced0: Layer already exists
0eb22bfb707d: Layer already exists
a2ae92ffcd29: Layer already exists
latest: digest: sha256:816288955be83bf80de4d9cb02703024897909d8a1e944ad65cb65c8b3160969 size: 2212
```

## Run from EC2

- Login to AWS ECR: “aws ecr get-login --no-include-email”
- Pull: “docker pull 914381644891.dkr.ecr.us-east-1.amazonaws.com/t3am”

```
[root@ip-172-31-1-174 ~]# docker pull 914381644891.dkr.ecr.us-east-1.amazonaws.com/t3am
Using default tag: latest
latest: Pulling from t3am
e12c678537ae: Already exists
8d9ed335b7db: Already exists
3318dd58ae60: Already exists
624ba6156166: Already exists
c7a02d193df7: Already exists
813b62320378: Already exists
81cf5426393a: Already exists
0a2b7222259b: Already exists
fffff8146f091: Pull complete
Digest: sha256:816288955be83bf80de4d9cb02703024897909d8a1e944ad65cb65c8b3160969
Status: Downloaded newer image for 914381644891.dkr.ecr.us-east-1.amazonaws.com/t3am:latest
```

- Tag image: “docker tag 914381644891.dkr.ecr.us-east-1.amazonaws.com/t3am t3am”
- Run: “docker run -p 8202:8202 t3am”

