INTERNSHIP REPORT

Tech-5

This is the Tech-5 internship report. It details the two main projects, gives technical documentation and reasons to take the trainee another time. The mission was to correct bugs, improve the web site and create a mobile application IOS and Android.

Issaka GALADIMA-IBRAHIM

Galadi_i

Society: Next-W

Internship period: 01/04/2015 – 30/09/2015

Internship report

Tech-5

Summary

Intership report

[Part 1]

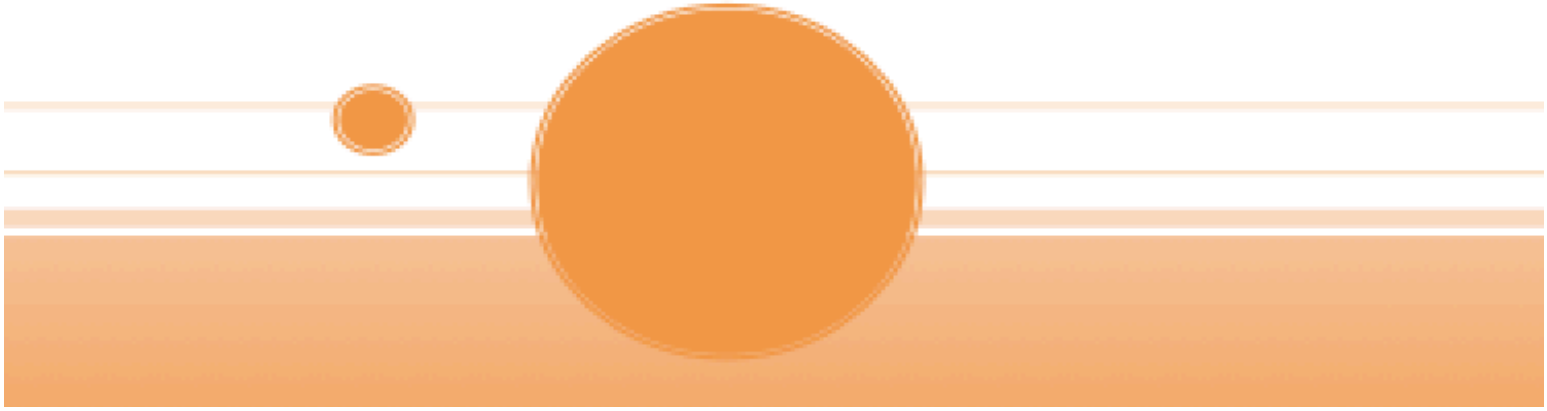
Report for the next programmer who will work on the project.

Issaka GALADIMA-IBRAHIM

Galadi_i

Society: Next-W

Internship period: 01/04/2015 – 30/09/2015



Summary

Introduction

This point describes everything a new developer in the team should know in order to be able to integrate the team easily, understand the projects I've been working on and quickly start to develop.

A. The company:

Next-W is a society based at Paris, located at 18 rue Yves Toudic 75010 Paris. The company created an email service (NtyMail) and a social network (Newmanity Community).

1. About Next W:

The main activity of the society is about emailing. Next-W provides a new way to use e-mail 100% eco-friendly, no personal information kept or sold, no advertisements while writing or reading messages and still free to use: the NtyMail.

Next-W main partner makes all this possible: the Green Fan, which is a company that uses green servers. That means its servers are powered by electricity produced exclusively from renewable energy sources, the carbon footprint is then neutral.

The second project of Next-W is "The Newmanity Community", which is a web site and a mobile application is made for people and societies who want to exchange, mobilize and engage themselves to create a better world. It allows them to create pledges, events, invite and find volunteers for charitable actions.

In August 2015, these two projects gathered together 75.000 users from all around the world.

2. The team organization

Next W is split in four teams: marketing, technical, graphic and administrative. As a developer you will be working in Technical and you will have a lead developer who will give you functionalities to develop on the website or on the mobile application according to the current needs and your specialties.

You may also work sometimes with the graphic team in order to find the simplest way to integrate a mockup.

The technical team is also split in three sub-teams: backend, frontend and mobile developers. The backend and frontend teams usually work on the Newmanity Community web site and the mobile team on the Newmanity Community mobile application. However, the NtyMail project is maintained by an external team: the YPCI company.

All these teams use the same workflow services (Jira and Github) to manage the tasks and the bug reports.

3. Development workflow

Newmanity developers and graphic artists use Github to manage files upload. All the tasks are listed and managed with Jira (an online task management service). You will need to know how these two services work and how the teams use them.

a. Jira:

Jira is a useful tool to manage tasks. The teams use it to collect all tasks to do, to attribute them to a developer or to plan some of them.

When a task is attributed to someone the first thing to do is to evaluate the time it will take to be done. This time is always evaluated in “hours” and can’t be more than fourteen hours, otherwise the task have to be split.

If anyone sees a new bug on any application, he has to record it in Jira by adding a new task online, and set it in “Bug“ category. The team leader of the application will then attribute the task to a developer.

b. Github:

In Newmanity, the developer’s main tool to share files, and manage the code is Github. That is why you have to know all the basics of Github, otherwise you won’t be able to participate to the development.

