Technical guide
Dribdat 2.0

Module 646-2
Projects

Students : Jonathan Schnyder, Caroline Moreira,
Anthony Ritz, Daniela Lourenço,
Mickaël Coluccia

Teacher : Cotting Alexandre

Product Owner : Oleg Lavrovsky

29th April 2019

Contents

1. Introduction

As part of the 646-2 Projects course, we developed a hackathon web site in vuejs and used the SCRUM methodology.

A version of this website already exists (dribdat 1), we had to be inspired to try to make an improved version (dribdat 2) completed and more simplified.

The goal of this technical guide is to provide useful information for further developers in case of the client would re-use what the team has done.

2. Source Code

Our application's source code can be found here : https://github.com/hackathons-ftw/dribdat2_frontend

We are using the dribdat application's API and modified a little the main route for retrieving data. It can be found here : https://github.com/hackathons-ftw/dribdat2

3. Technologies used

3.1 Frontend

Vue.JS: Vue is a progressive framework for building user interfaces. The core library is focused on the view layer only and is easy to pick up and integrate with other libraries or existing projects.

Vue CLI: It was used to work the command line project

Module VueX : VueX is used to have a state management pattern and a library

for Vue.js applications.

Module Axios : The module AXIOS is used to retrieve API data.

Node.js : it's used to retrieve all the changes in a development environment

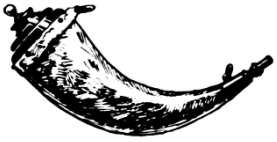
DISQUS : It is a comment plug-in, we used it to be able to leave a feedback

GitHub : GitHub is used to recover activities and issues of the repository.

Bootstrap : Bootsrap is used to do the design

3.2 Backend

The backend was taken from the previous version of Dribdat (dribdat 1). We made only some changes and added roots.

Python: The version of python we used for this project is the version "python2.7" which is compatible with flack.

Flask: Use python 2.7 to generate a virtual environment

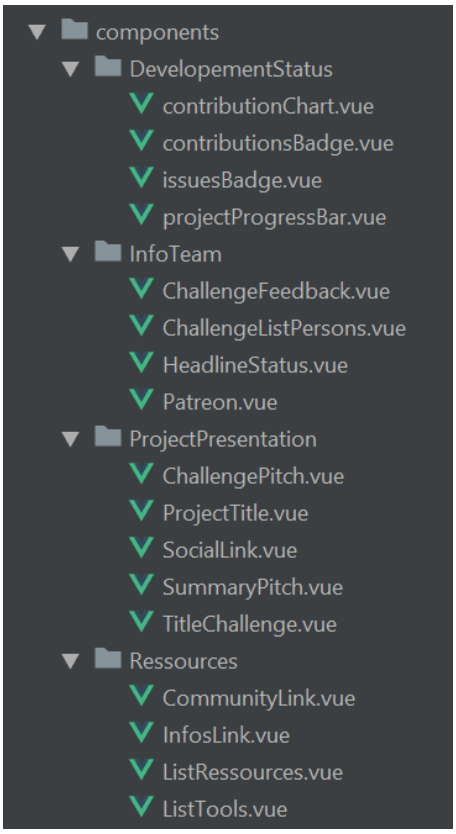
3.3 Unit test

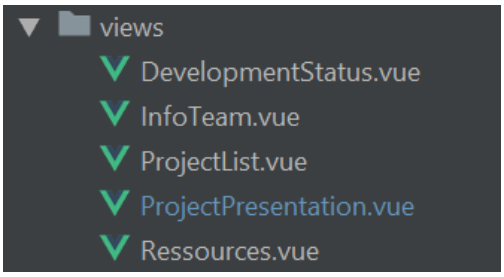
JEST : To do the unit test with continuous integration of **Travis**.


4. Project

Vue.js

The project was organized according to the recommendation vue.JS. We separate the different screens in a specific folder and each folder contains all the components for this screen. And then each component is added in the correspondent Vue.





