Fun With PhP and Kids

Ada Lungu

January 25, 2017

Contents

Chapter 1

Introduction

1.1 Introduction

1.2 Company Description

This is a project done in collaboratino with **SmartFun** . **SmartFun** is a small company, a start up that outset its activity in 2016. The company's vision is to come up with a friendly and holistic environment for children where they practice STEM (Science, Technology, Engineering, Mathematics) using interactive and playful methods.

They create an learning environment by providing workshops, where kids are involved in hands-on projects in coding and electronics. Children experiment with building robots, motors, using sensors and coding these devices. They learn a lot empirically by trial and error.

Part of solution they envision in creating a holistic learning experience is an online platform that will incorporate the knowledge transfer and learning materials of the actual workshops. We have already started to do that migration of information towards the platform. At the moment we are using the beta version of the platform as a helping material in our workshops, until step by step it will replace the need of tutors and running the workshops in a traditional way.

Workshops constitute the perfect testing environment for our application, as we receive constant feedback by observing kids interaction with it. Using them as a testing grounf for the platform assured the selection and prioritising of the developed features based of the users reactions.

1.3 Problem Analysis

Problem Statement/Challenge

Market Analysis

Preliminary Work - gathering requirements + writing specs http://www.joelonsoftware.com/articles/WhatT
- mock-ups

Chapter 2

The User Experience

2.1 The Target Audience/End-Users Of the Application

The main end-users of our product are children with ages between 8 and 15 years old. Part of our development strategy was a continuous testing of the platform with representatives of this group age. We have set our testing environment through the workshops in which we have been assisting as tutors. We have desiged the platform iteratively based on continuous feedback from our focus groups. We have redesigned features in accordance with free observations of users interacting with our product and with direct feedback from them.

The other end-users group of the platform are administrators, which at the moment are represented by the tutors of the workshops. By playing both the role of a tutor and the developer of the application, I had the chance in directly testing the usability myself in the real life context of the product.

Consequently, the application has two modes for the two target groups. The profile and the needs of two types of end-users appeal for quite different presentation styles. We focused in designing a user friendly interface for the main target which are the kids, and invested less effort in polishing the administrator panel.

As a central user experience convention, we have kept a minimalistic design for the whole app, both as an aesthetic principle and as a strategy for further future graphics integration. In the next subsections of the chapter I will briefly describe what are the main features of each end-users mode.

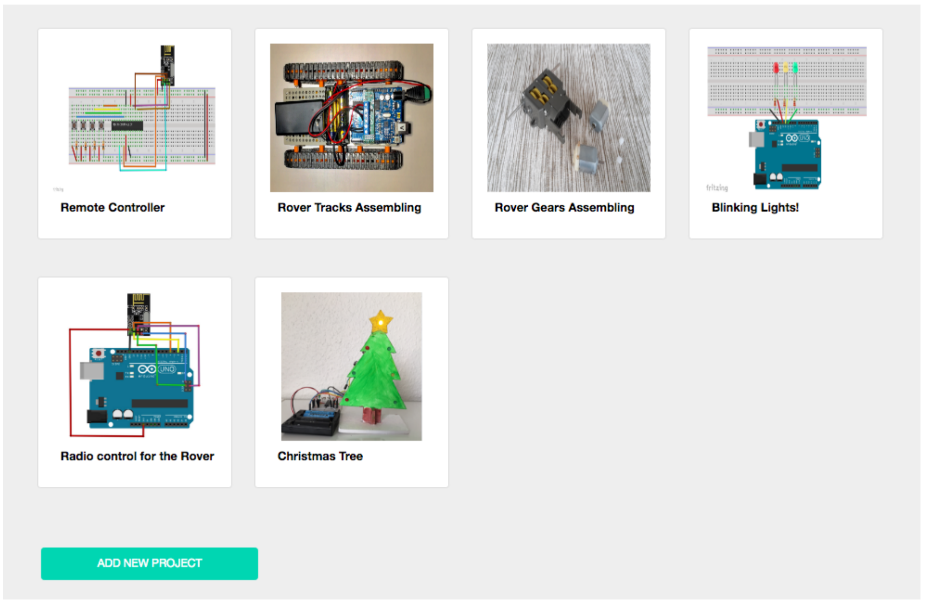


Figure 2.1: Display of projects.

2.2 Typical Usecase for Kids

2.2.1 Learning by doing: Projects

Projects Section is a central component of the learning experience on the platform. The main page of the module displays all projects to users and a searching feature for projects by labeled categories. Projects are briefly presented by a distinctive image and their title.

Users can create and edit a project. The editing mode is currently open to administrators of the platform for putting up and editing teaching materials related to the project: title, description, media resources, a list of needed materials. This mode supports uploading of files (pdf), images (png, jpeg) and videos. Kids can create projects as well, so that the users become active contributers to the platform. To ensure appropriate content, the data submitted by kids await check-ups and approval from the administrators before getting uploaded on the website. In the end we want to create an experience where the user is also author and contributor of the platform.

In the project presentation mode, the user finds all details about the current selected project: description, label tags, needed materials or items list (estimated time of completion). The materials list come along with an order and purchase system. Users can check the items they already have and can add to basket for purchase the ones they need. Users can leave comments on the page,

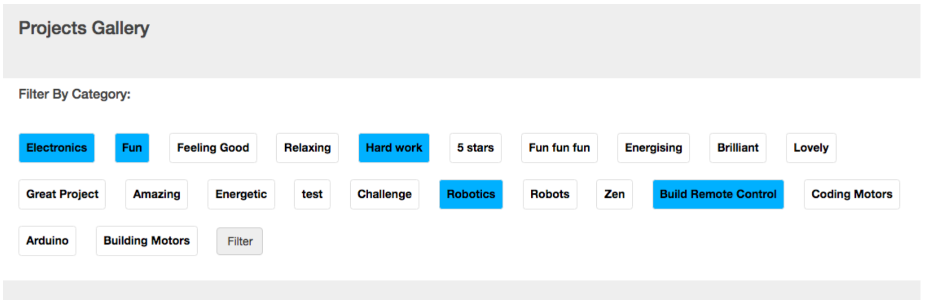
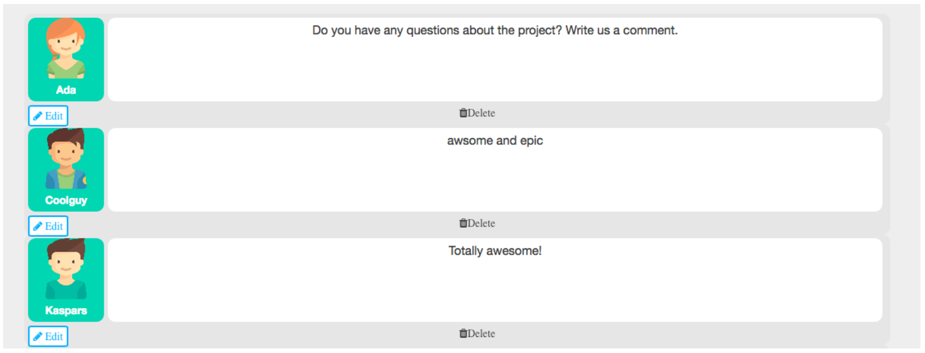


Figure 2.2: Labels.



Ada: Do you have any questions about the project? Write us a comment.

Coolguy: awsome and epic

Kaspars: Totally awesome!

Figure 2.3: Comments.

with questions or any information related to the project. The page includes a stars voting system, where they can rate the project based on a few criterias: functionality, difficulty, fun. (Users will be permitted to vote only after completing the project.)

Each project is split in a number of steps that the user must complete, in order to get rewards such as points and achievements. The user is able to visualize the progress on the tracking progress bar at the top of the steps view. [Figure 2.6]

2.2.2 Feedback: Gamification, Rankings, Achievements

The application is built around a motivational reward system. It uses gamification techniques to transform learning into an engaging and provocative activity.