Therefore, it is very important for each developer to know how the teams use GitHub, and to keep using the same patterns. You have to use the following steps to be sure you are doing things in the good way.

- Fetching branch

Once a task is added on Jira, the developer in charge has to update (pull) the “develop“ branch on Github. Then create a new branch based on “develop“, in which he will develop the task.

$ git pull origin develop

If you don’t have it in your local you have to download it from GitHub origin.

$ git remote show origin

$ git fetch origin

You will also see the branch Master. Be careful because this branch is the main branch. That means the production web site is running on this branch. Therefore it has to be stable, and used only by the lead developer. All new functionalities have to be tested before the merge.

Once you have the branch develop, you can create a new one based on it, that you will use for your code.

- Create a new branch

To create a new branch based on develop use the following commands (MY_BRANCH_NAME being the name of your new branch):

$ git checkout -b MY_BRANCH_NAME develop

While creating a new branch, the good practice is to name it by a prefix “dev-” with the name of the task. For instance “dev-adding-new-link“.

- **Development and commits:**

Once the new branch is created, you can start the development. After each step of your development you have to commit, so that your advancement can be easily followed. Use the command below to commit:
$ git add .
$ git commit -m YOUR_DESCRIPTION

- Make a pull request

You need to make a pull request before merging. It allows the lead developer to check and validate your work. Go on the github websitre (*www.github.com*). Then find and go on your branch from the website. Click on the button “Pull request“.

One you click, Github will automatically show you the conflicts with the main branch if there is some.

The lead developer will also receive a mail alert, telling him he has to review your code and validate it. Once he validate you can push the branch.

-Push your branch:

Once you have finished your task, and tested that they work the way it should, and validated your code by the lead developer, you can know merge with the develop branch. First of all, go back on the develop branch:

$ git checkout develop

Check there is no conflicts in the develop branch. Then merge your branch in develop:

$ git merge YOUR_BRANCH_NAME

When a conflict appears, it means you have modified the same file on the same line as another developer of the team. It will be displayed the following way:

```
<<<<<<< HEAD
 * Orignal text
 =======
 * Modified text
 >>>>>>> YOUR_BRANCH_NAME
```

You will have to manually edit the file and commit your modifications. When all the conflict are solved, you can push your branch:

$ git push origin YOUR_BRANCH_NAME

If the branch already exits on the server, it is possible to force the push with the following command:

$ git push -f origin YOUR_BRANCH_NAME

- Delete your branch

It is forbidden to reuse the branch you just merge on the branch develop. You have to delete it immediately to avoid confusion. Use the following command:

$ git branch -d YOUR_BRANCH_NAME

You can now check for another assignment and do the same steps again for your new task.

- Get someone else branch

This step is not mandatory, but you may need sometimes to get someone else branch, to help him or to finish his job.

First of all, get all the branches on the server. As the fellow developer has his own branch for his task you may not see it by default. Use the following command to do so:

$ git fetch origin
$ git remote show origin

We can know change our branch and work on the other one:

$ git checkout OTHER_BRANCH

Then update your local files, database and all asstes:

$ php composer.phar install
$ php app/console doctrine:schema:update --dump-sql
$ php app/console doctrine:schema:update --force
$ php app/console assetic:dump

You are now ready to work on all of the Newmanity projects. The workflows are the same for all of them.

B.The projects

As said before, two main projects leads the society: NtyMail and Newmanity community. As an external developer team already manages the NtyMail project, you will be working on Newmanity Community web site and/or mobile application.

Below is a technical description for each one.

1. The web site:

a. Languages and framework:

The used languages are all web languages: HTML/CSS, Php (with Symfony2 framwork), JavaScript (with Jquery on the frontEnd and some AngularJs on the back Office),

The main language used on the website is the Php. It's usually used on web project, but is not very effective if used alone. That is why for this project it's used with the Symfony2 Framework Version 2.5.4.

Symfony2 allows separating the web site in three parts: Models, views and controllers.

This kind of separation is called "pattern MVC":

-**The model** is the part that makes all queries relative to Database. Each Table in the Database has its model. These models contain implementations of all basic queries in MySQL (select, where, like, or_where, etc…).
-**The controllers** are the functional part of pages. They contain all php information and variables used in the web page.
-**The view** is the graphic part of a page. Each displaying page has a view. It mainly contains displaying and animation code (HTML/CSS/JavaScript).

Symfony2 also allows the developer to install widgets called "bundles" which makes the development easier. There is no need to remake functionalities that already exist somewhere else, if the "bundles" are used.

That's why if you are asked in the company to add a new functionality on the web site, try to look first on the Symfony2 official website (www.symfony.com), if there isn't already a bundle doing the job for you.

If the functionality asked is too specific to the Newmanity Website, and there are no available bundles to do it, you can then ask the lead developer the confirmation to develop it by yourself.

It's important to keep the communication between you and the lead developer and ask first, because even if you couldn't find a better way to develop he may have a better idea as he has a lot of experience.

b. How to begin (installation)

To install Newmanity on your computer, you first need to have a local server installed, the last version Php version and a MySql. Then follow the following steps.