Vue file are split in 3 parts:

```
1  <template>···
58
59  <script>···
64
65  <style scoped>···
```

In the <template> tag you'll write your html code specific to your component. You can also call component that you have created. (e.g. Navigation.vue

```
<template>
  <div id="project">
    <navigation :id="this.id"></navigation>
    <div class="content">
      <router-view :project="custom_project"></router-view>
    </div>
    <div class="content-bottom">
      <Footer :project="custom_project"></Footer>
    </div>
  </div>
</template>
```

In the <script> tab you'll write your javascript code to set variables, add methods, get props from the component, etc...

```
<script>
    export default {
        name: "Stepper",
        props: ["steps", "currentStep"],
        data: function() {···
        },
        methods: {
            editSteps() {
                this.tempProgress = this.$store.state.custom_project.progress
                this.isEditMode = !this.isEditMode
            },
            cancelEdit(){
                this.$store.dispatch("setProjectProgress", this.tempProgress)
                this.isEditMode = false
            },
            setNextStep(){
                this.$store.dispatch("setProjectProgress", this.$store.state.custom_project.progress + 1)
            },
            setPreviousStep(){
                this.$store.dispatch("setProjectProgress", this.$store.state.custom_project.progress - 1)
            },
            getEditButtonText() {
                return (this.isEditMode)? 'Save' : 'Set current step'
            },

        }
    };
</script>
```

In the <style> tag you'll write your CSS code. You can "scoped" your style to this component to avoid modifying other components with your specific style.

```
<style scoped>
#myFooter li {
  font-size: 11px;
}

#myFooter {
  background-color: #f4fcfc;
  opacity: 0.87;
  color: white;
  margin-top: 40px;
  border: 1px groove #333333;
  border-radius: 10px;
}

#myFooter .footer-copyright {
  background-color: #3a3838;
  padding-top: 3px;
  padding-bottom: 3px;
  text-align: center;
  font-size: 13px;
}
```

Vuex

According to the official documentation, Vuex is a **state management pattern + library** for Vue.js applications. It serves as a centralized store for all the components in an application, with rules ensuring that the state can only be mutated in a predictable fashion. It also integrates with Vue's official devtools extension to provide advanced features such as zero-config time-travel debugging and state snapshot export / import.

In the **main.js** file we import the store that implements Vuex:

```
1  import Vue from 'vue'
2  import App from './App.vue'
3  import router from './router'
4  import store from './store'
5  import './registerServiceWorker'
6
7  //Libray to use the different icon in the web app
8  import { library } from '@fortawesome/fontawesome-
9  import { faLightbulb, faUsers, faFileCode, faLink,
10 import { FontAwesomeIcon } from '@fortawesome/vue-
11
12
13 library.add(faLightbulb,faUsers,faFileCode,faLink,
14 Vue.component('font-awesome-icon', FontAwesomeIcon
15 Vue.config.productionTip = false
16
17
18
19 require('./design/custom.css');
20
21 new Vue({
22   router,
23   store,
24   render: h => h(App)
25 }).$mount('#app')
```

In the **store.js** file we export a new Vuex.Store with some parameters:

```
1   import Vue from 'vue'
2   import Vuex from 'vuex'
3   import axios from 'axios'
4   import VueAxios from 'vue-axios'
5
6   Vue.use(Vuex)
7   Vue.use(VueAxios, axios)
8
9   let github_apiURL = 'https://api.github.com/repos'
10  //let path = '/repos/ChallengeHunt/challengehunt'
11  const Backend_API_URL = 'http://127.0.0.1:5000/api'
12
13  export default new Vuex.Store({
14      state: {···
39      },
40      mutations: {···
65      },
66      actions: {···
157     },
158     getters: {···
162     }
163 })
164
```

State

We create some variable that we'll use later in our program and that we want to keep changes without save it on database. Custom_project is a copy of the project loaded from the API and will be used inside the application to avoid errors by modifying the other project. We set also this variable for the "edit" part (that has been comment in the code after discussion with the product owner).

```
state: {
    github_BaseURL: '',
    github_repoPath: '/repos/ChallengeHunt/challengehunt',
    project: {},
    custom_project: {
        name: "getaround.io",
        summary: "We want to get people to be more concerned about their health.
        challenge: {
            //name: "Help people be more active"
            name: ""
        },
        id: 1,
        pitch: "'https://www.youtube.com/embed/0lwOmIHcSno'",
        phase: 4,
        status: "Looking for designers",
        event: {
        }
    },
    contributors: [],
    issues: [],
    projectList: [],
    editMode: false
```