The reward system is represented by achievements and a points system. Achievements have two states: locked and unlocked. They are split in categories, each category standing for certain types of activitis and tasks that the participants can perform. Initially all achievements are locked and they can be

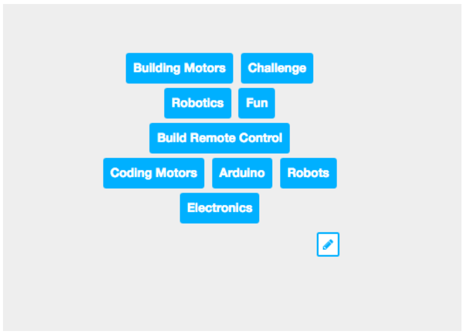


Figure 2.4: Add Labels.

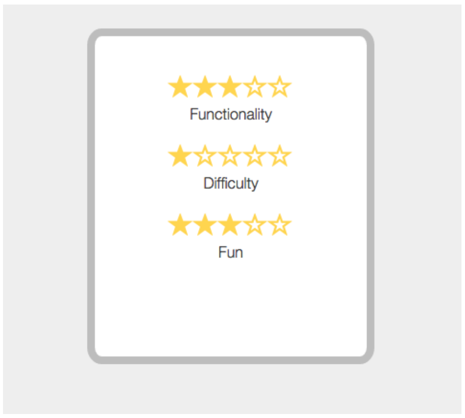
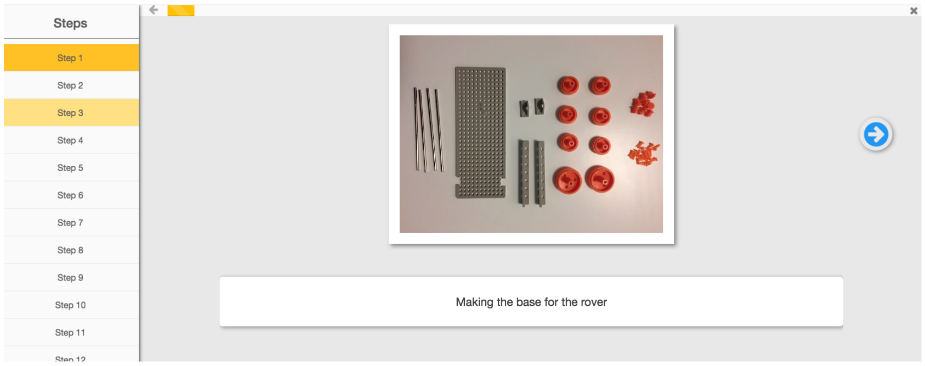
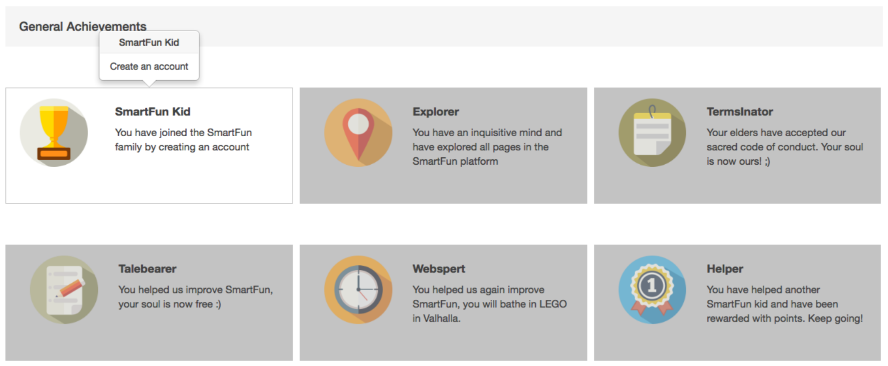


Figure 2.5: Stars Voting System.



Making the base for the rover

Figure 2.6: Project Steps.


General Achievements

SmartFun Kid

Create an account

SmartFun Kid
You have joined the SmartFun family by creating an account

Explorer
You have an inquisitive mind and have explored all pages in the SmartFun platform

TermsInator
Your elders have accepted our sacred code of conduct. Your soul is now ours! ;)

Talebearer
You helped us improve SmartFun, your soul is now free :)

Webspert
You helped us again improve SmartFun, you will bathe in LEGO in Valhalla.

Helper
You have helped another SmartFun kid and have been rewarded with points. Keep going!

Figure 2.7: General Achievements.

activated only when users perform specific actions. The locked state of achievements is visually represented by a grey overlay or background on the achievements (depending on the context where they are displayed). The unlocking of an achievement is marked by a pop up. The achievements have a general presentation page and can be found also on the user dashboard, where only a maximum of 6 achievements are displayed (always a few unlocked for challenging the user).

In the current beta version, there are 2 functional categories of achievements: the the Quiz Achievemnts and the General Achievemnts. The achievements are closely linked to the points system, each coming with a number of points as a reward.

The points system is linked to the rankings section which enhance a competitive experience for the users. There are 3 types of rankings: *general rankings* - rank individual achievements comparing the performance of all active users; *workshop rankings* - list individual achievements between members of same workshop; *groups rankings* - evaluate collective performance per workshops based on the average points gained by their members.

We have intended to design the rankings in a way that stimulates both an individual and collective competitiveness and performance. The rankings per groups targets to boost children spirit of collaboration and feeling of group adherance.

During our UX testing sessions with the kids, we have concluded that rankings is the most popular feature of the platform, as kids kept coming back and checking the rankings page many times per session. Rankings was motivating kids in taking more and more quizzes and redoing quizzes for gaining more points. Based on our UX testing findings, we include on the platform a number

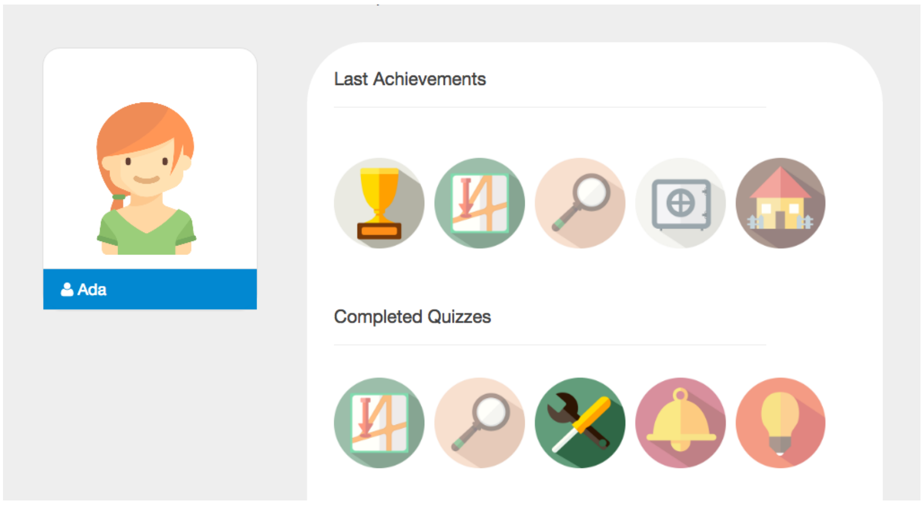
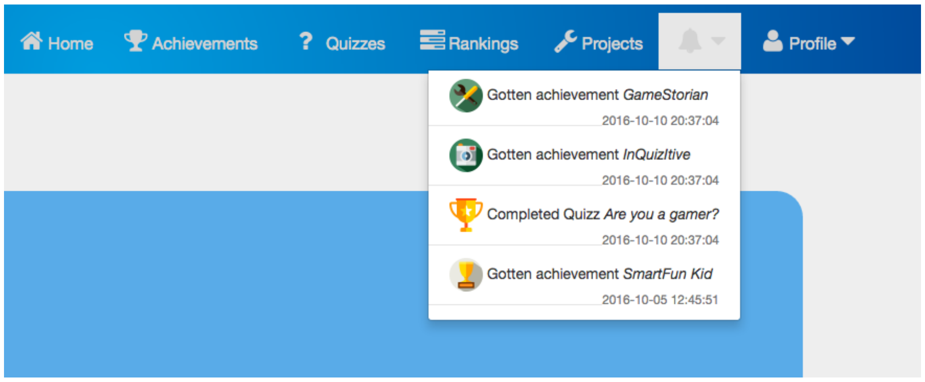
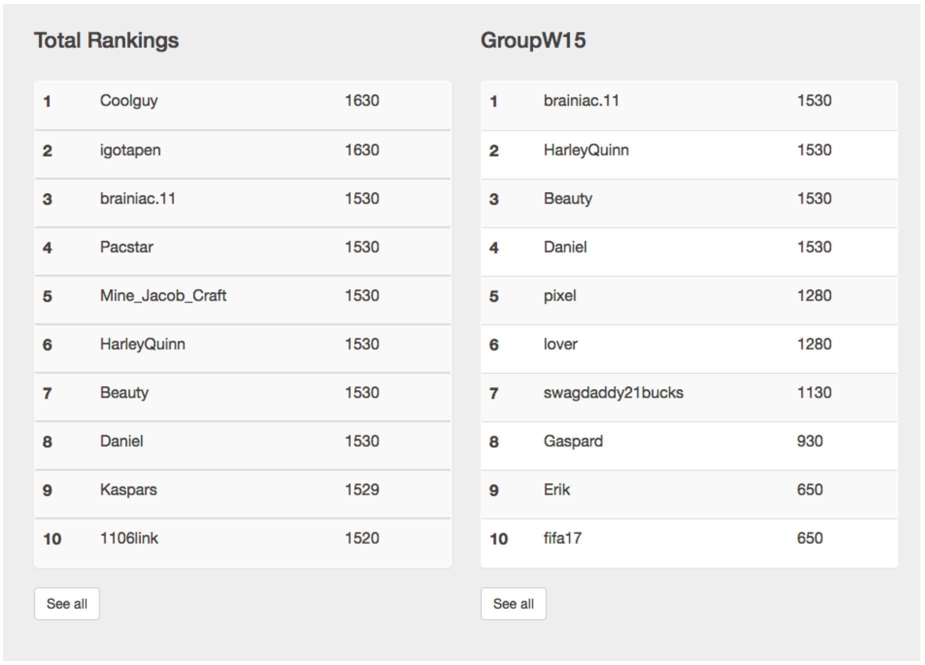


