Bachelor’s thesis

Bachelor of Engineering, Information Technology

NTIETS13B

2017

Aleksi Tuominen

DEVELOPMENT OF A WEB PLAYGROUND FOR SMALL CHILDREN

TURKU AMK

TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S / MASTER'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information technology | Healthcare technology

2017 | 31

Aleksi Tuominen

DEVELOPMENT OF A WEB PLAYGROUND FOR SMALL CHILDREN

The purpose of this thesis was to study and develop web-based playground-environment for daycare-aged children using open-source JavaScript libraries.

The thesis begins by introducing the company and the background of the daycare-project, as well as giving insight into the current situation of the problem revolving around day care centers in Japan. Next, the thesis covers the current situation of the company web pages where the playground-environment will be implemented. In addition, a walkthrough of the implementation plan is also presented. The thesis then moves into more detailed explanation of the used tools, in particular the physics- and puzzle engines and introduces their installation and usage. The thesis also briefly covers the development tool used in the project, and then examines the testing procedures, covering the test environment, methods and lastly results and problems, followed up by their solutions. Finally, the thesis covers the actual implementation and the finished stage of the developed playground -environment.

KEYWORDS:

Matter.js, JavaScript, HTML, CSS, Web-development, Healthcare Technology

CONTENTS

PICTURES

TABLES

LIST OF ABBREVIATIONS (OR) SYMBOLS

HTML	Hyper Text Markup Language
CSS	Cascade Style Sheets
FTP	File Transfer Protocol
IDE	Integrated development environment

1 INTRODUCTION

The rapidly declining birthrate of Japan [1] has led to the need of providing day care centers growing bigger than ever. The more populated prefectures like Kanto are suffering from not only the lack and under-funding of centers, but also day care staff, which makes finding a spot in day care even harder. This increasing lack of daycare centers [2] has given many companies in Japan a possibility to widen their reach. In the growing market of new daycare projects, a way to stand out from the others can prove to be the key for customers looking for a place for their children.

Following this train of thought, formerly only an office supplies provider, Shobundo Inc. started a project to establish a day care center in Sendai. In addition to hiring childcare specialists and going through the required legal procedures, a web-site had to be developed. As a part of this site, a plan to include a play-corner for children and parents to explore and enjoy together was brought up. The purpose of this thesis is to walk the reader through the planning, development and finalization of said function.

The first chapter takes a deeper look into the company itself, expanding on the agenda and the plan in more detail. The second chapter offers a brief overview of the current project followed by the implementation plan of the playground environment. Moving onto the development section, the fourth chapter introduces the JavaScript-libraries used to develop the playground. Bringing real life data into the scope, the fifth chapter presents the procedures and findings of developing the playground environment. Topics such as testing environment, methods used, and found problems are introduced, providing the reader with a closer look into the actual software. Finally, the thesis wraps things up by going through the final implementation to the actual web site, and taking a last look on the finalized page.

As a whole, these topics should provide the reader with insight on web development using open-source JavaScript libraries, from the first stages to the final product.

2 THE COMPANY

Founded in 1936, Shobundo Inc. focuses on the sales of foreign office supplies, working in close cooperation with companies such as Herman & Miller. The sale items mostly consist of office chairs manufactured by foreign companies. Shobundo Inc. currently employs approximately ten workers, and is located in the city of Sendai, Miyagi-prefecture. The company capital is around thirty million Japanese yen.

2.1 Day care project

The Day Care project started in July 2016, and the main goal was to establish a new day care center in the same city where Shobundo Inc. is located, namely Sendai. The future day care center would be called ぶんぶん保育園 (jap. Bunbun Day Care) [3]. Location wise, a place for the center was reserved near the central area of Sendai city, offering parents commuting to mentioned area to use the service conveniently.

As a part of the project, new web pages had to be developed, and this task was outsourced to another company. Due to the day care center’s information not being fully confirmed by the time the web pages were done, the administration was handed back to Shobundo Inc. This lead to the increase in workload, but also opened up possibilities to be creative.

From this, the idea to increase the interactive elements of the web site was born. In addition, since the character and theme design was already done and mainly in place, using these as a model, a plan to bring both children and their parents a way to experience the day care center’s cute characters through games was brought up and put to action.

3 PROJECT OVERVIEW

At first, it is important to take a look at what the web playground actually is. The main idea behind the project was to provide both the parents and children an activity that could be enjoyed together. In addition, this kind of a function would separate the day care center from others and appeal in a unique way. Because of the platform restrictions, the development team saw best to narrow down the usage to JavaScript and JQuery plugins, which proved to be more than a fitting solution.