```
[root@ip-172-31-1-174 ~]# docker run -p 8202:8202 t3am
log4j:WARN No appenders could be found for logger (org.apache.ibatis.logging.LogFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
INFO [main] - Loaded 222 type converters
INFO [main] - Apache Camel 2.19.3 (CamelContext: camel-1) is starting
INFO [main] - JMX is enabled
INFO [main] - Runtime endpoint registry is in extended mode gathering usage statistics of all incoming and outgoing endpoints
INFO [main] - StreamCaching is not in use. If using streams then its recommended to enable stream caching. See more details at
INFO [main] - Route: route4 started and consuming from: direct://bizoption
INFO [main] - Route: route5 started and consuming from: direct://bizget
INFO [main] - Route: route6 started and consuming from: direct://bizpost
Jul 17, 2018 6:21:27 AM org.restlet.engine.connector.NetServerHelper start
INFO: Starting the internal [HTTP/1.1] server on port 8202
INFO [main] - Route: routel started and consuming from: http://localhost:8202/api/v1/(path)?restletMethods=OPTIONS
INFO [main] - Route: route2 started and consuming from: http://localhost:8202/api/v1/(path)?restletMethods=GET
INFO [main] - Route: route3 started and consuming from: http://localhost:8202/api/v1/(path)?restletMethods=POST
INFO [main] - Total 6 routes, of which 6 are started.
INFO [main] - Apache Camel 2.19.3 (CamelContext: camel-1) started in 0.925 seconds
```

Run from ECS Fargate

- Define a Fargate Task using the docker image

Container Name	Image
t3am-c	914381644891.dkr.ecr.us-east-1.amazonaws.com/t3am

---

Details

---

Port Mappings

---

Host Port	Container Port	Protocol
8202	8202	tcp

---

Healthcheck

---

Command ping localhost

---

Interval	Timeout	Start period	Retries
100	3	100	3

- Create a cluster and run the task

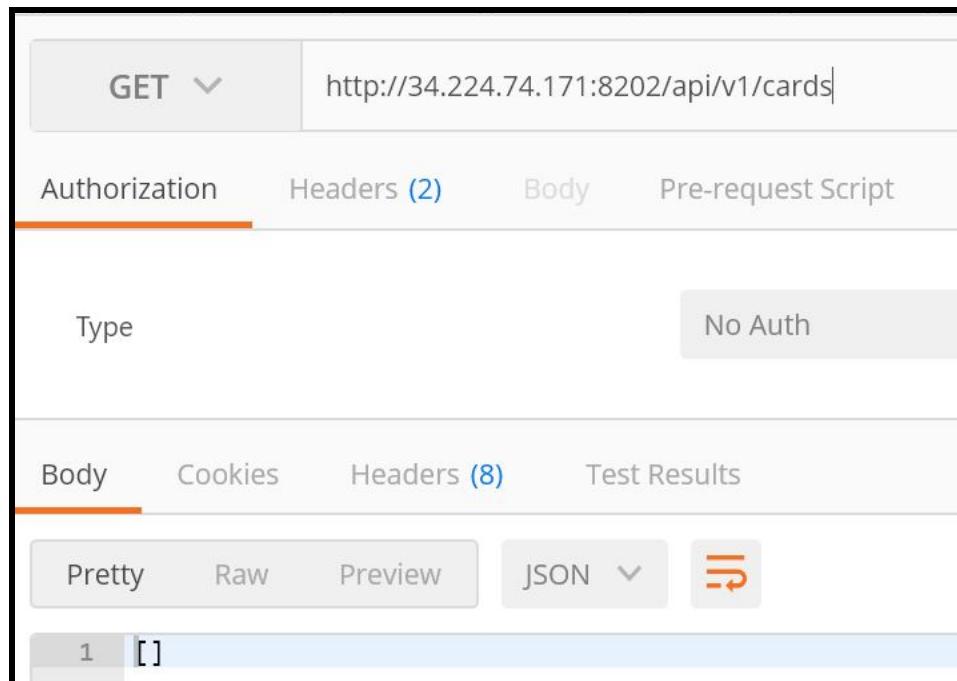
## Cluster : t3am

Get a detailed view of the resources on your cluster.

Status	ACTIVE			
Registered container instances	0			
Pending tasks count	0 Fargate, 0 EC2			
Running tasks count	1 Fargate, 0 EC2			
Active service count	1 Fargate, 0 EC2			
Draining service count	0 Fargate, 0 EC2			
<hr/>				
<a href="#">Services</a>	<a href="#">Tasks</a>	<a href="#">ECS Instances</a>	<a href="#">Metrics</a>	<a href="#">Scheduled Tasks</a>
<hr/>				
<a href="#">Create</a>	<a href="#">Update</a>	<a href="#">Delete</a>		
<hr/>				
<input type="text"/> Filter in this page	Launch type ALL	Service type ALL		
<input type="checkbox"/> Service Name	Status	Service ty...	Task Defin...	
<input type="checkbox"/> t3am-s	ACTIVE	REPLIC	t3am-t:1	

- Get the created instance IP and try visiting

Network interface	Subnet ID	VPC ID	Zone	Security groups	Description	Instance ID	Status	IPv4 Public IP
eni-05e60662	subnet-1ebe6a...	vpc-541c5533	us-east-1d	t3am-s-1585	arn:aws:ecs:us...	<span>●</span> in-use		34.224.74.171



## Front End Heroku deployment

### App Setup

The screenshot shows the Heroku application dashboard for the app `cmpe202-t3am-starbucks-webapp`. The top navigation bar includes the Heroku logo, a search bar, and user account icons. Below the header, the app's name is displayed along with its GitHub integration information: `syngle/cmpe202-t3am-starbucks-webapp` and the `master` branch. A navigation menu at the top provides links to Overview, Resources, Deploy, Metrics, Activity, Access, and Settings. The main content area starts with a "Free Dynos" section featuring an "Upgrade to Hobby..." button. A promotional message about Web Dyno Autoscaling is visible, with "Hide" and "Learn More" buttons. At the bottom, a configuration section for the `web` dyno shows the command `node index.js`, a toggle switch, a price of `$0.00`, and a edit icon.

 HEROKU   

Info	Region	 United States
	Stack	heroku-16
	Framework	 Node.js
	Slug Size	17.3 MiB of 500 MiB
	GitHub Repo	 <a href="#">synle/cmpe202-t3am-starbucks-webapp</a>
	Heroku Git URL	<a href="https://git.heroku.com/cmpe202-t3am-starbucks-webapp.git">https://git.heroku.com/cmpe202-t3am-starbucks-webapp.git</a>

## Config Vars

API\_HOST is set in the config file which the front end app is used to communicate with the back end API

**Config Vars**   

Config vars change the way your app behaves. In addition to creating your own, some add-ons come with their own.

<input type="text" value="API_HOST"/>	<input type="text" value="https://cmpe202-jav..."/>
<input type="text" value="KEY"/>	<input type="text" value="VALUE"/>
<input type="button" value="Add"/>	

## Database Schema

sb\_user

This is where we stores users' information

Schema

▼	public	sb_user
▼ Columns		
□	user_id	CHARACTER VARYING
□	email	CHARACTER VARYING
□	full_name	CHARACTER VARYING
□	password	CHARACTER VARYING
□	balance	NUMERIC
□	date_added	BIGINT

Results

Rows 3						CSV	JSON	CSV	JSON
user_id	email	full_name	password	balance	date_added				
958bbc20-cda2-4...	syle@gmail.com	Sy Le	password	0	1531952274019				
feab2bad-d5dd-4..	syle1@gmail.com	Sy Le	password	0	1531959952188				
e4c226eb-2168-4...	syle2@gmail.com	Sy Le2	password	0	1532036670245				

sb\_card

This is where we stores previous reloaded card. This has a user\_id foreign key that relates records to users

## Schema

public.sb_card
Columns
card_id BIGINT
uid CHARACTER VARYING
number CHARACTER VARYING
code CHARACTER VARYING
balance NUMERIC
date_added BIGINT

## Results

Rows 5	CSV	JSON	Rows 5	CSV	JSON
card_id	uid	number	code	balance	date_added
4	958bbc20-cda2-...	1111111111	111	10	1531952457886
5	958bbc20-cda2-...	2222222222	222	20	1531952473593
6	feab2bad-d5dd-4...	123456789	123	100	0
7	958bbc20-cda2-...	333333333	333	30	1532036464962
8	958bbc20-cda2-...	444444444	444	40	1532049960445

## sb\_purchase

This is where we stores previous purchases. This has a user\_id foreign key that relates records to users

## Schema

▼	public.sb_purchase
▼ Columns	
□	purchase_id BIGINT
□	uid CHARACTER VARYING
□	balance NUMERIC
□	date_added BIGINT
□	note CHARACTER VARYING

## Results

Rows 7	CSV	JSON	Rows 7	CSV	JSON
purchase_id	uid	balance	date_added	note	
1	958bbc20-cda2-...	3	1531952793526	Iced Coffee	
2	958bbc20-cda2-...	3	1531952794984	Iced Coffee	
3	958bbc20-cda2-...	3	1531952798744	Iced Coffee	
4	958bbc20-cda2-...	4	1531952994579	Frap	
5	958bbc20-cda2-...	5	1531954158600	Chai	
6	958bbc20-cda2-...	6	1532036491432	Thai Tea	
7	feab2bad-d5dd-4...	30	1532037687755	Meal Combo #1	

# API Spec

## Sign up (Create User)

This API is used for NEW users to sign up for new account.

### Curl Sample

