**The Transition Document for the Web Application Team**

# About the design of the web application

The web application is a subsystem of the project. The major goal is that creates a webservice to allow users to access their account though a browser. In May, we had an initial design of the web application and started to build our prototype. In July, after Devi showed his demo of the ML, we rebuild the application since the database and the design changed.

## The Initial design

### Use case

At the beginning, we decided to add a webservice for the company employees, a default internal user “Admin” can create an account for other administrators and sales representatives. The different type user can access the different services. Our web application includes two parts, one is the “customer” part, the other is the “admin” part.

A close up of text on a whiteboard

Description automatically generated

You also can use this link to access the [use case diagram](https://app.lucidchart.com/invitations/accept/555f8d90-c587-444c-ac60-b24c622ccb27).

### Site map

The website for the customers

A screenshot of a cell phone

Description automatically generated

The website for the company employees, we also called admin part of the web app.

A close up of a map

Description automatically generated

You also can use this link to access the [sitemap](https://app.lucidchart.com/invitations/accept/f8357bd7-01ee-458a-a7bd-b049eeb5946c).

### The relationship with the other subsystem

My opinion is that three subsystems are independent. Once the database schema is decided, each team can work on the subsystem without waiting for other teams. Later on, we changed the architecture design and the database we use.

A close up of text on a white background

Description automatically generated

## The changes of the design.

We changed the design around July 5th since Devi believed that if our endpoints register a user into the database will lead the issue in data analysis subsystem.

**The major change is about the architecture design**. Devi wants one API to provide three services for the web app, the mobile app, and data analysis website. But the problem is Only one flask-login instant can be used in a flask project. We decided that our endpoints to call the API to access the database to avoid the problems(which the left picture shows). Devi created user login, log out and registration in the API, we need to create the other endpoints we want at the API and call them from our web service. Later on, we found that we can use flask-session to replace the flask-login. **Our web service became a blueprint of the API now (the right picture shows)**. You can use this link to access the [ads](https://app.lucidchart.com/invitations/accept/dd416085-e103-472e-aed7-e4131c6ab83a).

A close up of text on a white background

Description automatically generated A close up of a map

Description automatically generated

Other changes include that we **changed the database from Mysql to mongoDB** and **abandoned Admin part of the web application** since Devi’s website have same idea. We only need to focus on the customers website.

# About the implementation of the web application

We choice **python and flask framework** to create our web application. We create the models in models.py to interact with the database, build the endpoints in routes.py as the controller, and construct templates as the views.

## Admin web app, the RESTful API and the demo

**If you are familiar with python and web application development, you may skip this section and start from section 2.2.** Before the changes of the design, I created admin web application and a RESTful API. Edwin Also made a demo by integrated his customer registration page into my admin part of the web application, another approach for one webservice provide login support for two web applications.

### Admin web app and the Restful API

The purpose of building the RESTful API is that I tried to help the mobile application team to use a MySQL database to store the user registration info and user authentication instead of using AWS Congito. This small project allows a mobile application to register a user, request a token, and revoke a token. It also helps you to understand the differences between the backend of a web application and a mobile application.

The RESTful API is a blueprint of the admin web service.

Download it: <https://github.com/shiweiwei168/Web-admin>

**Install and run the service:**

apt-get install python3-venv

python3 -m venv flask //Create a venv called flask.

source flask/bin/activate //activate the venv

pip install -r requirements.txt

run "FLASK\_APP=admin.py"

run "flask run"

**Test the admin website:**

Use your browser to access 127.0.0.1:5000

The system has a default admin account {“admin”,”admin”}.

**Test the RESTful API**:

I include a readme.docx under the app/api folder.

<https://github.com/shiweiwei168/Web-admin/blob/master/app/api/Readme.docx>

**The database schema**

**A screenshot of a cell phone

Description automatically generated**

You can access the file by this link: [the schema](https://app.lucidchart.com/invitations/accept/dba6dad9-8a5b-4a6b-a245-9e9dac147ffd)

You also can review the models.py in the project.

**Major Flask modules**

Flask-SQLalchemy //SQL database support

flask-migrate. //help you to migrate your database easily

werkzeug. // password hash

Flask-login //user authenticate

Blueprint. //The RESTful API is a blueprint of the admin web app.

Base64. // generate the token.

**Switch the database**

one advantage of using a SQL database is that you can use SQLite when you develop the application on your computer. You can switch the database to MySQL database when you deploy the application to AWS.

