GITHUBNIZE

Giithubnize = Github + Organize

Instructor: Dr. Suzan Üsküdarlı

Mustafa Gönül - 2016719060

Github:
https://github.com/mustafagonul/Fall2017Swe573

A. ABOUT GITHUBNIZE

Githubnize = Github + Organize

Semester Plan

Githubnize is the application used to organize the repositories starred. With the tags assigned to the repositories starred, you can focus the domain and be structural managing the the projects you mostly need, follow and use.

Future Plan

The aim of Githubnize application (In future) is to be Evernote like application, you can also take notes and code snippets from the repositories.

Vision

The ultimate goal is to be a platform lets its users develop their projects more collaboratively with other repositories.

B. REQUIREMENTS

1. Functional Requirements

1.1 Registration & Authentication

- User shall authenticate with her / his GitHub profile.
- User shall not create account in the application.

1.2 Profile

- User shall access basic information from her / his profile information like:
  - GitHub account name
  - E-mail used for GitHub
  - GitHub avatar

1.3 Tags

- User shall create / delete / edit tags.
- User shall assign / remove several tags to her / his starred repositories.
- User shall access all starred repositories from default All Stars tag.
- User shall access untagged starred repositories from default Untagged Stars tag.

1.4 Sort

- User shall sort tags. (Ascending / Descending)
- User shall sort the starred repositories with the selected tag. (Ascending / Descending)

1.5 Information

- User shall access main information of the repositories via README.md file resides in the directory.

- User shall access information of the repositories:
  - Name
  - Creator
  - Star information
  - Fork Information
- A link to the GitHub page of the repositories should be available.

1.7 Filter (Optional)

- User shall filter the tags with the text entered.
- User shall filter the repositories w/o the selected tag.

2. Non-functional Requirements

2.1 Tools

- Back-end and application programming interface of the system shall be implemented using Node.js.
- Applications of the system shall be implemented using React.
- JavaScript shall be programming language.
- MongoDB shall be used as database management system.
- Travis CI shall be used for continuous integration.

2.2 Security

- SSL / TLS should be used. (Optional)

C. SYSTEM ARCHITECTURE

Githubnize is a client-server application therefore it can be divided into two main components.

1. BACK-END SERVER

All the logical functions of system will be performed by a RESTFUL API. Which will act as a bridge between Persistent Data Storage and client. Clients will access API with authorization token to request operations. For the login required endpoints access token should be provided by client. Access token will be generated and issued to the client in form of JWT by using auth endpoint. Endpoints will receive and output information in form of JSON.

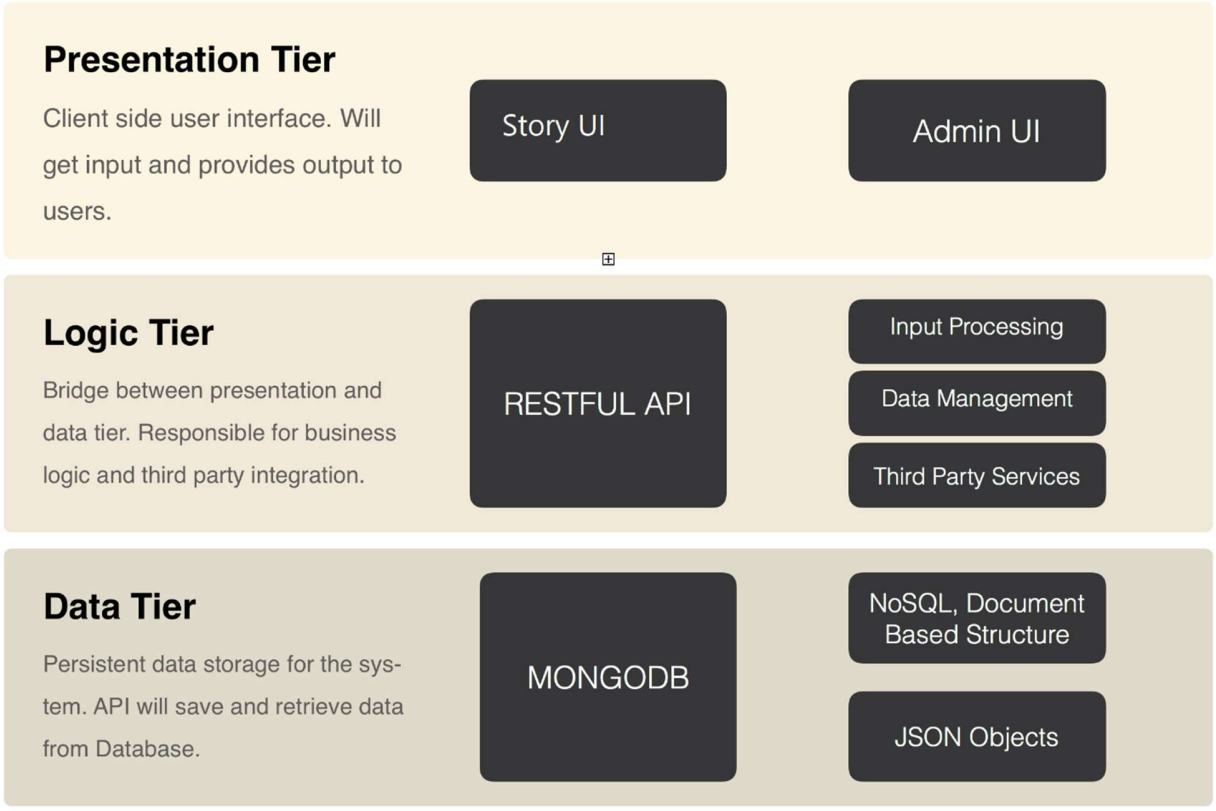
To provide maximum security and scalability back end server will be running on Amazon Cloud Service (AWS).