-**Local server:**
To begin you have to install a server on your computer. If you are on PC you can download WAMP server that is an efficient local server, working with Apach and MySQL.
See the link below to download it http://www.wampserver.com/.

Once downloaded, you will have to activate the “url rewriting” and to update the “Perl” library (see WAMP documentation or Forums for more information).

-Install Databases:
Then download the databases used on the web site and install them on your local server. (Ask the society for links and password).

-FTP client:
To download some files from the server or update them, you may sometimes need a FTP client. “Filezilla” is a very simple one and is easy to use. You can download the latest version here https://filezilla-project.org/

-Symfony2:
If you have no basis on the used framework Symfony2, I advise you to read its tutorial before starting.
You can find a tutorial the official website (http://symfony.com)

-Composer:
We use Composer to manage all the project dependences. To install Composer on Linux use the command below:

$ curl -sS https://getcomposer.org/installer | php

To install Composer globally on your computer use the command below:

$ mv composer.phar /usr/local/bin/composer

-Complete installation:
Once it is done, please follow the steps below to complete the installations.

- Install bower, uglifycss and uglify-js2

$ npm install -g uglify-js2
$ npm install -g uglifycss
$ npm install -g bower

- Get the project from the github

$ git clone [PROJECT_SOURCE]
(ask the lead developer for the PROJECT_SOURCE)

- Configure permissions
On linux and MacOS, you have to configure for some folders (ask the lead developer for more details)

- Set Uploads and media cache

$ sudo setfacl -R -m u:"www-data":rwX -m u:`whoami`:rwX web/uploads web/media
$ sudo setfacl -dR -m u:"www-data":rwX -m u:`whoami`:rwX web/uploads web/media

- Execute Composer
$ php composer.phar install

-Install assets and symbolic links
$ php app/console assets:install –symlink

- Create databases
$ php app/console doctrine:database:create

- Create schema
$ php app/console doctrine:schema:create

-Install Bower dependencies

$ php app/console sp:bower:install

- Compile assets (images, css, JS)

$ php app/console assetic:dump

-Create an admin user

$ php app/console fos:user:create monpseudo monpseudo@ntymail.com p@ssword
$ php app/console fos:user:promote monpseudo ROLE_ADMIN

You can now start your development.

To develop you own bundle or improve the current ones you will need to know how the folders are organized and where to put your files.

c. The bundles structure

As said before, keeping the same organization and code pattern allows more readability for the team. We use the following guidelines for determining the structure of the bundle:

- Number of directories should not be “too large” (in most cases, maximum 10)
- Opening the directory should not bring any surprises regarding its contents or where they are located. For this reason, the first level directory is generic, rather than specific to the domain.
- Unless specified in the description of the directory, the class definition must be placed in a folder that reflects the structure of the Symfony or folder structure associated with the component that is related to the class (Twig, Doctrine).

The base was a basic bundle which is generated by the task generate:bundle. Below is what was created as a result of these assumptions:

/Command : symfony tasks.

/Controller : standard symfony folder. Controllers should have a small amount of actions and the actions themselves should not be large. Controllers exist only in FrontBundle and AdminBundle.

/DependencyInjection: standard symfony folder. It consists primarily of the bundle configuration, occasionally CompilerPass.

/Entity: model classes managed by an ORM model and their repositories.

/Event: event classes.

/Listener: listeners and subscribers. It can contain subfolders, e.g. for Doctrine or Form listeners, etc. You would usually see a reversed directory structure. In other libraries Form/EventListener is more common than EventListener/Form. I’d recommend the latter in order to avoid spreading the EventListener directory within a single bundle. This also goes hand in hand with the guidelines, which refer to a small amount of directories and their generic nature.

/Form: form related class. It contains subfolders, in particular Form/Type and Form/DataTransformer.

/Form/Model: model classes that are not managed by ORM/ODM.

/Form/Type: Custom FormTypes.

/Resources: standard symfony folder.

/Tests: unit tests. The folder structure corresponds to the structure of the folders in the bundle, so if you want to test a class located in the Service/MyService.php, put the test case in the file Tests/Service/MyServiceTest.php.

/Twig/Extensions: classes that are extensions for Twig.

/Validator: validators.

Events.php : final class containing constants with the names of the events defined in the bundle.

The bundle structure is relatively easy to understand if you are used to Symfony2 systems. So is the database system. As you will need to get and change data in from the databases it's also important to understand how it works.

d. Doctrine

If you are familiar with Symfony2, you may have heard of Doctrine, one of the database systems of Symfony2. Below are some good practices and rules about Doctrine that we have to use on the project to keep everything unified.

Flush:

'Flush' only from the controller and commands. The rule is definite, mainly due to limit attempts to tamper with it, because although there are exceptions to the rule, they should be treated carefully.

Only a handful of services actually need to write data to the database. Preferably, they would only need to call 'persist' and wait for the flush until the flow returns to the controller. Sometimes it is necessary, for example, for us to have the 'id' before we go any further, however, in most cases, it can wait.