Mutations

Mutations allows you to modify stored variables by calling these methods.

```
mutations: {
    SET_CONTRIBUTORS (state, contributors) {
        state.contributors = contributors
    },
    SET_ISSUES (state, issues) {
        state.issues = issues
    },
    SET_PROJECT (state, project) {
        state.project = project
    },
    SET_CUSTOM_PROJECT (state, custom_project) {
        state.custom_project = custom_project.project
        state.custom_project.challenge = { name: '' }
        state.custom_project.event = custom_project.event
        state.custom_project.pitch = 'https://www.youtube.com/embed/0lwOmIHcSno'
    },
    SET_PROJECT_LIST (state, projectList) {
        state.projectList = projectList;
    },
    SET_EDITABLE (state, editMode) {
        state.editMode = editMode;
    },
    SET_PROJECT_PROGRESS (state, progress) {
        state.custom_project.progress = progress
    }
},
```

Actions

Mutations allows you to modify stored variables by calling these methods from your Vue or Js files.

```
actions: {
    loadContributors ({ commit, state }) {
        axios
            .get(github_apiURL
                + state.custom_project.source_url.replace('https://github.com', '')
                + '/stats/contributors')
            .then(r => r.data)
            .then(contributors => {
                commit('SET_CONTRIBUTORS', contributors)
            })
    },
    loadIssues ({ commit, state }) {···
    },
    loadProject ({ commit }) {···
    },
    loadCustomProject ({ commit }, id) {···
    },
    loadProjectList ({ commit }, event) {···
    },
    setModeEdit ({ commit }) {···
    },

    setModeDisplay ({ commit }) {···
    },
    setProjectProgress ({ commit }, progress) {···
    }
},
```

Getters

Getters allows you to get data from the store.

```
getters: {
    projectSourceAPI_Path: state => {
        return state.custom_project.source_url.replace('https://github.com', '')
    },
}
```

4.1 API

There are several API calls that admins can use to easily get to the data in Dribdat in CSV or JSON format. See GitHub issues for development status.

Basic data on an event :

- `/api/event/<EVENT ID>/info.json`
- `/api/event/current/info.json`

Retrieve data on all projects from an event:

- `/api/event/<EVENT ID>/projects.csv`
- `/api/event/<EVENT ID>/projects.json`
- `/api/event/current/projects.json`

Recent activity in projects (all or specific):

- `/api/project/activity.json`
- `/api/<PROJECT ID>/activity.json`

Search project contents:

- `/api/project/search.json?q=<text_query>`

Push data into projects (WIP):

- `/api/project/push.json`

We only use the `/api/project/{id}/info.json` route to gather all the relevant data for a project. the data is structured like this :

```
creator:
    id:              1
    username:        "jon"
event:
    community_url:   ""
    ends_at:         "Thu, 18 Apr 2019 16:00:00 GMT"
    has_finished:    true
    has_started:     false
    hostname:        "Me"
    id:              1
    location:        "Here"
    name:            "Hackathon1"
    starts_at:       "Mon, 18 Feb 2019 13:08:00 GMT"
    webpage_url:     ""
  is_webembed:       true
  phase:             "Sketching"
  pitch:             "<iframe src=\"https://hackdash.org\"></iframe>"
project:
    contact_url:     "https://github.com/impronunciable/hackdash/issues"
    hashtag:         ""
    id:              1
    image_url:       "https://avatars0.githubusercontent.com/u/165799?v=4"
    name:            "Hackdash-V1"
    phase:           "Sketching"
    progress:        10
    score:           43
    source_url:      "https://github.com/impronunciable/hackdash"
    summary:         "Ideas for Hackathons"
team:
  0:
      id:            2
      link:          ""
      name:          "Bob"
  1:
      id:            3
      link:          ""
      name:          "Marc"
  2:
      id:            1
      link:          "https://github.com/jonHESSO"
      name:          "jon"
```


This data should be pretty self-explanatory. However some parts still need a

little explanation:

- is_webembed : boolean indicating if the project pitch should be embed into the project presentation page.
- pitch : url of the pitch to display
- project.score : completion of page, calculated by the backend
- source_url : url to the source code repository

4.2 Développer guide

Backend

Install Python, Virtualenv and Pip or Pipenv to start working with the code. You may need to install additional libraries (`libffi`) for the misaka package, which depends on CFFI, e.g. `sudo dnf install libffi-devel`.