Figure 2.8: Achievements Dashboard.



Gotten achievement *GameStorian*
2016-10-10 20:37:04
Gotten achievement *InQuizItive*
2016-10-10 20:37:04
Completed Quizz *Are you a gamer?*
2016-10-10 20:37:04
Gotten achievement *SmartFun Kid*
2016-10-05 12:45:51

Figure 2.9: Achievements PopUps.



Total Rankings

1	Coolguy	1630
2	igotapen	1630
3	brainiac.11	1530
4	Pacstar	1530
5	Mine_Jacob_Craft	1530
6	HarleyQuinn	1530
7	Beauty	1530
8	Daniel	1530
9	Kaspars	1529
10	1106link	1520

GroupW15

1	brainiac.11	1530
2	HarleyQuinn	1530
3	Beauty	1530
4	Daniel	1530
5	pixel	1280
6	lover	1280
7	swagdaddy21bucks	1130
8	Gaspard	930
9	Erik	650
10	fifa17	650

Figure 2.10: Rankings.

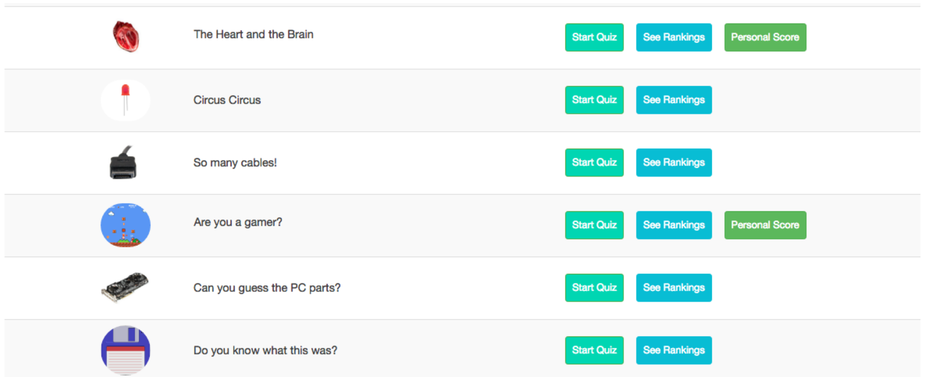
of occasions for users to access the rankings: when first logging in on dashboard page, after the completion of a quiz.

A central gamification feature intended for the release version will integrate and link together the current gamification elements (achievements and points) by introducing an avatar buiding system. Users will build their avatars trading their achievements and points for items that will constitute parts or accesories of the avatar. We take into consideration adopting a monetary system represented by coins, with an equivalent of points for one coin.

2.2.3 Assessment: Quizzes

The core feature of the current version, that generates points and unlock achievements are quizzes. Quizzes are linked to workshop themes and to projects and assess users level of mastering a topic. They have multiple choise answers with one or more correct answers. After completing a quiz, users can evaluate their answers and visualize the wrong answers against the correct ones.

Users are evaluated by the number of correct answers and by the time of quiz completion. They can redo the quiz to upgrade the completion time, but they can't update their points, which are decided by their first quiz completion.



	The Heart and the Brain	Start Quiz · See Rankings · Personal Score
	Circus Circus	Start Quiz · See Rankings
	So many cables!	Start Quiz · See Rankings
	Are you a gamer?	Start Quiz · See Rankings · Personal Score
	Can you guess the PC parts?	Start Quiz · See Rankings
	Do you know what this was?	Start Quiz · See Rankings

Figure 2.11: Quizzes.

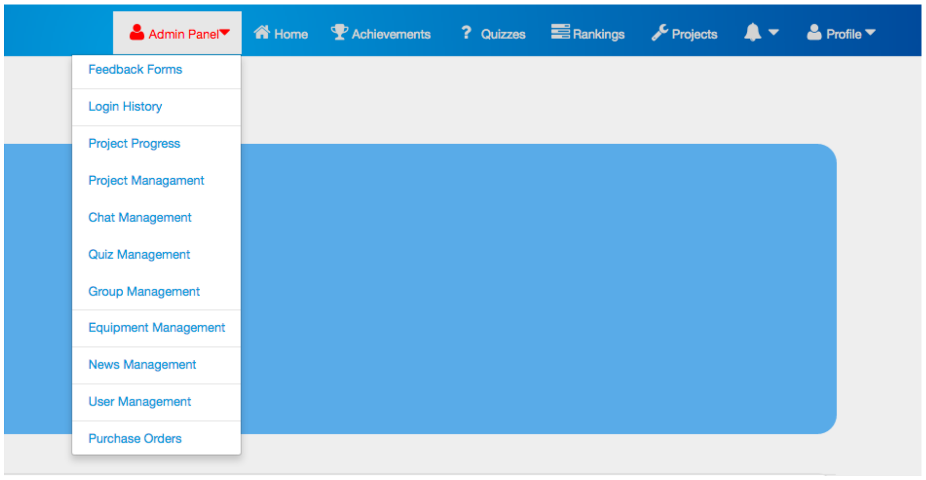


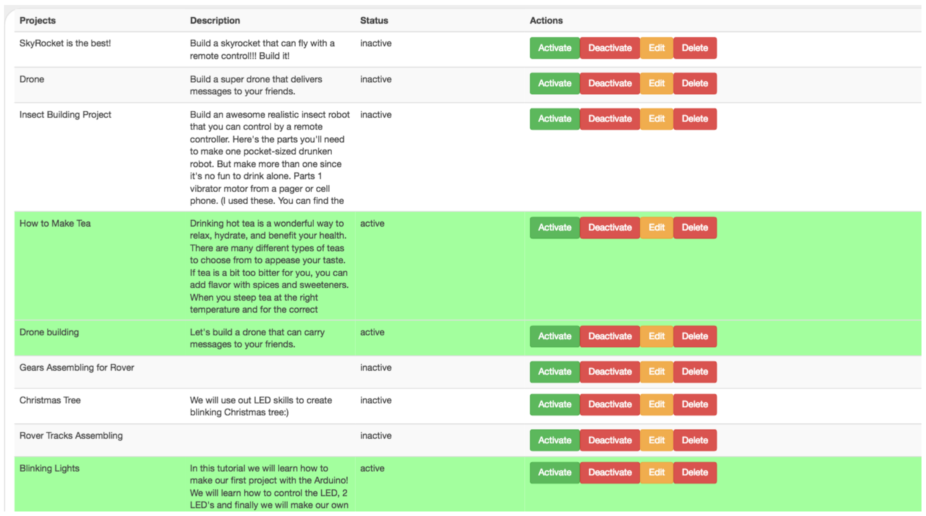
Figure 2.12: Admin Panel Menu.