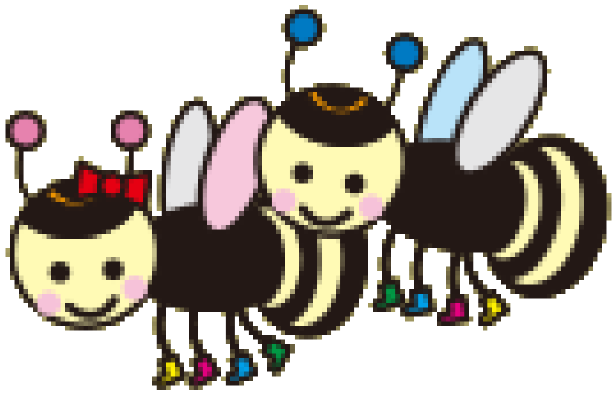
3.1 Implementation plan

Making use of said JavaScript libraries, more specifically Matter.js [4] and Jigsaw.js [5], two main features were achieved. The first, a physics –based drag & drop- type playing environment is based on the matter.js extension. The goal was to create an easily accessible, not too difficult building block simulator for the younger children to play with. The physic objects would respond to touch and be movable, throwable and all in all toy-like.

The main focus of this function was in its appearance. As a part of the day care project, logo characters had been designed, and using these characters as a part of the playground environment was deemed suitable. Picture 1. shows one of the logo characters. Simply put, the plan was to develop a function that even without having a set goal or finish, would still be interesting.

The second part of the playground environment is a jigsaw puzzle game. The feature is based on jigsaw.js, a JavaScript extension including the calculations and placement measurements for the puzzle pieces, which later on had to be revised to fit the page in development. Unlike the previous feature, the jigsaw puzzle was aimed at a little bit older children, who are already able to comprehend matters such as puzzles and mechanics.

The plan was to implement the playground as part of the site structure, accessed through the navigation bar. A banner in the top page also provided a link to the playground site, while at the same time advertising the function.



Picture 1. Character design.

4 LIBRARY INTRODUCTION

4.1 Matter.js

According to the matter.js Github page, “Matter.js is a 2D rigid body physics engine for the web written in JavaScript.” The features include rigid bodies, compound bodies, composite bodies, concave and convex hulls and a lot more. The engine is developed by Liam Brummitt, and licensed under the MIT License (MIT). In addition, add-ons such as MatterTools [6] are available for developers, making debugging easier.

4.1.1 Installation

In order to be able to use the matter.js physics engine, two additional components had to be installed. The first one, Node.js [7], is a JavaScript runtime built on Google Chrome's V8 JavaScript engine. Thanks to the non-blocking I/O model, Node.js is lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world. The installation itself is easy and efficient, mainly due to the fact that Node.js also comes with it’s own installer.

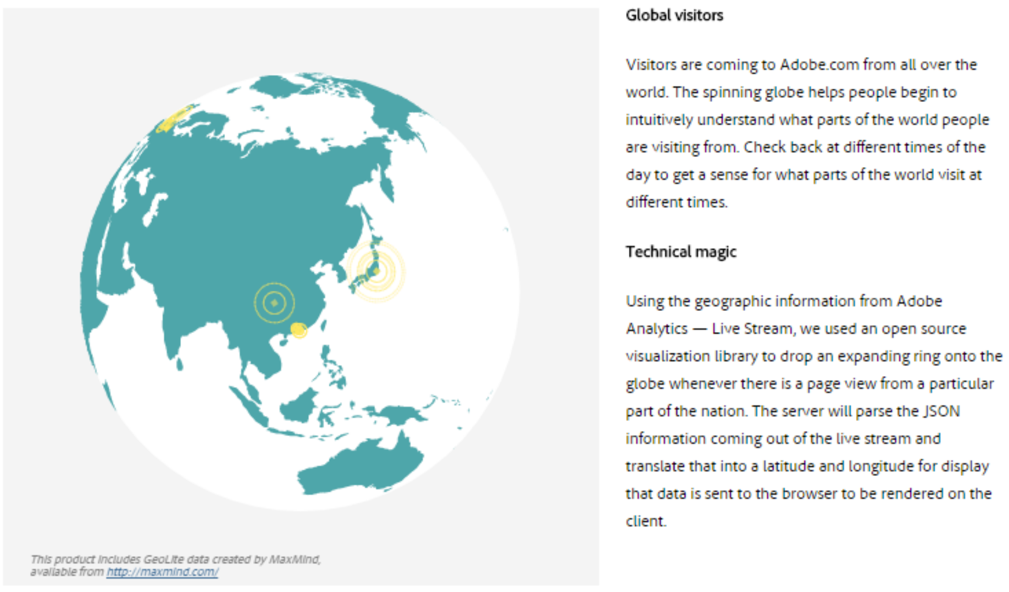
The second required component is Gulp[8]. Gulp is a toolkit developed to automate time-consuming tasks while developing, enabling the developer to focus on the actual work. Installing Gulp requires the previously installed Node.js, and unlike Node.js, Gulp has no installer, meaning the installation is done using command line. The installation command is shown in table 1.