```
curl -X POST \
  http://localhost:8202/api/v1/signup \
  -H 'accept: application/json' \
  -d '{
    "email": "style1@gmail.com",
    "full_name": "Sy Le",
    "password": "password"
}'
```

### Response

<https://github.com/nguyensjsu/su18-202-t3am/blob/master/starbucks2-domain/src/main/java/model/UserProfile.java>

## Sign in (Log in)

This API is used for EXISTING users to sign in with their old account.

### Curl Sample

```
curl -X POST \
  http://localhost:8202/api/v1/signin \
  -H 'accept: application/json' \
  -d '{'
```

```
"email": "style1@gmail.com",
"password": "password"
}'
```

## Response

<https://github.com/nguyensjsu/su18-202-t3am/blob/master/starbucks2-domain/src/main/java/model/UserProfile.java>

## Get Cards Reload by UserId

This API is used to get a list of cards reloaded by userId

### Curl Sample

```
curl -X GET \
'http://localhost:8202/api/v1/cards?uid=2c60158e-d432-4b78-a300-360cc6fa7260'
```

## Response

<https://github.com/nguyensjsu/su18-202-t3am/blob/master/starbucks2-domain/src/main/java/model/Card.java>

## Reload Card by UserId

This API is used to add/reload a card for a user by UserId

### Curl Sample

```
curl 'http://localhost:8080/api/v1/reload' \8081/no-referrer' \
-H 'Connection: keep-alive' \
-H 'DNT: 1' --data-binary
'{"uid":"958bbc20-cda2-4f57-9fe6-e1ecb6e6e913","number":"1111111111","code":"111","balance":"10"}'
```

## Response

<https://github.com/nguyensjsu/su18-202-t3am/blob/master/starbucks2-domain/src/main/java/model/Card.java>

## Get Purchases by UserId

This API is used to get a list of previous purchases by UserID

### Curl Sample