RESTFUL API will be written in Express Framework on Node.js to benefit from high throughput capacity.

2. CLIENT SIDE

Client side which provides User Interface will be written in React to benefit from reusable and easy to use components. *Redux* and *immutable* should be used for data management on client side. As described, system will use Three-

tier architecture pattern, which is illustrated below.



3. RESTFUL API ARCHITECTURE

Back-End of *Githubnize* will be exposed as a restful API to the client applications. MongoDB will be used for database and Node.js will be used for API. JWT tokens will be used for API authentication. JWT token will be generated for user upon sign-in and it should be stored in client-side for future requests.

4. API ENDPOINTS

General Properties for Endpoints

- API uses semantic versioning v1/, v2/
- API output as JSON and able to get requests with JSON objects.
- API responses follow HTTP Response Code standards.
- Routes configured by RESTFUL approach.
  - GET resource/ -> List resources
  - GET resource/:id -> Output specific resource
  - POST resource -> Create resource
  - PUT resource/:id -> Update specific resource
  - DELETE resource/:id -> Delete specific resource

4.1 API ENDPOINTS

Authenticate Endpoint

Endpoint consumes *GithubnizeUser* Schema

POST /auth/callback

- The endpoint is called by GitHub authentication service

Tag Endpoint

Endpoint consumes *GithubnizeTag* Schema and *GithubnizeRepo* Schema

GET /tag

- Lists all user tags

GET /tag/:tagId

- Lists all repositories added to the tag with the given id

POST /tag

- Creates a new tag

POST /tag/:tagId

- Adds the repository to the tag with the given id

PUT /tag/:tagId

- Updates the tag with the given id with the given data

DELETE /tag/:tagId

- Deletes the tag with the given id

**DELETE /tag/:tagId/:repoId

- Removes the repository with the given id from the tag with given id

Repo Endpoint

Endpoint consumes *GithubnizeTag* Schema and *GithubnizeRepo* Schema

GET /repo/:repoId

- Lists all tags assigned to the repository with the given id

4.1 GITHUB ENDPOINTS

User Info

GET /user

- Gets the authenticated user info

Starred

GET /user/starred

- List repositories being starred by the authenticated user

Repositories

GET /repos/:owner/:repo

- Get the repository information

D. CLIENT-SIDE APPLICATION ARCHITECTURE

Modern Front-end technologies are used in client-side web application. Application developed on top of React and Redux with ECMAScript 6 and Webpack is used for transpiling, linting and creating optimized build scripts.

1. Technology Overview

React

In computing, React (sometimes styled React.js or ReactJS) is a JavaScript library for building user interfaces.