In the future release version, quizzes will be locked by default and will be unlocked only in a logical sequence.

2.2.4 Typical Usecase For Admin Panel

We have designed the administration panel system in such a way that admins can manage most of the platform content and monitor users activity. From the admin panel, administrators can add and edit content displayed on the platform (quizzes, projects), activate or disable components (the chat system), view and manage users profiles and activity (login history, reset passwords).

Each main section on the platform has a correspondent management control in the admin panel.



Projects	Description	Status	Actions
SkyRocket is the best!	Build a skyrocket that can fly with a remote control!!! Build it!	inactive	Activate Deactivate Edit Delete
Drone	Build a super drone that delivers messages to your friends.	inactive	Activate Deactivate Edit Delete
Insect Building Project	Build an awesome realistic insect robot that you can control by a remote controller. Here's the parts you'll need to make one pocket-sized drunken robot. But make more than one since it's no fun to drink alone. Parts 1 vibrator motor from a pager or cell phone. (I used these. You can find the	inactive	Activate Deactivate Edit Delete
How to Make Tea	Drinking hot tea is a wonderful way to relax, hydrate, and benefit your health. There are many different types of teas to choose from to appease your taste. If tea is a bit too bitter for you, you can add flavor with spices and sweeteners. When you steep tea at the right temperature and for the correct	active	Activate Deactivate Edit Delete
Drone building	Let's build a drone that can carry messages to your friends.	active	Activate Deactivate Edit Delete
Gears Assembling for Rover		inactive	Activate Deactivate Edit Delete
Christmas Tree	We will use out LED skills to create blinking Christmas tree:)	inactive	Activate Deactivate Edit Delete
Rover Tracks Assembling		inactive	Activate Deactivate Edit Delete
Blinking Lights	In this tutorial we will learn how to make our first project with the Arduino! We will learn how to control the LED, 2 LED's and finally we will make our own	active	Activate Deactivate Edit Delete

Figure 2.13: Admin Panel Projects.

In the Projects Management section, the admins have an overview of all projects, can edit the display data of the projects or can delete projects from the system. Projects can be disabled or turned on, meaning admins choose which projects are currently displayed on the projects page. Project Management panel includes an users progress tracking feature, which gives admins an overview of the projects started by users and their level of completion progress.

The Quiz Management system allows adding or editing quizzes: uploading images and creating multiple choice answers, setting the correct answer. As it does in projects, the admin system allows the enabling or disabling of quizzes.

The profiles and activity of users can be monitored and managed in the User Management panel. The Login History comprises the timestamps of the users having logged in on the application. The Reset Password feature came as a result of a common problem tutors faced in the actual workshops: kids frequently forgetting their credentials. The Group Management is also linked to the real needs of the workshop organisation. We want facilitate the tutors in changing names or adding new workshops, without the need of the developer changing entried in database.

In the Chat Control panel, tutors can disable the chat. We have designed this feature, after chatting during the workshops became a way of distracting kids from working on the projects and paying attention.

Chapter 3

Implementation

3.1 Working Environment Settings

We use a server host provider service (Meebox), which comes with a package that supports php development, offers access to Mysql version XXX, and includes phpMyAdmin as database management tool. On top of these it provides SSH Access and SSL Management.

I have suggested Github as our Version Control System, and learned Slack as the company's favorite Communication Management Tool. We have created a separate environment for testing and for production and for this purpose we have created a database for each environment. All the local changes are tested on the test DB which on the server.

I have personally chosen as my IDE (Integrated Development Environment) the Jetbrains product for php development PHPStorm.

3.2 Front-End

We have opted for a minimalistic, simple design for our beta version, as a strategy to iteratively develop it while testing its usability with end-users. This approach facilitates later integration of graphics. The trademarks of our front-end user interface design are: responsiveness, minimalist and intuitive, kids-friendly.

3.2.1 Programming Languages

On front-end I mostly use with Javascript' Jquery libraries. Apart from substantially compressing the code in fewer lines, JQuery is also the most broswer-friendly. It does not need Adobe Flash plug-in to be interpreted by the browser, meaning it is readable in any browser. (http://www.javaworld.com/article/2078613/java-web-development/6-reasons-you-should-be-using-jquery.html)

I use SCSS as a preprocessor programming language, to facilitate better standardization and omogenity of UI components. Jetbrains PHPStorm comes with a File Watcher, a transpiler tool for SCSS that facilitated its integration.

As a working practice I keep the mark-up language separated from Javascript as much as possible. I use small modules of Javascript code, usually a Javascript file corresponding to an UI component. The application also has a dedicated layout folder, that comprises the layout elements common to all or some of the application pages.

3.2.2 UI Framework

The front-end framework we use is Bootstrap Twitter 3.3.6. We have settled on Boostrap after analysing pros and cons of alternative frameworks. Although frameworks like Foundation, Semantic UI, Material Design, Ulkit have some competitive features, Boostrap won by having some unbeatable advantages.

It corresponds to one of our core design principles: responsiveness. It exceeds other frameworks by its stated attention towards resposiveness and mobile-friendliness. Bootstrap is still regarded as the most popular front-end framework, with large community support and consequenlty with rich documentation and resources: articles and tutorials, third-party plug-ins and extensions, theme builders.

I also prefer Boostrap beacause of its level of specificity. Being more generic than other frameworks that tend to have higher-level specificity, its minimal style framework is easier to customise. I consider it is more practical to add up styling rules than overwriting existing ones. (Ref: https://www.sitepoint.com/5-most-popular-frontend-frameworks-compared/)

A good framework needs to level up constantly with the latest web technologies, especially with regards to mobile. Bootsrap does that, being constantly under active development. It also has a generous browser support.

Some of the other frameworks have some very competitive features. Semantic UI has some extra unique UI components and outperforms Boostrap as a very friendly, semantic language. Its overall structure of the framework and the naming conventions, in terms of clear logic and semantics of its classes has a clear vantage point. It prounds itself with a scalable and modular architecture for CSS as main property.

Frameworks such as Semantic UI and Foundation also outweight others by a unique, rich pallette of UI components. YOOtheme distinguishes itself with a better GUI Customizer. It offers a flexible and powerful customization mechanism, either manually or via its GUI customizer. Regarding the spectrum of browser support, Boostrap finds itself a step behind Foundation.

3.3 Back-End

3.3.1 Programming language

I have built the application using the scripting language PHP as server-side choice. I believe it is a reliable choice, as PHP has steadily been one of the most widely used developing language for web applications. It is commonly known for its portability and it is supported by the highest number of web server hosting providers. PHP is widely backed by large programming communities and it is open source, which assures a steady development and a solid community commitment in stabilizing it.

PHP is also known for its flexibility/resilience. It it portable on all major operating systems, can function both as a module or a CGI processor and has support on a wide range of web servers. PHP holds up both procedural and object-oriented (OOP) programming paradigms, and has compatibility with a vast collection of databases.

For relational database management system (RDMS) I have opted for MySql. PHP can use mainstream MySQLLi and also provides an abstraction layer through PDO, which I have made use of.

3.3.2 RDMS

PDO stands for PHP Data Objects. It provides a data-access abstraction layer, using a unified API, which is a lean, consistent way to access databases regardless of the database. This increases the portability of our code, if we want to change the database we operate with.

One core advantage of PDO over MySQLi lyes in its database driver support. PDO upholds around 12 different drivers, opposed to MySQLi, which supports MySQL only. This fact has implications for further software changes, as using PDO makes it facile to transfer to a different database.