```
curl -X GET \
  'http://localhost:8202/api/v1/purchases?uid=2c60158e-d432-4b78-a300-360cc6fa7260'
```

## Response

<https://github.com/nguyensjsu/su18-202-t3am/blob/master/starbucks2-domain/src/main/java/model/Purchase.java>

## Add Purchase by UserId

This API is used to add a purchase by UserID

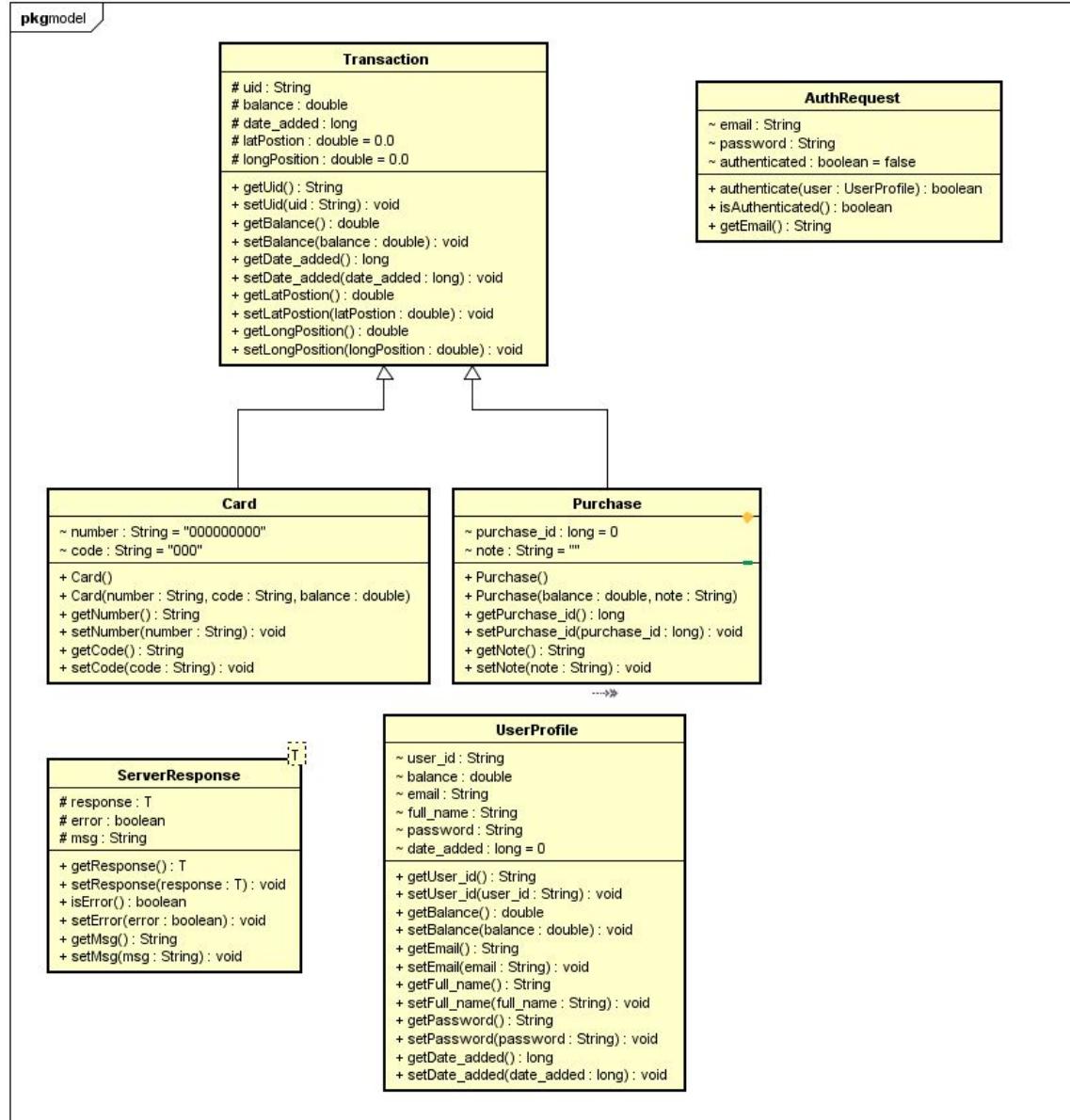
### Curl Sample