I include a document about the switching the database in the project. <https://github.com/shiweiwei168/Web-admin/blob/master/Switch%20database.txt>

**Deploy your web application on the AWS**

<https://github.com/shiweiwei168/Web-admin/blob/master/deploy%20to%20AWS.txt>

### The Demo

The original demonstration of the website had a very different design then the current version. Initially we had one login page that both the users and admins would use to log in. The login request would check if the account was in the database as an admin or not, then redirect you to their corresponding pages. After more thought, we decided it was safer to separate the admins and users into different pages and databases. Next we migrated from an SQL database to using mongoDB as it was easier to implement with AWS and the mobile team.

## Current version of the web application

Current version of the web application is a blueprint of the API. Our goal focus on build a customer website.

A screenshot of a cell phone

Description automatically generated

\*My Dashboard will show user today’s detailed data, include steps, calories burned, heart rate, and activity rating.

\*My profile will show user’s profile and allow user to upload a picture as a user’s avatar, edit their profile and change the password.

### Install and run the service:

Devi provide a document and two videos which help you install and run the services (include our service).

Here is a link for the document. <https://github.com/statefarmuta/api_ml_admin/blob/master/README.md>

After install the service on your computer, you should check if the **collection “user\_stats”** exist. If the collection doesn’t exist, you need to create one. [database.txt](https://github.com/statefarmuta/api_ml_admin/blob/webapp/app/web/doc/database.txt)

you can test our web service at <http://0.0.0.0:5000/web/>.

After you register the first user on the website, you may notice that the profile picture is empty. In order to avoid this situation, you should upload a picture “default” in the database. You can login a user first, access the myprofile page, upload the picture “default” under “\app\web\doc” folder to the system. This picture will become a default picture for all users.

### Database schema

A screenshot of a cell phone

Description automatically generated

You can access the file by this link: [the schema](https://app.lucidchart.com/invitations/accept/9312cec8-f42d-4525-8ce2-eeafa0d42615)

\*user\_stats table is still unclear at the end of our project. You should read Devi’s Document and talk with your teammate to make sure the attributes of the table.

### Application Structure

As I mentioned before, our web service is a blueprint of the API. The Blueprint under app/web/

A screen shot of a computer

Description automatically generated

We added the codes in the API’s \_\_init\_\_.py to register blueprint, Pymongo, and Session.

A screenshot of a cell phone

Description automatically generated



Three of our endpoints, user registration, login, and log out call the API’s endpoints to get job done to avoid causing the issue for data analysis subsystem. In the API’s user registration, Devi also provides the email verification support. You may read his documentation to learn his approach.

A screenshot of a cell phone

Description automatically generated



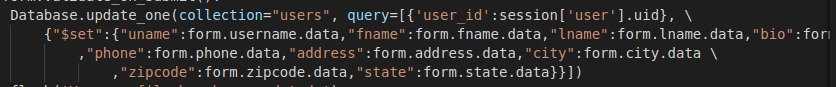
Bhupendra created the endpoints in the API and calls those endpoints from our endpoint to complete password reset. He has a section (2.2.9) to talk about it. The rest endpoints of the web service can directly access the database to complete the task.

### Database access

We tried two approach to access the database.

The first one is Devi’s approach. He uses it in the API.





The second one is Pymongo

A close up of a sign

Description automatically generated



A screenshot of a cell phone

Description automatically generated

If you upload a file to the database, you need to use pymongo.

### Session

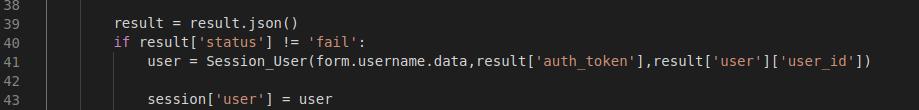
We used Session-based user authentication instead of flask\_login. The server stores session info when a user login.

In the application, we have a sessionuser model under app\web\models folder.

A screenshot of a cell phone

Description automatically generated

When a user login, create a session user and store the info in the server.



We can check the session user to protect the endpoint.

A screen shot of a computer

Description automatically generated

### Dashboard

The dashboard presents today’s data to the user. If the data didn’t exist, the endpoint will return all 0 Dict.

A screenshot of a cell phone

Description automatically generated

### Profile\_pic

The tricky part of the profile related endpoints is profile\_pic upload.

We use GrifFS of mongo DB to store the profile picture files. [The details](https://docs.mongodb.com/manual/core/gridfs/)

Although the mongoDB doesn’t prohibit user upload the files that have same filename, I decided that the filename should be unique. Therefore, if the filename has been used, I will ask user to choose another one. Once user upload a new picture, I will delete the old one from the database except “default”, a default user profile\_pic. (don’t forget to upload a picture “default” to the database before you run the service.)