It is maintained by Facebook, Instagram and a community of individual developers and corporations.

React allows developers to create large web-applications that use data and can change over time without reloading the page. It aims primarily to provide speed, simplicity, and scalability. React processes only user interfaces in applications. This corresponds to View in the Model-View-Controller (MVC) pattern, and can be used in combination with other JavaScript libraries or frameworks in MVC, such as AngularJS. Our front-end structure which built with React:

Redux

Redux is a predictable state container for JavaScript apps. It helps to write applications that behave consistently, run in different environments (client, server, and native), and are easy to test. On top of that, it provides a great developer experience, such as live code editing combined with a time traveling debugger (Available as Chrome extension). Redux creates a global singleton State store for react application on top of Functional Programming Practices that are actions and reducers.

Immutable

Immutable data cannot be changed once created, leading to much simpler application development, no defensive copying, and enabling advanced memoization and change detection techniques with simple logic. Persistent data presents a mutative API which does not update the data in-place, but instead always yields new updated data.

Immutable.js provides many Persistent Immutable data structures including: List, Stack, Map, OrderedMap, Set, OrderedSet and Record.

These data structures are highly efficient on modern JavaScript VMs by using structural sharing via hash maps tries and vector tries as popularized by Clojure and Scala, minimizing the need to copy or cache data. By using immutable with React and Redux we will achieve strong and persistent Application state management.

Webpack

Webpack is a powerful module bundler for javascript which supports hot module reloading, lazy loading, bundle splitting, hashing and source maps. It takes all kinds of assets which are used for modern web development such as source code, images, css, etc. and turns that into an optimized and compressed bundle which can be used by client browsers.

2. Application Structure

Client web application is a Single Page React Application.

Routes

Responsible for application routing. Redirects incoming routes to the related container pass required parameters to container and manages browsers history state.

Containers

Containers are the controllers for the route. Containers are responsible for requesting data by dispatching actions and transfers data to the dump components within page. They are directly connected to the Redux store and contains dump components. App container is the root container of the system and all other containers are living inside App container.

Components

Components are dump and reusable views. They are not connected to the redux store they receive props from its parent and process according to their props and they delegate actions via props.

State Shaping and Redux

Requests to the API are called according to the dispatched actions from containers. Dispatched actions may be seen on the figure above on the left hand-side. Actions make required API calls by API helper and dispatch results to the Reducers.

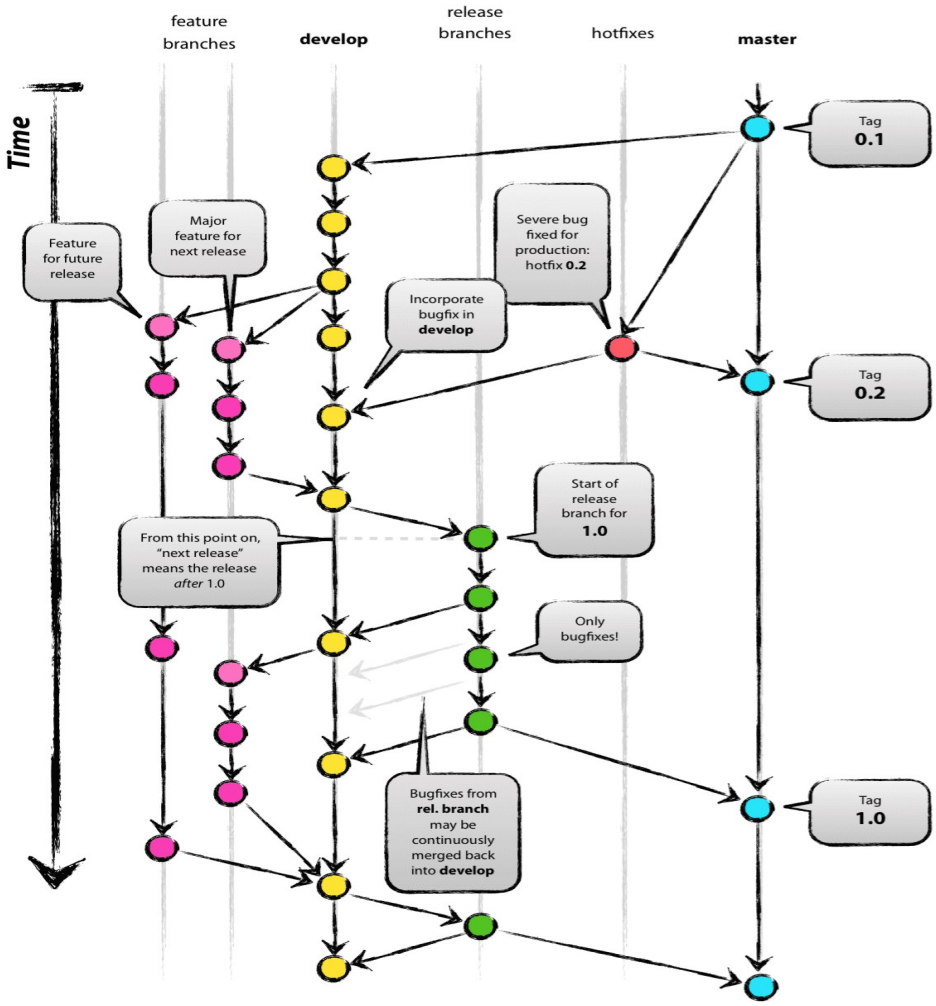
Reducers are the objects for the global redux store and they map and filter response data and store it on global state. Global state is changed after actions. On the figure above, final state of the application can be observed on the right-hand side.