```
curl 'http://localhost:8080/api/v1/purchase' \
  -H 'Content-Type: application/json; charset=utf-8' \
  -H 'DNT: 1' --data-binary '{"uid":"958bbc20-cda2-4f57-9fe6-e1ecb6e6e913", "balance":"3", "note":"Iced Coffee"}'
```

## Response

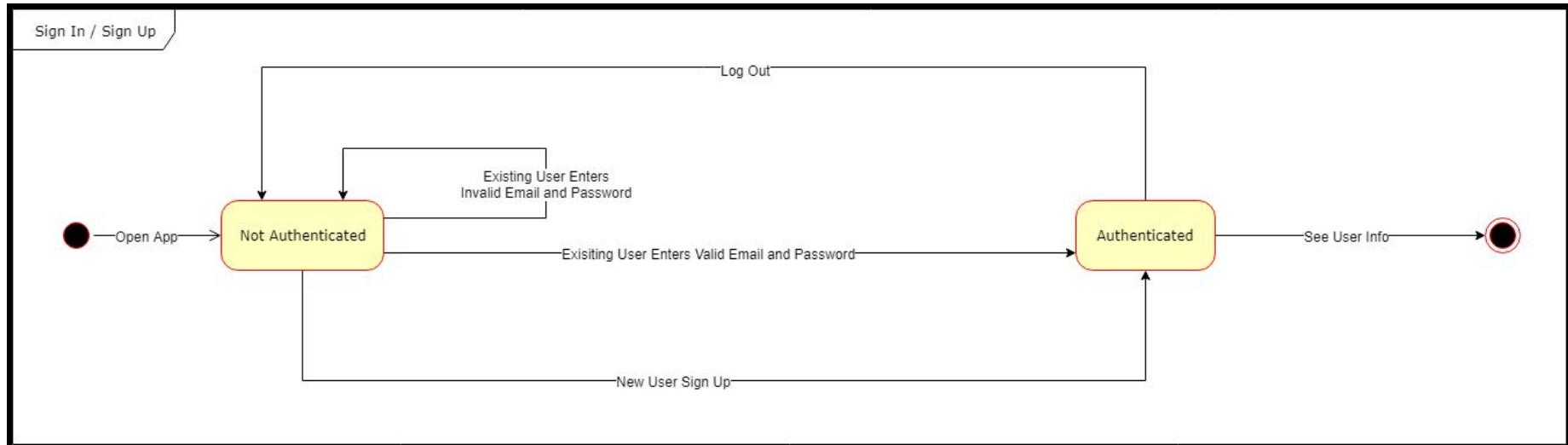
<https://github.com/nguyensjsu/su18-202-t3am/blob/master/starbucks2-domain/src/main/java/model/Purchase.java>

# Class Design UML Diagrams

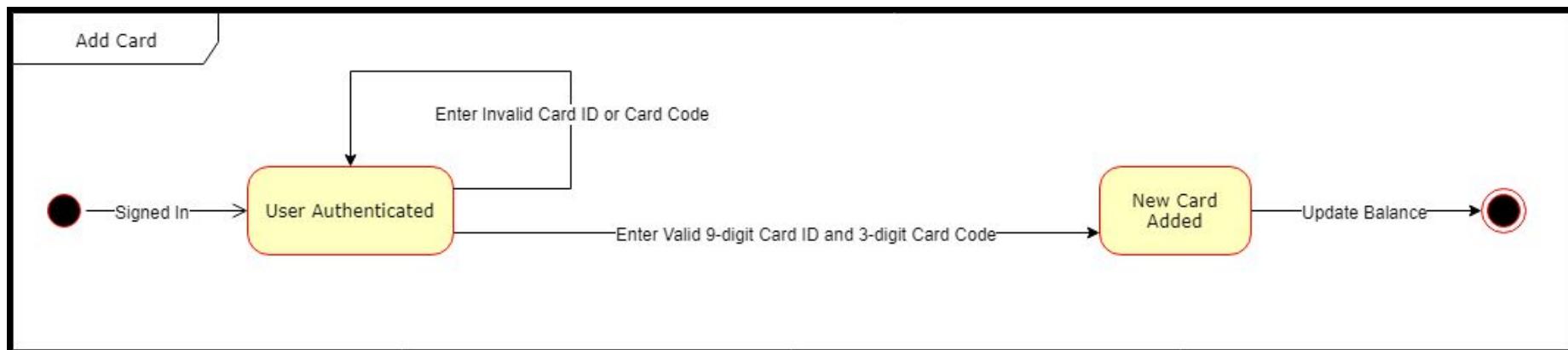


## State Diagrams

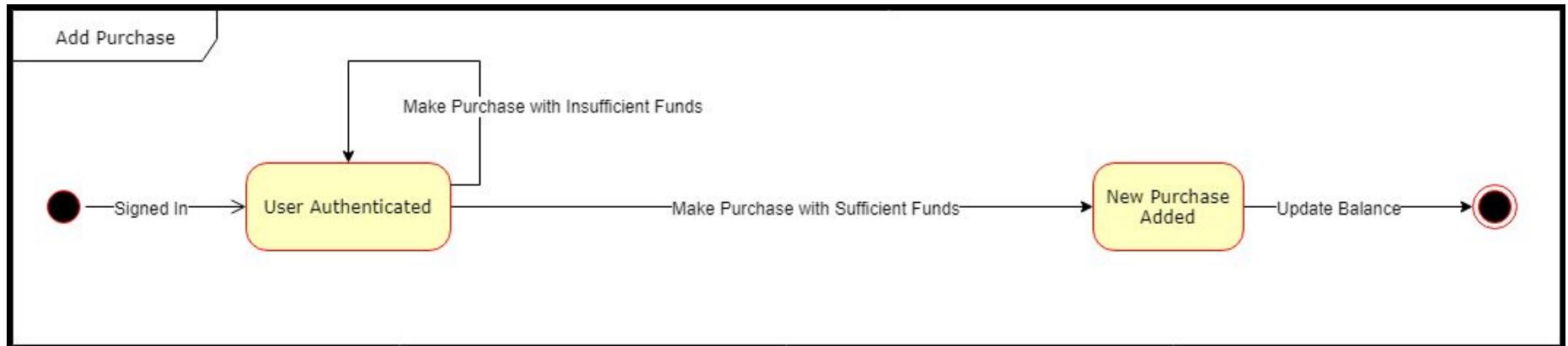
### Sign In / Sign Up



### Add Card



## Add Purchase



## Unit Testings

### Java Unit Tests

Helper package JUnit