The advantage of adhering to this principle is that we do not have 'flushes' spread all over the different types of classes and we do not have to worry about handling transactions manually – 'flush' wraps its queries in the transaction by default.

Naming:

-Table names must be named in the singular, using lowercase letters and words separated by underscores '_'.

-Column names must be written with lowercase letters and words separated by underscores '_'.

QueryBuilder and results:

Queries must be constructed in a repository class using QueryBuilder. It is a good practice to separate construction of a query from its execution using separate functions for each part.

Query\Expr

Do not use the Query\Expr. Sometimes people abuse it. That causes the expressions to lose their readability. A simple expression is able to grow from a few to a dozen characters.

- good*: $qb->where('op.status IS NULL')*
- bad: *$qb->where($this->getQueryBuilder()->expr()->isNull('op.status'))*

Many to many

Avoid "Many to Many" jointures. Prefer "one to many" and "many to one" jointures by creating a new entity for that. It is recommended to add a createdAt and updatedAt fields into this entity.

The more you use these rules, the better. In that way the backend code will be using same patterns, same logic, and same nomination. And that goes even for the frontend code.

e. CSS:

To separate the design code (CSS), it's better to name its variables by the following ways:

- No underscores ("_"). Use dashes instead ("-").
- Don't repeat the actions names

Avoid classes globalization. The "body" should be hierarchically the first element.

The main layout includes all shared Css files and even the Jquery Css. Each category directly call its parents. It includes first the main css, then includes its own based css (ex: action, page, etc)

If your changes don't apply, don't forget to check if the main css file correctly includes your file:

```
{% block head_css %}
 {{ parent() }}
 {% stylesheets
    "bundles/nmfront/css/MACATEGORIE.less"
    filter="cssrewrite"
    output="css/MACATEGORIE.css"
    %}
    <link rel="stylesheet" href="{{ asset_url }}" />
 {% endstylesheets %}
```

{% endblock %}

f. Bundle description

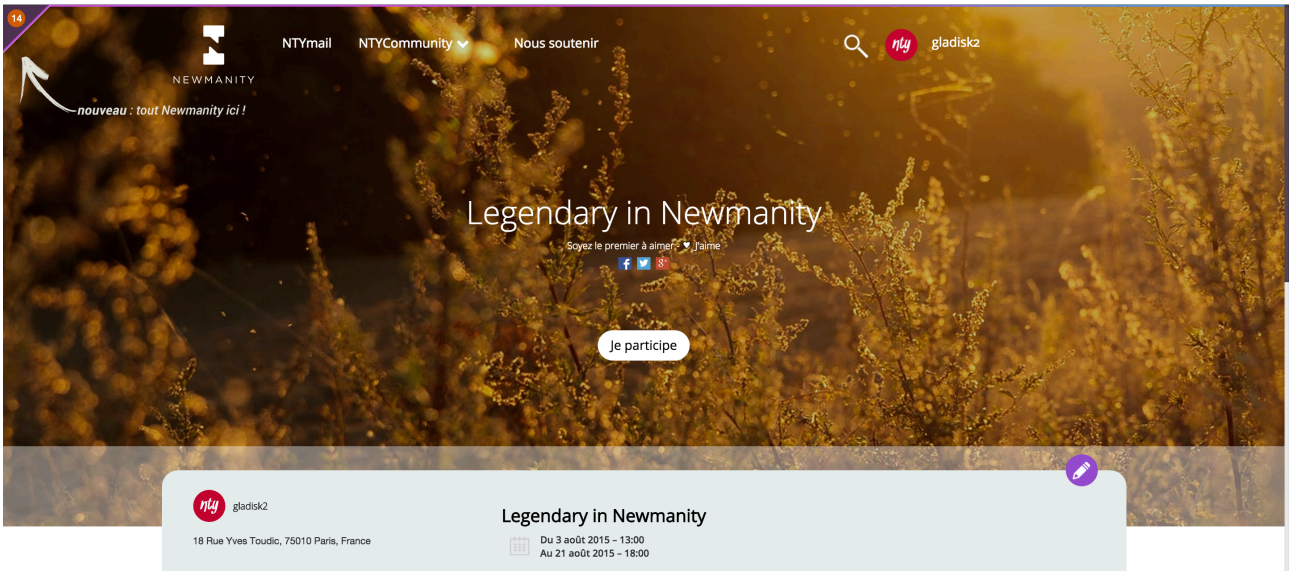
In Symfony2 (the used framework), all specific parts of the project are split in bundle folder. For instance, the admin part is managed in “AdminBundle”, the users in “UserBundle“ and so on.

All the external bundles are in the “Vendor” folder and the bundles made by the company in “src“ folder.

Below is a list and explication of all the bundles made by the company:

-**ActionBundle**: An action is an event created by an organization. It has four types: Event, Pledge, Collect, Petition and Volunteer (these types will be explained and illustrated later in the document).

The ActionBundle is the bundle that manages these actions. It contains the “create” and the “edit” form, that allows you to change information about an action you created or simply add a new one.