The abstraction layer PHP offers in interaction with the database through PDO optimises the program execution and maximises security measurements. The performance is boosted by the use of Prepared Statements, which allows the parsing and execution of a query only once. The queries are sent only once to the database, where they wait for the parameters. The bandwidth to the server is minimalized through the use of Binding Parameters, which are sent independently to the query waiting on the db.

The second advantage of employing PDO is the layer of security it provides. It is designed to shield against one of the most invasive forms of security breaches: SQL injections. The parameters sent in the query are transmitted later than the query commands using a different protocol. They are scrunitized with security filters through these protocols.

In the design of our database, we have used all types of relational raports between tables. One-to-one relationship model is predominent, followed by one-to-many pattern. The many-to-many relationship is used in merging data between tables such Members-Projects, Projects-Steps, Members-Achievements, Members-Activities, Members-Quizzes, Members-Comments. I describe the database design using the following Entity Relation Diagram. I have used a coloristic relation map to differentiate the types of dependency between tables and data. [Sceenshot DB Entity-Relation]

3.3.3 Security

3.4 Application Architecture and Design Principles

I will exemplify the application architecture and the design principles by presenting the interactions of one of the projects main modules: the Projects section. The following diagram shows the organization and dependencies between the files of the module. [Figure 3.2]

3.4.1 Design Principles

I have designed my code around few pillarstone principles: separation of concerns, DRY (Do Not Repeat Yourself) and modularity (reusable components). Separation of concern principle is commonly associated with the MVC architecture design pattern. Although I do not employ a standard MVC framework, I have structured the composition of the code by applying MVC patterns, by striving to preserve a neat separation between the business logic and the view.

On Client-side I have separated the plain presentation from functionality, by keeping the the mark-up language from javascript in separate files. For readability purposes, I structured the front-end code in smaller separate files, so any major UI component is represented by a distinct file.

Because I make frequent use of Ajax calls, steadilyI have decided to dedicate a separate file where I have modelled an ajax API, that defines all ajax calls functions used throughout Projects (ajaxProjectsAPI.js). Doing so, I gain

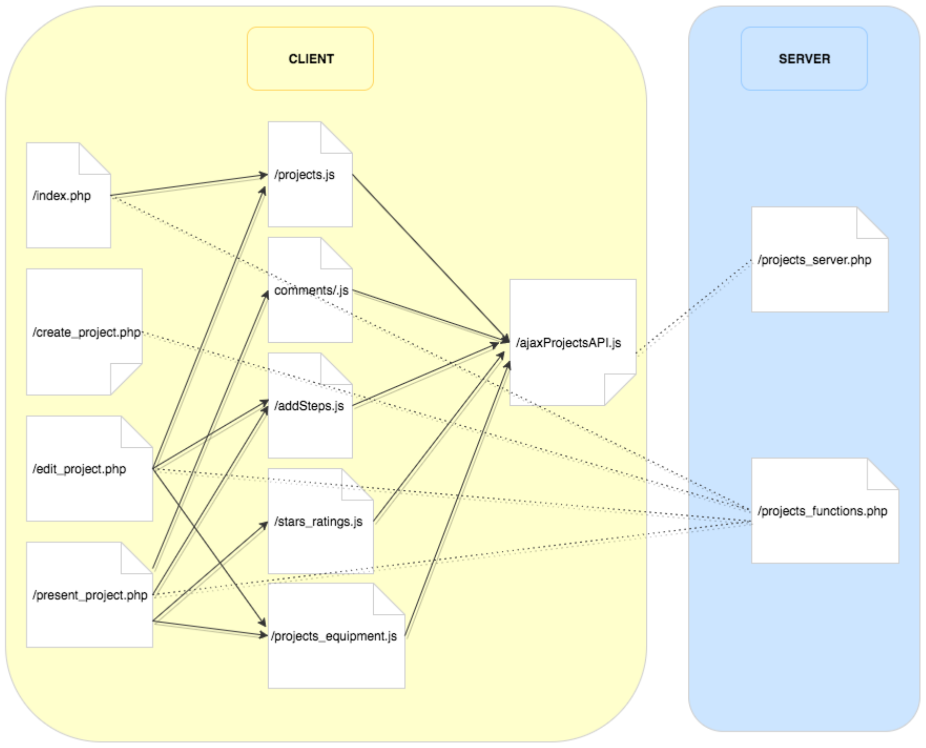


Figure 3.1: File Dependencies Projects.

readibility of the code, by having a javascript file that only deals with the actual ajax calls, and the other javascript file only displaying the data in the view. The next advantage it promotes is modularity. Having a universal ajax API for the Projects module, I centralize all the ajax calls under reusable functions, that are accessible from any file of the module and even to other modules. This way, I can use same ajax function when needed and forestall redundancy of code. For example,(as shown in the diagram) the ajaxUploadFile function is called from different js files of the same module/folder, or can be called from other modules, anywhere in the system.(or change example)

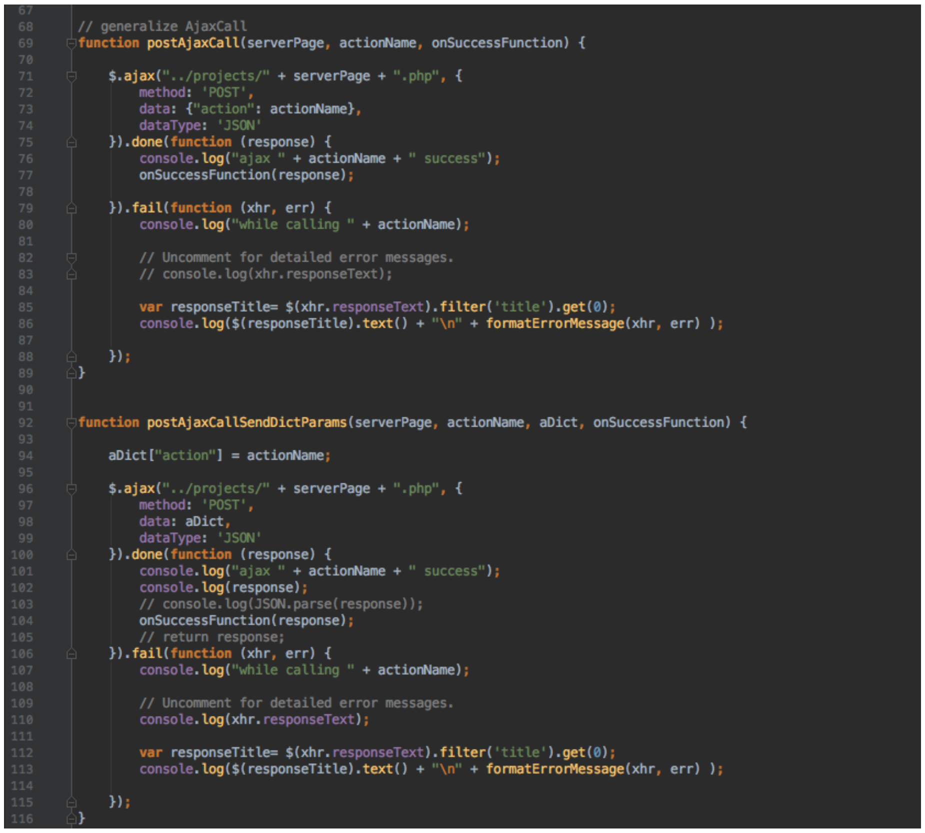
I'll describe the design of the AJAX API. It has 2 central generalized Ajax calls functions: postAjaxCall and postAjaxCallSendDictParams. postAjaxCall function sends as data only one parameter, while the other takes a dictionary for sending a couple of arguments to the server. Both functions take the following arguments: the server-side endpoint they are sending data to, the data to be sent (name of the action and parameters), and a function that will be executed (in javascript) if the ajax call has been successful. [See Figure XX]

All the other functions of the Ajax API are calling one of the 2 generalized functions, depending on the amount of data they send. I have payed attention at functions naming conventions, as part of my clean code effort strategy. Consequently, these functions are suggestively named by the type of action they perform on the back-end, their name having in composition the name of action parameter sent as data.[See Figure XX]

The Ajax API is for my application what the Controller is for the MVC pattern. It acts as an intermediary between the server-side(business logic) and the front-end(view). Basically, it solicits data from the server, takes the result data and transfers it to the view, without processing it in any way.