Run the following commands from the repository root folder to bootstrap your environment using Pipenv:

```
pipenv --three
```

```
pipenv shell
```

```
pipenv install
```

Or using plain pip:

```
pip install -r requirements/dev.txt
```

By default in a dev environment, a SQLite database will be created in the root folder (`dev.db`). You can also install and configure your choice of DBMS supported by SQLAlchemy.

Run the following to create your app's database tables and perform the initial migration:

```
python manage.py db init
```

```
python manage.py db migrate
```

```
python manage.py db upgrade
```

Finally, run this command to start the server:

```
FLASK_DEBUG=1 python manage.py run
```

You will see a pretty welcome screen at http://localhost:5000

Frontend

Node.js, git and npm must be installed in your environment to be able to launch the frontend.

Firstly, clone our repository:

```
git clone https://github.com/hackathons-ftw/dribdat2_frontend.git
```

Then, run the following command to install all the missing dependencies:

```
npm install
```

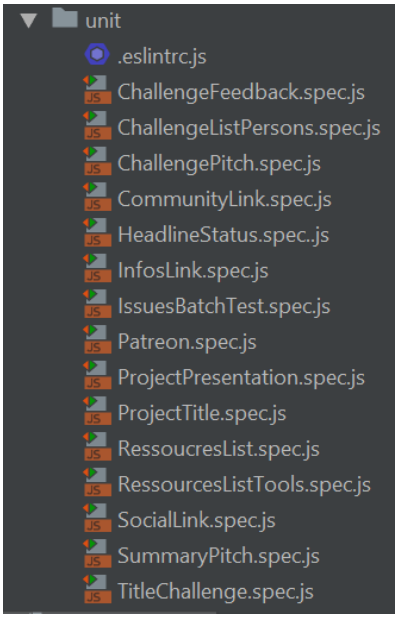
Finally, launch the node server:

```
npm run serve
```