An action details pages

The “delete” action is managed by an “event listener”. When the delete button is clicked on an action, the “RemoveActionListener“ is triggered and manages the deletion of the action (see EventListener/removeActionListener.php).

Actions also contain images as background. These images are in the “image” folder. They can be uploaded while creating or editing an action. When an image is uploaded using the form, the “ImageUploaderListener” is triggered. It will first check if the uploaded image size is under 2mo, then will check the image format (the format must be jpg, gif or png), and finally upload it in image folder, and add the link to the database.

-**AdminBundle:** The AdminBundle manages the backend.

The backend allows you to modify static contents of the site (contact, partners and privacy policy). You can also change a user status and manage the company events. Usually the back office is used by the commercial and marketing teams.

The views are developed using AngularJs framework. Which makes the animations and the navigation between pages faster.

Each modifiable element has its own page and it’s own controller. You will find all these controllers in AdminBundle/Controller folder and the forms in AdminBundle/Form.

-**ApiBundle:** The apiBundle manages the api part of the website.

The mobile application uses requests to communicate with the website (fetching datas from database, adding comment to actions, editing post, etc). All these requests are developed in this bundle. The api bundle is the mobile application backend part.

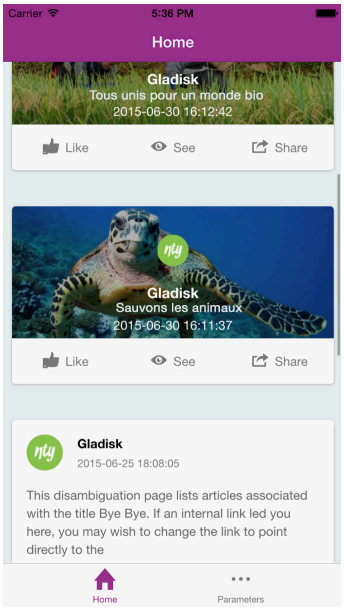
Each request starts by the string “api/”. The second word is the element managed and/or the folder, and the other words are parameters. For instance, “api/comment/post/1“ is a comment request, developed in “ApiBundle/Comment” folder, with parameters “post“ and “1“.

The Api requests can only be used by the application. All the request needed to be sent with a token to verify the user except the authentication request. Otherwise the security system blocks the request and send an error 400 with an error message.

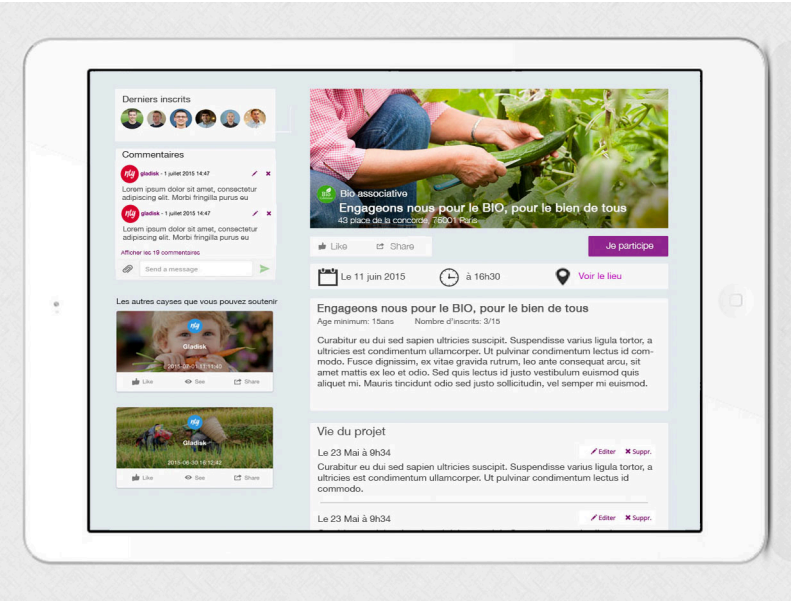
The security system is managed in ApiBundle/Security folder. You may need to directly ask the lead developer about how it works.

2. The Mobile application:

The mobile application is the mobile version of the Newmanity Community project. It has the same main elements (events, pledge, petition, etc), but displays them differently by using the mobile UI.



Home page on mobile



Action page on Ipad

a. Language and framework:

The mobile application uses Javascript language only, with AngularJS framework and Ionic/Cordova.

Ionic is the front end framework used with Cordova, and offers the possibility to add new navigation animation that looks exactly the same a s IOS and Android.

You first develop on your web browser and manage the application as a simple web application. Then you transform the code in Android and IOS and run it on a real mobile device.

b. How to install and use it on a mobile device:

Once you have tested your application, and are ready to install, follow the following steps to use it on a mobile device:

-Compile the application:

$ ionic platform add ios
$ ionic build ios

- Upload to Ionic platform

$ ionic login

$ ionic upload

- Download the application for test:

- Download “Ionic View” application on your mobile
- Create an account on the device
- Use the newmanity mobile ID to get the application (ask the lead developer for more details)

You can know run your application on your device, check the results of your work and test your api requests.

c. The mobile API

The mobile devices communicate with the web site by using the API services. All requests are made with AngularJS Http module.