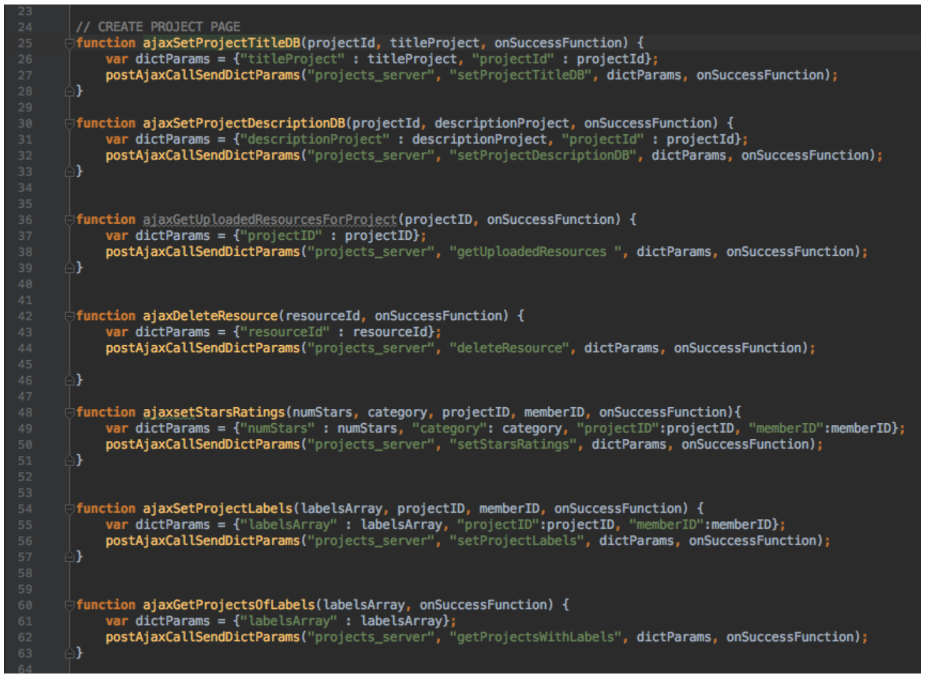
Having the Ajax calls isolated, the other javascript files are left to perform the display of data. I encapsulate the display mechanism in functions. I have created distinctive functions for each display demand of data in the interface. By doing so, display functions can be reused by other functions and the readibility of the code is increased. Moreover, it increases modularity and it facilitates easy future changes or integration of new components in the UI. Commonly, a javascript function in the system will call one or more ajax functions and will pass it as argument a display function, like in the next snippet example.

On Server-side, I have organised the API in 2 main files: projects_functions.php and projects_server.php. In projects_server.php are all functions and the database requests/queries called through the Ajax API that are sent to the Data Server, while in projects_functions all the other functions, associated with the Web Server requests. Besides the two endpoints, I separate the back-end functions for self-contained actions in independent files, like uploading media resources in upload.php, adding the steps for the project in addSteps.php and uploading



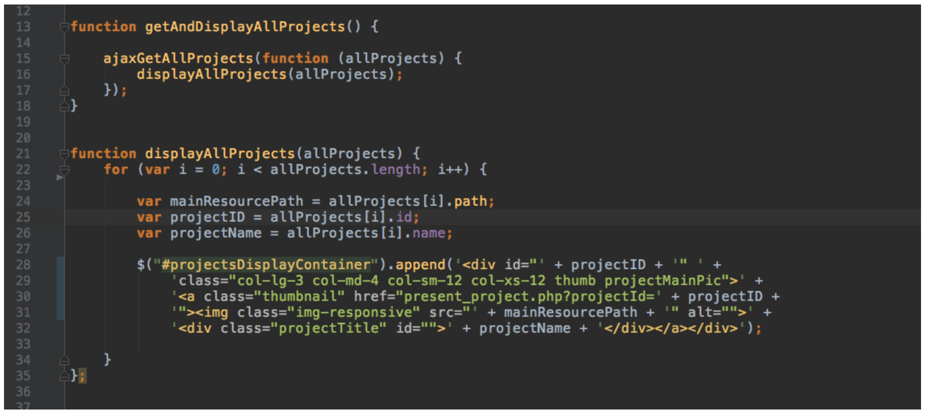
```javascript

// generalize AjaxCall
function postAjaxCall(serverPage, actionName, onSuccessFunction) {

    $.ajax("../projects/" + serverPage + ".php", {
        method: 'POST',
        data: {"action": actionName},
        dataType: 'JSON'
    }).done(function (response) {
        console.log("ajax " + actionName + " success");
        onSuccessFunction(response);

    }).fail(function (xhr, err) {
        console.log("while calling " + actionName);

        // Uncomment for detailed error messages.
        // console.log(xhr.responseText);

        var responseTitle= $(xhr.responseText).filter('title').get(0);
        console.log($(responseTitle).text() + "\n" + formatErrorMessage(xhr, err) );

    });
}


function postAjaxCallSendDictParams(serverPage, actionName, aDict, onSuccessFunction) {

    aDict["action"] = actionName;

    $.ajax("../projects/" + serverPage + ".php", {
        method: 'POST',
        data: aDict,
        dataType: 'JSON'
    }).done(function (response) {
        console.log("ajax " + actionName + " success");
        console.log(response);
        // console.log(JSON.parse(response));
        onSuccessFunction(response);
        // return response;
    }).fail(function (xhr, err) {
        console.log("while calling " + actionName);

        // Uncomment for detailed error messages.
        console.log(xhr.responseText);

        var responseTitle= $(xhr.responseText).filter('title').get(0);
        console.log($(responseTitle).text() + "\n" + formatErrorMessage(xhr, err) );

    });
}
```

Figure 3.2: Generalised Ajax Calls API.



```javascript

// CREATE PROJECT PAGE
function ajaxSetProjectTitleDB(projectId, titleProject, onSuccessFunction) {
    var dictParams = {"titleProject" : titleProject, "projectId" : projectId};
    postAjaxCallSendDictParams("projects_server", "setProjectTitleDB", dictParams, onSuccessFunction);
}

function ajaxSetProjectDescriptionDB(projectId, descriptionProject, onSuccessFunction) {
    var dictParams = {"descriptionProject" : descriptionProject, "projectId" : projectId};
    postAjaxCallSendDictParams("projects_server", "setProjectDescriptionDB", dictParams, onSuccessFunction);
}


function ajaxGetUploadedResourcesForProject(projectID, onSuccessFunction) {
    var dictParams = {"projectID" : projectID};
    postAjaxCallSendDictParams("projects_server", "getUploadedResources ", dictParams, onSuccessFunction);
}


function ajaxDeleteResource(resourceId, onSuccessFunction) {
    var dictParams = {"resourceId" : resourceId};
    postAjaxCallSendDictParams("projects_server", "deleteResource", dictParams, onSuccessFunction);

}

function ajaxsetStarsRatings(numStars, category, projectID, memberID, onSuccessFunction){
    var dictParams = {"numStars" : numStars, "category": category, "projectID":projectID, "memberID":memberID};
    postAjaxCallSendDictParams("projects_server", "setStarsRatings", dictParams, onSuccessFunction);
}


function ajaxSetProjectLabels(labelsArray, projectID, memberID, onSuccessFunction) {
    var dictParams = {"labelsArray" : labelsArray, "projectID":projectID, "memberID":memberID};
    postAjaxCallSendDictParams("projects_server", "setProjectLabels", dictParams, onSuccessFunction);
}


function ajaxGetProjectsOfLabels(labelsArray, onSuccessFunction) {
    var dictParams = {"labelsArray" : labelsArray};
    postAjaxCallSendDictParams("projects_server", "getProjectsWithLabels", dictParams, onSuccessFunction);
}

```

Figure 3.3: Ajax Projects API.



```javascript

function getAndDisplayAllProjects() {

    ajaxGetAllProjects(function (allProjects) {
        displayAllProjects(allProjects);
    });
}


function displayAllProjects(allProjects) {
    for (var i = 0; i < allProjects.length; i++) {

        var mainResourcePath = allProjects[i].path;
        var projectID = allProjects[i].id;
        var projectName = allProjects[i].name;

        $("#projectsDisplayContainer").append('<div id="' + projectID + '" ' +
            'class="col-lg-3 col-md-4 col-sm-12 col-xs-12 thumb projectMainPic">' +
            '<a class="thumbnail" href="present_project.php?projectId=' + projectID +
            '"><img class="img-responsive" src="' + mainResourcePath + '" alt="">' +
            '<div class="projectTitle" id="">' + projectName + '</div></a></div>');

    }
};

```

Figure 3.4: Javascript Typical Function Design.

steps in upload_steps.php.