Frontend should be running at http://localhost:8080

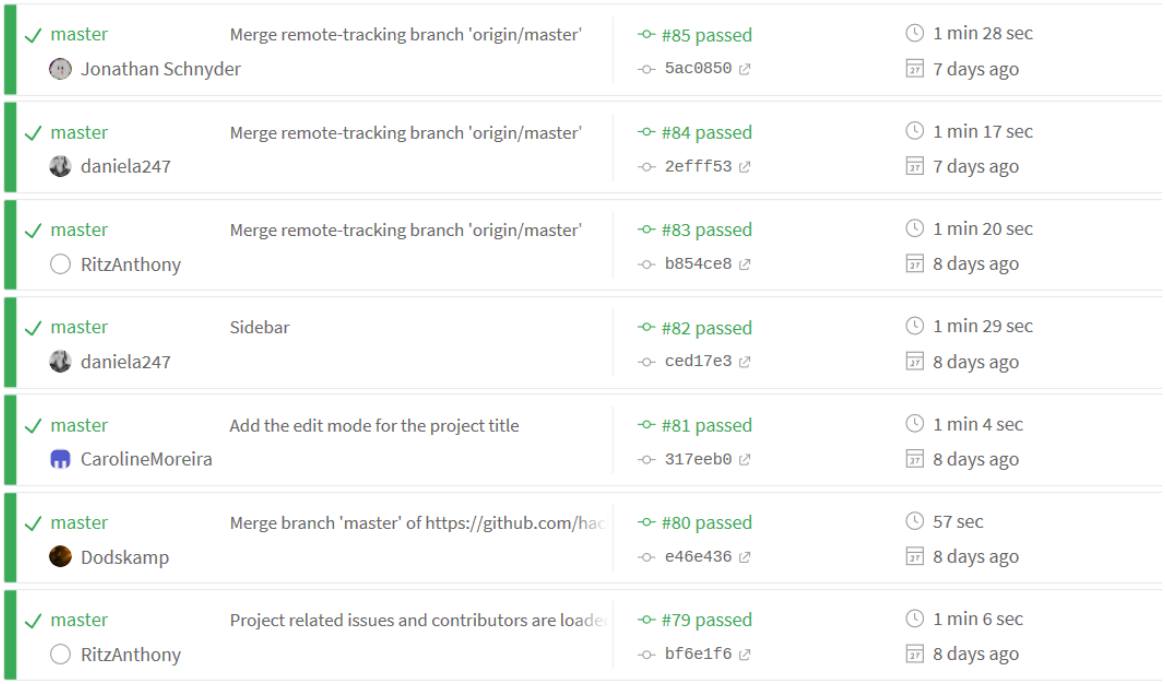
5. Unit Test

All the tests are in the "unit" folder and are written with: .spec.js.



We use JEST to create, execute and test the components. The tests are done in order to check if the component is displayed correctly or if the behavior of the component is correct. The tests are executed automatically with each commit using the continuous integration of Travis.

Example of Build History:



Result on Travis when a test is passed:

```
File                                 |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovered Line #s |
-------------------------------------|----------|----------|----------|----------|-------------------|
All files                            |    13.17 |    21.74 |     8.11 |    13.33 |                   |
 src                                 |        0 |        0 |        0 |        0 |                   |
  APIService.js                      |        0 |        0 |        0 |        0 |... 42,43,46,53,60 |
  registerServiceWorker.js           |        0 |        0 |      100 |        0 |... 17,20,23,26,29 |
  router.js                          |        0 |      100 |        0 |        0 |... 15,25,32,37,42 |
  store.js                           |        0 |        0 |        0 |        0 |... 18,122,126,131 |
 src/components                      |        0 |      100 |        0 |        0 |                   |
  EditButton.vue                     |        0 |      100 |      100 |        0 |          25,28,31 |
  Project.vue                        |        0 |      100 |        0 |        0 |       12,13,19,25 |
  Stepper.vue                        |        0 |      100 |        0 |        0 |                31 |
 src/components/DevelopementStatus   |        0 |        0 |        0 |        0 |                   |
  contributionChart.vue              |        0 |        0 |      100 |        0 |... 24,34,35,48,49 |
  contributionsBadge.vue             |        0 |      100 |      100 |        0 |... 46,60,62,63,65 |
  projectProgressBar.vue             |        0 |      100 |        0 |        0 |          14,15,21 |
 src/components/InfoTeam             |       75 |       50 |    66.67 |       75 |                   |
  ChallengeFeedback.vue              |       75 |       50 |    66.67 |       75 |             11,12 |
 src/components/ProjectPresentation  |      100 |       80 |      100 |      100 |                   |
  ChallengePitch.vue                 |      100 |       80 |      100 |      100 |                32 |
 src/components/Ressources           |       20 |        0 |      100 |       20 |                   |
  ListRessources.vue                 |       20 |        0 |      100 |       20 |       43,46,47,48 |
  ListTools.vue                      |       20 |        0 |      100 |       20 |       41,44,45,46 |
 src/views                           |     12.5 |      100 |        0 |     12.5 |                   |
  DevelopmentStatus.vue              |        0 |      100 |        0 |        0 |... ,96,97,101,102 |
  InfoTeam.vue                       |        0 |      100 |        0 |        0 |... 99,102,110,111 |
  ProjectList.vue                    |        0 |      100 |      100 |        0 |          12,17,22 |
  ProjectPresentation.vue            |      100 |      100 |      100 |      100 |                   |
  Ressources.vue                     |        0 |      100 |      100 |        0 | 14,15,16,17,18,26 |
-------------------------------------|----------|----------|----------|----------|-------------------|

Test Suites: 14 passed, 14 total
Tests:       18 passed, 18 total
Snapshots:   0 total
Time:        5.058s
Ran all test suites.
The command "npm run test:unit" exited with 0.
```