Some of the services can be used anonymous, that means the user does not have to be authenticated to use them. However most of them are secured and needed authentication first. Below is the list of the current Api services and their description.

Anonymous services

The anonymous functions are all account related. They are the first actions a user can on the application launch.

Authentification (api/auths): Use this service to authenticate a user.

- *@POST username, password,*
- *@return JSON {token, language, preferences: {newsletter, crawler, partners}} code 201*
- *@Error 401 - User not found*

Create account (api/subscribes): Use this service to create new accounts

- *@POST username, password, account-type, language*
- *@return JSON {success: true}*

Secured services

The secured services are protected with an access token. This access token is set after the user connects. That means the user has to be connected to use the following services.

Refresh the access token (api/refresh): Use this service to refresh the token. The service launches automatically after a certain time (configurable in the config file of app)

- *@GET*
- *@return JSON {token: THE_TOKEN} - code 201*

Change user preferences (api/user/preferences): Use this service to set the user preferences.

- @POST newsletter, crawlers, partners
- @return {newsletter, crawlers, partners}

Get all the post comments (api/post/:postId/comments): Use this service to get all comments made for post

  - *@GET*
  - *@Return Comment Object*

Edit a comment (api/comment/:commentId/edit): Use this service to edit a comment

- *@POST {newMessage}*

- *@Return { "id": int, "oldMessage": string, "newMessage": string }*

Create a new comment (api/post/:postId/comment/new): Use this service to create a new comment

- *@POST {newMessage}*
- *@Return Comment Object*

Delete a comment (api/comment/:commentId/delete): Use this service to delete a comment

- *@GET*
- *return {success: true}*

Get home page elements (api/timeline/global): Use this service to get the actions and post to display on the user's home page.

- *@GET*
- *@Return Action objects and Post objects*

d. The routes

The routes are the links to use to change pages. They are all set in app.state.js. Below is the list of all the routes.

- Connexion page:
    - login : /login : The page that allow connexion
- Subscription steps and pages:
    - subscribe.terms : /subscribe/terms :
    - autoLogin : ^/autologin'
    - subscribe.terms: /subscribe/terms
    - subscribe.pseudo: /subscribe/
    - subscribe.password : /subscribe/password
    - subscribe.account-type: /subscribe/account-type
    - subscribe.loading: /subscribe/loading
- Home page and feed elements details:
    - app.home: /home,
    - app.post: /home/post/:postId

    - app.action: /home/action/:actionCategory/:actionId
- Parameters:
    - app.parameters : /parameters

e. The services modules and filters

AuthService:

The auth service module manages all the services relative to the authentication in order to simplify the process for the developer. It has some useful functions that you may want to use.

Login(username, passord): This function makes a request by the API to authenticate the current using the “username“ and the “password“ given as parameters. It returns NULL on failure.

getCurrentUser(): This function gets the information about the current authenticated user. All these information are stocked in local storage during the first login.

refresh(): This function requests a new access token to replace the old one. This operation has to be done regularly, to keep the mobile application secured.

LoggerService:

This service module is one of the most used one. It has all the function to display alert, confirm, and prompt dialogue.

warnDialog(dialogTitle, dialogMsg): Displays an alert dialog

confirmDialog(dialogTitle, dialogMsg): Displays a confirm dialog

customDialog(dialogTitle, dialogTemplate): Displays a custom dialog window based on the given HTML template.

HumanizedDate filter:

Humanizeddate is used to transform date in user friendly text form. For instance 24/12/2015 will become “3 days ago“. It’s used to display action or post creation dates.

Intership report

[Part 2]

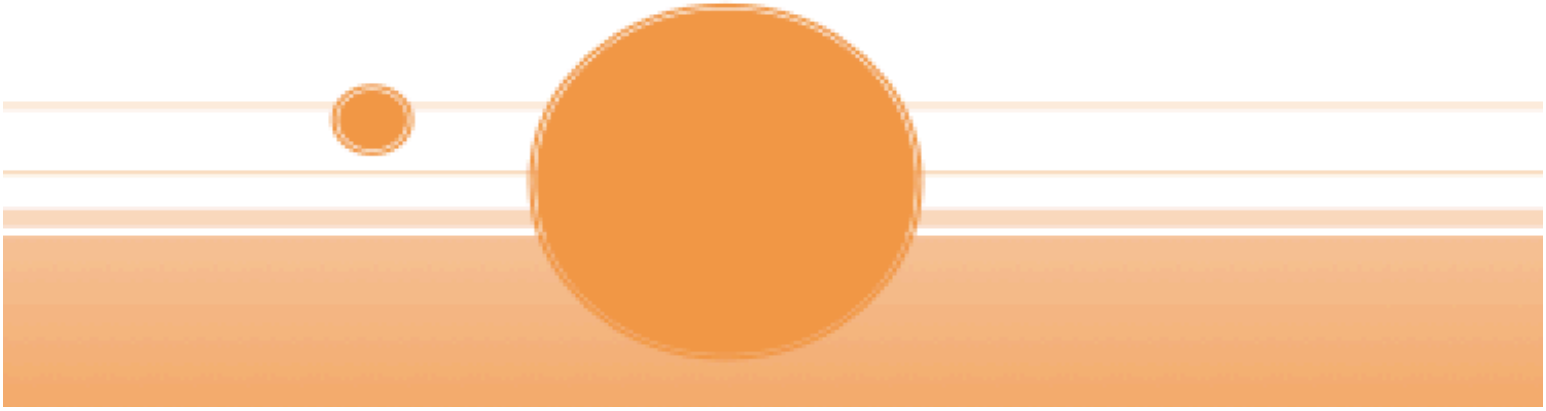
Reasons to hire the trainee another time (for a technical superior)