Another clean code principle I have developed while designing the application is trying to perform as much processing of data as possible on the back-end and in queries, obtaining as clear-cut data as needed in front-end (with less functionalities in javascript).

3.4.2 Client-Server Architecture

The Front-end API is represented by the functions responsible with the data display which are organised in the js files and the ajax functions.

I make a formal distinction between functions which are both called and executed on server-side and those which are actually called on client side in javascript, by keeping them in separate files/by refering to them in 2 distinct end-points. The ones on the client are passing through the AjaxProjectsAPI and are executed in projects_server.php.The functions that are both called and executed on server are collected/defined in projects_functions.php. I have decided which of the two function execution model to use in each situation, by analysing the on the amount of data requested and the probability of future reuse of the inquired data.

In the following I will present the client-server architecture of the application by analysing the interactions derived from each page of the projects module.

3.4.3 All Projects Page

`/projects/index.php`

This page displays all existent projects and a collection of labels that users can select for filtering and searching the projects by categories.

The data of the category tags is retrieved through the getAllProjectsLabels() of the projects_functions.php endpoint, which is displayed directly in the index.php.

For the category filter tags system, I use Ajax in retrieving and displaying the projects that result in labels selection. I have prefered to dynamically update the page with the selected projects rather than refresh the page, as it was more facile to hide the unselected projects in javascript than updating the page with the previously selected labels. This way, after the filter event is triggered, only the projects section is dynamically changed, the selected labels remaining unchanged.

All these interactions are described in Figure 3.4.

The page also gives the user the possibility of adding a new project. The 'Add New Project' button will redirect the user to the following url: `projects/edit_project.php?projectId`. In the background, the click event actually links to `projects/create_project.php`, from where the user is redirected to the edit page. I will explain the logic of this process in the 'Create New Project Page' description.

3.4.4 Create New Project Page

`projects/create_project.php`

create_project.php has only a mediating and rerouting function between index.php and edit_project.php. Before saving the edited information of a newly created project, we first need an entry of the new project in the db. In create_project.php, we insert a new project in the database with no other data than its id. Then we extract the id, sending it as an argument and redirecting to edit_project.php?projectID. See Figure 3.7.

In the end, creating a new project and editing an existing one, both happen in the edit page, as we'll subsequently see.

3.4.5 Edit Project Page

`/projects/edit_project.php/{projectID}`

The page is accessed in two situations. One instance is when the user creates a new project, case in which it receives the id of the new project through a GET protocol from the create_project.php. The other is when the project was already created and the user only expects to edit the information. This case is accessed from the present_project page. At the moment the editing of an existing project is open only to administrators. Their access to edit the project is done from the same project presentation page available to all users, through an event which is visible only the the admins of the system, as shown in Figure 3.8.

Editing projects connects to both main endpoints: the projectsDataServerAPI.php and the projectsAPI.php. The projectsAPI.php returns information about the editing project: title, description, uploaded media resouces, materials list. The functions contained in this endpoint are depicted in figure XX.

The projectsDataServerAPI.php endpoint processes the edited information of the project sent to the server through the Ajax API. The Ajax functions are called by triggered events defined in projects.js, projects_equipment.js and addSteps.js files, all included in edit_project.php.

A core feature of the projects section is the presentation of the project in a succession of steps. The javascript functions of editing these steps are initialized in addSteps.js. Although it is a self-sustaining solid feature, instead of dedicating a new page for editing the steps, we opted for displaying it within the edit_project.php page, in a full-page pop-up element. The content of each step is modified dynamically inside same markup elements. Javascript functions used in addSteps.js: showStepPanel(), addStep(), deleteStep(), loadProjectProgress(),

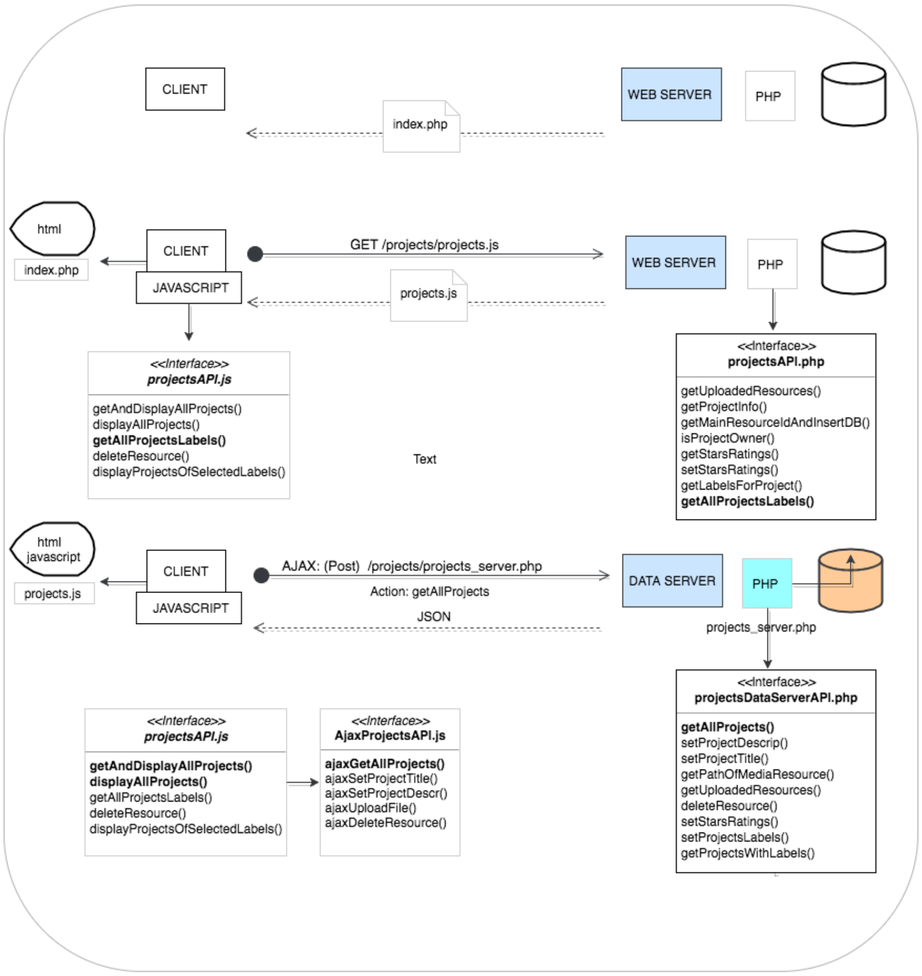
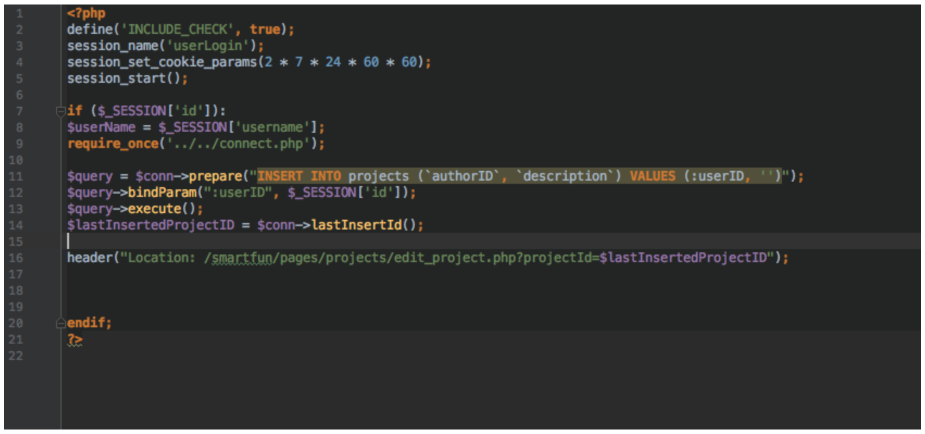
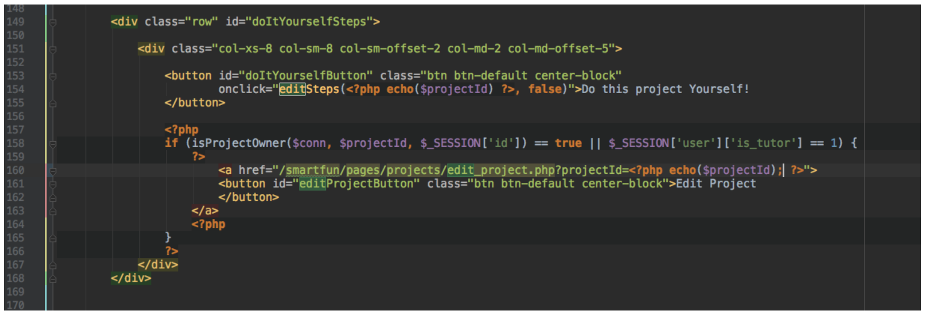