In the template “myprofile.html”:

A close up of a screen

Description automatically generated

Endpoint:

A screenshot of a cell phone

Description automatically generated

If you don’t like this design, you can change it. For example, store a file’s “\_id” in the attribute profile\_pic instead of “filename”.

### UI ()

The web application utilizes flask, jinja2 templates, HTML, CSS, as well as chart.js and a jquery plugin to display the data to the user.

Within the templates directory you will find all the HTML files:

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

The base.html file contains the styling for the navbar which all other pages have in common. All other HTML files inherit from the base.

One directory up, in the app directory, there is a static directory. The static directory is where flask looks for images, css, and js files.

A screenshot of a cell phone

Description automatically generated

All the images, css, and js files used in the web application can be found in the static directory.

A screenshot of a cell phone

Description automatically generated

For the line graph and bar chart on the dashboard page chart.js was used. For the circular progress bars a jquery plugin was used, which can be found here: <https://github.com/rendro/easy-pie-chart>.

Here are some youtube videos I found helpful when coding the front end:

* Animated Circular Progress Bar Using Easy Pie Chart Plugin - Create a Progress Bar With Javascript
  + <https://www.youtube.com/watch?v=BOgc1KRYk30>
* Customized Animated Circular Progress Bar - Part 2 - Easy Pie Chart.js Simple jQuery Plugin Tutorial
  + <https://www.youtube.com/watch?v=rOtNHrEDBbM&t=170s>
* Beautiful Charts with JavaScript - Chart.JS Tutorial
  + <https://www.youtube.com/watch?v=f-7uQXGur2o&t=217s>
* Flask Forms and Styling
  + <https://www.youtube.com/watch?v=6LvmaAtuwfU&t=1686s>

### Password reset

This section will explain about the password reset for web application (Statefarm Fitbit). Password reset is most important in any application. This part will explain the working of password reset features for those who forget their password.

Web/templates/login.html

This html file has a link for password reset. When you click this link it will redirect you to reset\_request where you should provide email.

Web/templates/reset\_reuest.html

This is html file where you can fill up the related email when you want to change the password or forgot the password.

A screenshot of a computer screen

Description automatically generated

Web/forms.py

This is the corresponding form for request\_reset.html that allows you to enter your email field. When you enter a valid email and click on submit button, it will trigger the endpoint reset\_password\_request.

A screenshot of a cell phone

Description automatically generated

Web/routes.py

A screenshot of a cell phone

Description automatically generated

This will provide a route that will link to the API endpoint. Here we are running locally so the payload will be post to that endpoint i.e. 'http://0.0.0.0:5000/auth/reset\_password\_request'. If you are trying to use aws, you need to have link for ec2 which is something like <http://ec2-34-212-133-20.us-west-2.compute.amazonaws.com/auth/request_password_request>' This will redirect into that endpoint which is at view.py

A screenshot of a cell phone

Description automatically generated

Here if email does not exist in the database, it will provide some error. If exist, it will send a reset link to that email. But the token expires at 600 seconds. For this

token = user.get\_reset\_token() will use user\_id for token expiry.

A screenshot of a cell phone

Description automatically generated

After user receive mail and click on the reset token link. It will be trigger towards another end point i.e. reset\_token. Here first user\_id is verified with verify\_reset\_token which is at app/models/users.py

A screenshot of a cell phone

Description automatically generated

If the token is invalid, it will redirect again to reset\_request.html to enter email again and start process. If token is valid, the endpoint direct you to the form where you can enter the new password which is app/templates/page/reset\_token.html. The corresponding .html and forms are shown below:

A screenshot of a cell phone

Description automatically generated

A screenshot of text

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Here in this end point the database will be updated and sends you with the message of successful password reset and redirect to the login page. Now you can enter your username and new password to login into your account.

# Contact Information

“May, Chelsea” [chelsea.may@mavs.uta.edu](mailto:chelsea.may@mavs.uta.edu) If you have questions about the UI section.

"Ramdam, Bhupendra" [bhupendra.ramdam@mavs.uta.edu](mailto:bhupendra.ramdam@mavs.uta.edu) if you have questions about password reset section.

"Popaja, Edwin E" [edwin.popaja@mavs.uta.edu](mailto:edwin.popaja@mavs.uta.edu) if you have questions about the Demo section.

"Shi, Wei" [wei.shi2@mavs.uta.edu](mailto:wei.shi2@mavs.uta.edu) if you have questions about all other sections.