It should be noted that data is stored as normalized on the store instead of arrays. Which means instead storing users as array, they have stored as key value objects according to their slugs. Also, boards of users stored in same manner. By normalizing data, it is easier to access data when necessary by just using the key, instead of searching through the array.

E. DEVELOPMENT ENVIRONMENT

For both backend and client, Git was used for code versioning and GitHub was used for centralized repository. For development environment and release management we have used Git flow branching model which is proven to be very effective for multiple developer environments and safe for release management.

Git Flow



As illustrated above, Git flow branching basically depends on separated branches. There are two main branches, one is master which represents production code and one is develop which is development code.
Developers should create branches from develop to make their changes to the code and after their work finalize they should open pull requests to the develop branch.

1. Repos and Environment Variables

There are separate GitHub repositories for Backend Code and Client Code. Both repo should have configured to have develop branch as origin. And both repos configured to read config variables from JSON files by environment type. By using separate config files for different environments, it is easy to configure and safer to split production and development environment from each other without changing the code.

Preparing the Development Environment for Backend API

Prerequisites

- *Node*
- *MongoDB*

Installing Node with NVM on Ubuntu

First, we need to install Nodejs to Ubuntu server. We will use nvm (node version manager) to manage NodeJS installations.

Note: Sudoer account will be necessary to install NodeJS on Ubuntu
We will install required dependencies to build nvm by typing below to the terminal.

```
$ sudo apt-get update
$ sudo apt-get install build-essential libssl-dev
```

After the dependencies successfully installed we may install nvm by following;

```
$ curl -sL https://raw.githubusercontent.com/creationix/nvm/v0.31.0/install.sh -o install_nvm.sh

$ bash install_nvm.sh

$ source ~/.profile
```

Once nvm installed, we can install all versions of NodeJS by using it. For our application we will be using latest stable NodeJS version.

```
$ nvm install stable
```

Following Node.JS libraries will be used:

2. Backend Tools

Package	Link	Explanation
bcryptjs	https://www.npmjs.com/package/bcryptjs	Optimized bcrypt in JavaScript with zero dependencies.
body-parser	https://www.npmjs.com/package/body-parser	Node.js body parsing middleware.
compression	https://www.npmjs.com/package/compression	Node.js compression middleware.
cors	https://www.npmjs.com/package/cors	CORS is a node.js package for providing a Connect/Express middleware that can be used to enable CORS with various options.