Figure 3.5: Architecture All Projects.



```php
<?php
define('INCLUDE_CHECK', true);
session_name('userLogin');
session_set_cookie_params(2 * 7 * 24 * 60 * 60);
session_start();

if ($_SESSION['id']):
$userName = $_SESSION['username'];
require_once('../../connect.php');

$query = $conn->prepare("INSERT INTO projects (`authorID`, `description`) VALUES (:userID, '')");
$query->bindParam(":userID", $_SESSION['id']);
$query->execute();
$lastInsertedProjectID = $conn->lastInsertId();

header("Location: /smartfun/pages/projects/edit_project.php?projectId=$lastInsertedProjectID");



endif;
?>
```

Figure 3.6: Create Project Page.



```php

        <div class="row" id="doItYourselfSteps">

            <div class="col-xs-8 col-sm-8 col-sm-offset-2 col-md-2 col-md-offset-5">

                <button id="doItYourselfButton" class="btn btn-default center-block"
                        onclick="editSteps(<?php echo($projectId) ?>, false)">Do this project Yourself!
                </button>

                <?php
                if (isProjectOwner($conn, $projectId, $_SESSION['id']) == true || $_SESSION['user']['is_tutor'] == 1) {
                    ?>
                        <a href="/smartfun/pages/projects/edit_project.php?projectId=<?php echo($projectId); ?>">
                        <button id="editProjectButton" class="btn btn-default center-block">Edit Project
                        </button>
                    </a>
                    <?php
                }
                ?>
            </div>
        </div>


```

Figure 3.7: Edit Project.

updateProgresBar().

Functions of the ajaxProjectsAPI.js that connect the editing projects front-end with the endpoint projectsDataServerAPI.php: ajaxSetProjectTitleDB, ajaxSet-ProjectDescriptDB.

3.4.6 Present Project

`/projects/present_project.php/{projectID}`
Gets information from web server about the requested project. Besides the project description and a carrousel with project pictures, it includes a few dynamic components the user can interact with. present_project.php include a number of javascipt files, each correspondent to one of these UI components.

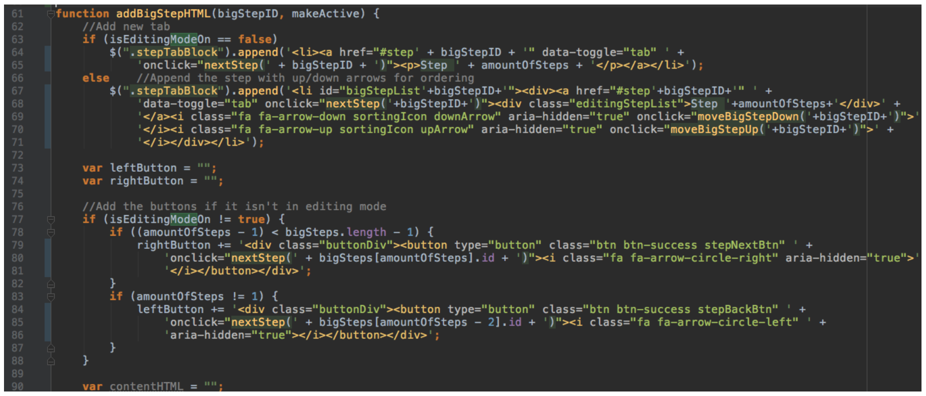
One component is the Stars Rating System, where users can evaluate the project by voting on a 5 stars scale on criterias denoting project's level of: difficulty, fun and general. In stars_ratings.js I define the functions that show interactions with the ratings system. Each criteria voted in the system is individually updated in the database through a concurrent Ajax.

Second dynamic component is Projects Labels Section, where users can participate in adding descriptive tags about the project. In the current version, users can add up any labels that are yet non-existant. We have two alternative scenarios for dealing with the obvious problems this approach raises (too many user entries, unrelevant of inapropriate label names). We consider implementing a filter of the labels as a feature in the admin panel, or alternatively we could leave the feature only for admin use.

The Project Equiments Section is the third component users can interact with. This component lists all the needed equipments for carrying out the project. Users can check the items they already possess or can send to basket items they would need and like to buy. The event handling for the equipments is encapsulated in projects_equipment.js.

Users can exchange ideas through the Comments Section, which can be monitored and managed(edited/deleted) by admins. comments.js processes the events linked to users comments.

Users can find a detailed description of how to do themselves the project in the Steps Section. Steps represent a tutorial about building the project. We use the same UI component for displaying the steps as for editing them, by using control structures and javascript hide and display functions, as the following code snipet shows [See Figure 3.9].



```javascript
function addBigStepHTML(bigStepID, makeActive) {
    //Add new tab
    if (isEditingModeOn == false)
        $(".stepTabBlock").append('<li><a href="#step' + bigStepID + '" data-toggle="tab" ' +
            'onclick="nextStep(' + bigStepID + ')"><p>Step ' + amountOfSteps + '</p></a></li>');
    else    //Append the step with up/down arrows for ordering
        $(".stepTabBlock").append('<li id="bigStepList'+bigStepID+'"><div><a href="#step'+bigStepID+'" ' +
            'data-toggle="tab" onclick="nextStep('+bigStepID+')"><div class="editingStepList">Step '+amountOfSteps+'</div>' +
            '</a><i class="fa fa-arrow-down sortingIcon downArrow" aria-hidden="true" onclick="moveBigStepDown('+bigStepID+')">'
            '</i><i class="fa fa-arrow-up sortingIcon upArrow" aria-hidden="true" onclick="moveBigStepUp('+bigStepID+')">' +
            '</i></div></li>');

    var leftButton = "";
    var rightButton = "";

    //Add the buttons if it isn't in editing mode
    if (isEditingModeOn != true) {
        if ((amountOfSteps - 1) < bigSteps.length - 1) {
            rightButton += '<div class="buttonDiv"><button type="button" class="btn btn-success stepNextBtn" ' +
                'onclick="nextStep(' + bigSteps[amountOfSteps].id + ')"><i class="fa fa-arrow-circle-right" aria-hidden="true">'
                '</i></button></div>';
        }
        if (amountOfSteps != 1) {
            leftButton += '<div class="buttonDiv"><button type="button" class="btn btn-success stepBackBtn" ' +
                'onclick="nextStep(' + bigSteps[amountOfSteps - 2].id + ')"><i class="fa fa-arrow-circle-left" ' +
                'aria-hidden="true"></i></button></div>';
        }
    }

    var contentHTML = "";
```

Figure 3.8: Show or Edit Steps Control Structure.