Result on Travis when a result is not passed:

```
 Footer.vue                          |        0 |      100 |      100 |        0 |                57 |
 Project.vue                         |        0 |      100 |        0 |        0 |   14,15,16,22,28 |
 Stepper.vue                         |        0 |        0 |        0 |        0 |... 60,61,64,67,70 |
src/components/DevelopementStatus    |        0 |        0 |        0 |        0 |                   |
 contributionChart.vue               |        0 |        0 |      100 |        0 |... 24,34,35,48,49 |
 contributionsBadge.vue              |        0 |      100 |      100 |        0 |... 46,60,62,63,65 |
 projectProgressBar.vue              |        0 |      100 |        0 |        0 |          16,17,23 |
src/components/InfoTeam              |       75 |       50 |    66.67 |       75 |                   |
 ChallengeFeedback.vue               |       75 |       50 |    66.67 |       75 |             11,12 |
src/components/ProjectPresentation   |      100 |      100 |      100 |      100 |                   |
 ChallengePitch.vue                  |      100 |      100 |      100 |      100 |                   |
src/components/Ressources            |    18.18 |        0 |        0 |    18.18 |                   |
 ListRessources.vue                  |       20 |        0 |      100 |       20 |       45,48,49,50 |
 ListTools.vue                       |       20 |        0 |      100 |       20 |       42,45,46,47 |
 TicketService.vue                   |        0 |      100 |        0 |        0 |                13 |
src/views                            |    12.82 |      100 |        0 |    12.82 |                   |
 DevelopmentStatus.vue               |        0 |      100 |        0 |        0 |... ,96,97,101,102 |
 InfoTeam.vue                        |        0 |      100 |        0 |        0 |... 03,106,114,115 |
 ProjectList.vue                     |        0 |      100 |      100 |        0 |          12,17,22 |
 ProjectPresentation.vue             |      100 |      100 |      100 |      100 |                   |
 Ressources.vue                      |        0 |      100 |      100 |        0 |   14,15,16,17,25 |
-------------------------------------|----------|----------|----------|----------|-------------------|

Test Suites: 1 failed, 13 passed, 14 total
Tests:       1 failed, 17 passed, 18 total
Snapshots:   0 total
Time:        5.462s
Ran all test suites.
npm ERR! code ELIFECYCLE
npm ERR! errno 1
npm ERR! dribdat2_frontend@0.1.0 test:unit: `vue-cli-service test:unit`
npm ERR! Exit status 1
npm ERR!
npm ERR! Failed at the dribdat2_frontend@0.1.0 test:unit script.
npm ERR! This is probably not a problem with npm. There is likely additional logging output above.

npm ERR! A complete log of this run can be found in:
npm ERR!     /home/travis/.npm/_logs/2019-04-16T07_07_33_977Z-debug.log
The command "npm run test:unit" exited with 1.
```

5.1 Possible issues

If you get these errors:

```
FAIL tests/unit/SocialLink.spec.js
  ● Test suite failed to run

    TypeError: Cannot assign to read only property 'Symbol(Symbol.toStringTag)' of object '#<process>'

      at exports.default (node_modules/jest-util/build/create_process_object.js:15:34)

FAIL tests/unit/TitleChallenge.spec.js
  ● Test suite failed to run

    TypeError: Cannot assign to read only property 'Symbol(Symbol.toStringTag)' of object '#<process>'

      at exports.default (node_modules/jest-util/build/create_process_object.js:15:34)

FAIL tests/unit/SummaryPitch.spec.js
  ● Test suite failed to run

    TypeError: Cannot assign to read only property 'Symbol(Symbol.toStringTag)' of object '#<process>'

      at exports.default (node_modules/jest-util/build/create_process_object.js:15:34)

FAIL tests/unit/IssuesBatchTest.spec.js
  ● Test suite failed to run

    TypeError: Cannot assign to read only property 'Symbol(Symbol.toStringTag)' of object '#<process>'

      at exports.default (node_modules/jest-util/build/create_process_object.js:15:34)
```

You must change your version of node by using this command line:

Windows:

```
npm install -g nvmw
```

```
nvmw use 11.10.1
```

Linux:

```
npm install -g nvm
```

```
nvm use 11.10.1
```