Package	Link	Explanation
express	https://www.npmjs.com/package/express	Fast, unopinionated, minimalist web framework for node.
express-boom	https://www.npmjs.com/package/express-boom	Boom response objects in Express.
helmet	https://www.npmjs.com/package/helmet	Helmet helps you secure your Express apps by setting various HTTP headers.
jsonld	https://www.npmjs.com/package/jsonld	This library is an implementation of the JSON-LD specification in JavaScript.
jsonwebtoken	https://www.npmjs.com/package/jsonwebtoken	An implementation of JSON Web Tokens.
mongoose	http://mongoosejs.com/	elegant mongodb object modeling for node.js
chai-as-promised	https://github.com/domenic/chai-as-promised	Chai as Promised extends Chai with a fluent language for asserting facts about promises.
eslint	https://eslint.org/	The pluggable linting utility for JavaScript and JSX
nodemon	https://nodemon.io/	Nodemon is a utility that will monitor for any changes in your source and automatically restart your server
nyc	https://www.npmjs.com/package/nyc	Istanbul's state of the art command line interface, with support for: applications that spawn subprocesses

Installing Mongodb on Ubuntu

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv
0C49F3730359A14518585931BC711F9BA15703C6

$ echo "deb [ arch=amd64,arm64 ] http://repo.mongodb.org/apt/ubuntu xenial/mongodb-
org/3.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.4.list

$ sudo apt-get update

$ sudo apt-get install -y mongodb-org
```

After installing mongodb, mongo service should be started;

```
$ sudo service mongod start
```

Configuring Mongodb

To Create Database user and test user;

```
#!/bin/bash

# Dropping Users
mongo githubnize         --eval 'db.dropUser("githubnize")'
mongo githubnize-dev  --eval 'db.dropUser("githubnize-dev")'
mongo githubnize-test --eval 'db.dropUser("githubnize-test")'

# Dropping Databases
mongo githubnize         --eval 'db.dropDatabase()'
mongo githubnize-dev  --eval 'db.dropDatabase()'
mongo githubnize-test --eval 'db.dropDatabase()'
```

```
# Creating users
# No users should be added for production database by now.
mongo githubnize-dev  --eval 'db.createUser({user:"githubnize-dev",pwd:"123456",
roles:[{role:"readWrite",db:"githubnize-dev"}]});'
mongo githubnize-test --eval 'db.createUser({user:"githubnize-test",pwd:"123456",
roles:[{role:"readWrite",db:"githubnize-test"}]});'
```

Getting Code and Configuring for Development

Code for Backend API is stored on Github repository. To clone repository;
Note: Git should be installed.

```
$ git clone https://github.com/mustafagonul/githubnize-api.git

$ git clone https://github.com/mustafagonul/githubnize-app.git
```

After cloning the repository, environment configuration should be created;

```
$ cp src/config/enviroments/enviroments.json.example development.json
$ cp src/config/enviroments/enviroments.json.example  test.json
$ cp src/config/enviroments/enviroments.json.example  production.json
```

After creating json files, DB configuration should be filled with respect to previous steps. Also twitter consumer key and consumer secret should be filled.
After above requirements fulfilled, to run development backend application;
To install dependencies;

```
$ npm install
```

To start project;

```
$ npm start
```

To run linter;

```
$ npm run lint
```

To run tests, jest is used for tests;

```
$ npm run test
```

To create optimized production build;

```
$ npm run build
```

Build will be placed under build directory.

3. Test Tools

Package	Link	Explanation
chai	http://chaijs.com/	Chai is a BDD / TDD assertion library for node and the browser that can be delightfully paired with any javascript testing framework.
sinon	http://sinonjs.org/	Standalone test spies, stubs and mocks for JavaScript.
mocha	https://mochajs.org/	Mocha is a feature-rich JavaScript test framework

Package	Link	Explanation
chai-HTTP	https://github.com/chaijs/chai-http	HTTP integration testing with Chai assertions.

F. DEPLOYMENT

Githubnize consist of 2 main components:

- BACKEND : Githubnize API
- FRONTEND: Githubnize APP

Githubnize can be install to your local directory automatically on linux based distributions by using following scripts, which can be found in project files under Scripts directory:

- prepare_api.sh
- prepare_app.sh

Also, there is an automated DB clean up script for local deployment. PLEASE note that using docker prevents you doing that:

- resetdb.sh

Please refer this wiki page and readme.md for more information:

https://github.com/mustafagonul/Fall2017Swe573/wiki/Auto-Deployment

https://github.com/mustafagonul/githubnize-api/blob/development/README.md

https://github.com/mustafagonul/githubnize-app/blob/development/README.md