```
-----  
T E S T S  
-----  
Running helper.JSONHelperTest  
Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.246 sec  
Results :  
Tests run: 3, Failures: 0, Errors: 0, Skipped: 0  
[INFO]
```

Model Package JUnit

```
-----  
T E S T S  
-----  
Results :  
Tests run: 0, Failures: 0, Errors: 0, skipped: 0
```

## Python End-To-End Tests

Workflow:

1. Run “mvn build” to build the JAR
2. Run the JAR, and start the backend server listening to localhost:8202
3. Run python test “api\_tests” which send REST API requests to the localhost server
4. Verify that responses and fields match the expected results

```
python -m unittest api_tests  
...  
-----  
Ran 3 tests in 0.249s  
  
OK  
dock2 ~/wses/su18-202-t3am/jenkins/tests$ █
```

Each Jenkins Job runs the python end-to-end tests. A screenshot of a successful run is shown below:

```
python-requests/2.6.0 CPython/2.7.5 Linux/3.10.0-862.3.2.el7.x86_64 -  
.  
-----  
Ran 3 tests in 0.635s  
  
OK  
make[2]: Leaving directory `/var/lib/jenkins/workspace/API Testing/su18-202-t3am/jenkins/tests'  
make[1]: Leaving directory `/var/lib/jenkins/workspace/API Testing/su18-202-t3am/jenkins'  
+ set +x  
sed -i "s/value=\"_your_db_password\"/value=\"****\"/g" starbucks2-  
domain/src/main/java/sql-maps-config.xml  
++ ps -ef  
++ grep 'java.*jar'  
++ grep -v grep  
++ grep Rest  
++ awk '{print $2}'  
+ kill -9 3706  
make: *** [run] Killed  
Finished: SUCCESS
```