Issaka GALADIMA-IBRAHIM

Galadi_i

Society: Next-W

Internship period: 01/04/2015 – 30/09/2015



Summary

A. A well done internship

B. The most efficient solution

C. A good asset for the team

For my internship, I've been working at Next W. In which I've been working on the society website and the society mobile app.

Below are the reasons to hire me again to continue the society mobile.

A. A successful internship

Since my arrival in the society I was interested in how everything work. What is the hierarchy in the company, who had which post, what is everyone's job about, etc. That is how I manage to quickly integrate the developer team.

I spent my first week reading the entire code, and since the second week I was already finishing some tasks and functionalities. That means I learn quickly.

After two month of development on the website project, the mobile project was attributed to me. I made the expected Beta version before the end of my internship, and was adding more and more functionalities.

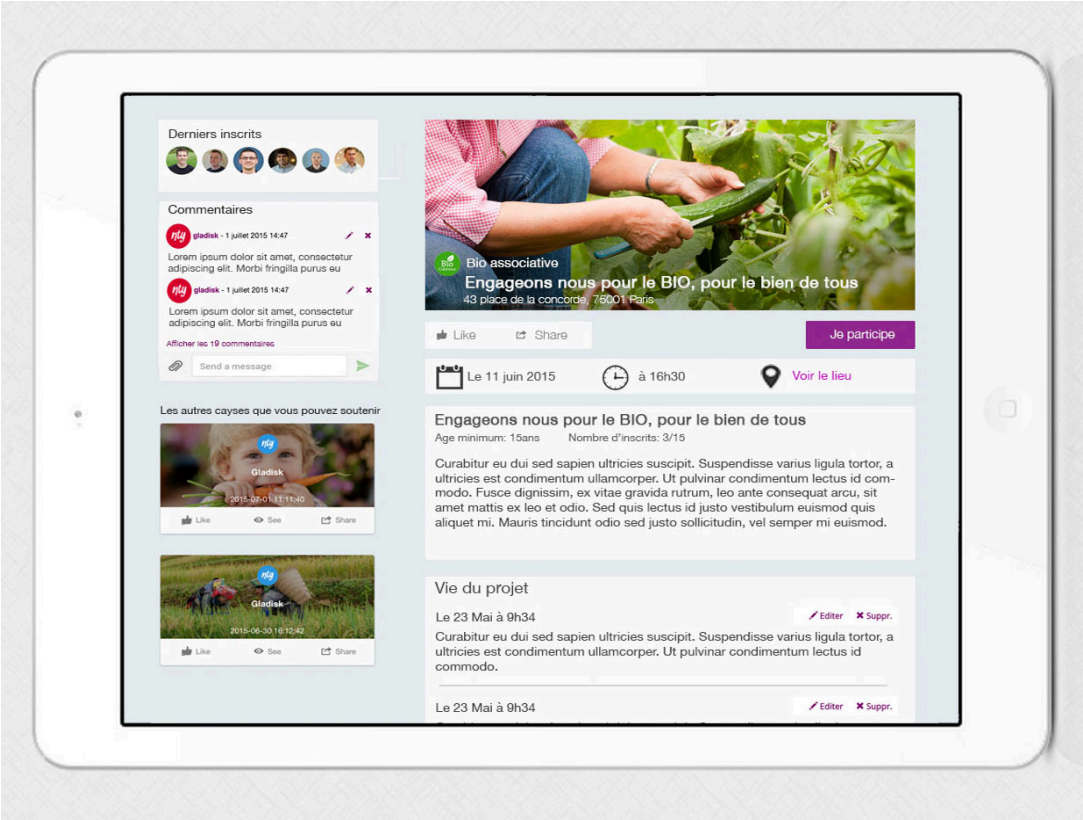
B. The most efficient solution for the mobile project

As I have been working on the project since it started, and made the most part, I know everything about its code. That means I will be speeder than any other developer to know in witch files we have to make modification for adding a new functionalities.
Hiring me again to continue the project would be the most efficient choice for quick and immediate results.

C. A good asset for the team:

As I said before, I made all the mobile project graphics and mockups, in an ergonomic, simple and user-friendly way.
I was always been giving ideas to improve the project (like using

HumanizedDate as an angular filter to display dates, storing loaded action and post in a Timeline service, using the ApiSecureKey for more security, etc), because I could understand the needs of the customers and how to integrate them in the project. By giving me other projects to do you are sure to get ergonomic applications.



Ipad version – Action pages

Conclusion

In conclusion, I am familiar with the project and the customers need. I made the code structure, the mockups, and even managed the security and the API part. That is a proof that I am conscious, professional, proactive and polyvalent.

To continue the mobile project, giving its responsibility to me would be the better choice for quick results and ergonomic pages.

Intership report

[Part 3]

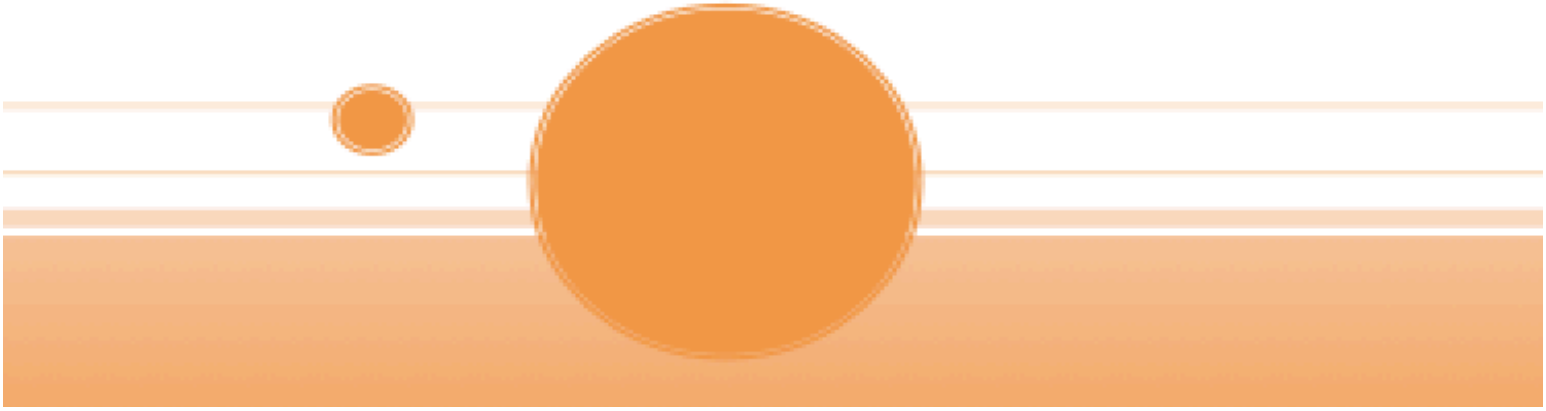
Reasons to hire the trainee another time (for a non-technical superior)

Issaka GALADIMA-IBRAHIM

Galadi_i

Society: Next-W

Internship period: 01/04/2015 – 30/09/2015



Summary

During my internship period I worked on the society web site Next W. It gave me more programming experience. Besides this internship showed me how things works in a small society, not only in IT domains but in all other aspects (commercial, marketing, etc…).

Below are the reasons to hire me again in Next W.

A. Issaka GALADIMA : A serious and efficient programmer.

I was always been interested in IT domains. So programming is not a boring work for me. Besides I like theoretical challenges, I never hesitate to search for answers by my own, so that I will never be a trainee that will need to be followed or helped step by step. I deliver quickly well done work.

Besides, I quickly get used of the used tools on a project. For instance before the beginning of the internship, I had never make a project using “Angular” and not so much with Symfony2 system (the main IT tool and functions used on the web site). But in only two weeks, I learnt how to use it. So that at a moment the senior programmer didn’t need to verify what I upload on the server.

I don’t have difficulty to integrate new working environments.

My will is the key of my success in every project.

B. Will of success

As I said before, when I am concentrated on a project, I never see it as a boring work. Therefore if there is work to do, I just do it, whatever difficult it is.

And when there is timelines I never hesitate to stay more time at work if necessary. I can also work during the weekend and during my free time on

web site functions or pages to implement if it permits to the team to respect its timelines for any commercial event or product delivery.

To enhance the team productivity I work sometimes in other domains according to my skills (graphic for example).

So when I haven't works to do immediately, it's not rare that I work on other aspects of the project, doing illustrations for instance.

C. A polyvalent programmer.

I have previously listed my qualities as a programmer. Now I will talk about my other skills, which is one of the web site creation bases: **graphics**.

I've always loved drawing. It really becomes a passion. I've learned to use graphics softwares like Photoshop, Illustrator, CorelPaint, Vicman Studio and also graphics pad for making illustrations, web sites graphic model, etc…

Soft programming is a domain that is linked to multimedia. All softs or web sites need an appropriate graphic style and a logo. As I have graphics competences I can also make logos and models for projects.

Having a programmer/graphic artist in a team is a precious asset for the society.

D. Conclusion

In conclusion:

-I have technical Skills and competences required to be a good programmer.

-I am autonomous and learn speedily. So I will never waste other team member's productivity time.

-I am also a graphic artist. So my polyvalence permits me to work on two sides.

Besides I have already been in the team, so I know how the works are organized. And I am used to the server side files organizations and systems.

These elements are the reasons to hire me to continue on the project or to work on another new one.