Table 1. Gulp command line installation.

```
npm install gulp-cli -g
npm install gulp -D
touch gulpfile.js
gulp --help
```

4.1.2 Usage

The matter.js is used in many different applications, ranging from games to art [9]. Regardless of it being open source material, companies like Adobe are using matter.js in their own projects [10]. A dashboard displaying from which countries visitors are coming to adobe.com is using matter.js to render the spinning globe and its splash effects. Picture 2. shows the Adobe web page.



This product includes GeoLite data created by MaxMind, available from http://maxmind.com/

Global visitors

Visitors are coming to Adobe.com from all over the world. The spinning globe helps people begin to intuitively understand what parts of the world people are visiting from. Check back at different times of the day to get a sense for what parts of the world visit at different times.

Technical magic

Using the geographic information from Adobe Analytics — Live Stream, we used an open source visualization library to drop an expanding ring onto the globe whenever there is a page view from a particular part of the nation. The server will parse the JSON information coming out of the live stream and translate that into a latitude and longitude for display that data is sent to the browser to be rendered on the client.

Picture 2. Adobe Analytics Livestream.

The basic structure of a matter.js software has to include at least two of the components; the actual matter.js file, as well as render.js. These files provide the core functions required to create a basic software. First, the environment where the objects are going to be handled must be defined. In matter.js, this is called Matter.World, and can be assigned as value of a variable. The Matter.world module contains methods for creating and manipulating the world composite. A Matter.World has properties such as gravity and bounds, enabling basic features such as collision to be applied to objects, Matter.Bodies. Next, an engine must be created. According to the documentation of matter.js, “The Matter.Engine module contains methods for creating and manipulating engines. An engine is a controller that manages updating the simulation of the world.” (brm.io 2016). Lastly, a render has to be created. The Matter.Render visualizes instances of the previously created Matter.Engine. The render is actually a simple HTML5 canvas, and supports options such as wireframes, vectors and sprites, from which the last ones were used in the thesis project. With this, the core elements are in place, but the software is still not ready to be launched. The next step is to add objects to the world. Objects are called Matter.Bodies, which have a plethora of different methods for creating rigid body models. According to the matter.js site, the most commonly used configurations are “rectangles, circles and other polygons.”(brm.io 2016). Table 2. introduces a simple command for creating a rectangular Matter.Bodies object. In this example, the x and y values define the location where the object is spawned, and the options can include features such as friction, restitution etc.

Table 2. Creating a simple Matter.Bodies –object.

```
Matter.Bodies.rectangle(x, y, width, height, [options]) → Body
```

After creating the objects, they must be added to the world. This is done through the World.add command, which takes the variables storing the values of the previously created Matter.Bodies objects. Lastly, the engine and render are launched and the program should now include the added objects, affected by the gravity from the Matter.World instance. Table 3 introduces a basic matter.js program which simply creates two rectangle objects, hitting each other.

Table 3. Basic matter.js program using built in renderer and runner.

```
// module aliases
var Engine = Matter.Engine,
    Render = Matter.Render,
    World = Matter.World,
    Bodies = Matter.Bodies;

// create an engine
var engine = Engine.create();

// create a renderer
var render = Render.create({
    element: document.body,
    engine: engine
});

// create two boxes and a ground
var boxA = Bodies.rectangle(400, 200, 80, 80);
var boxB = Bodies.rectangle(450, 50, 80, 80);
var ground = Bodies.rectangle(400, 610, 810, 60, { isStatic: true });

// add all of the bodies to the world
World.add(engine.world, [boxA, boxB, ground]);

// run the engine
Engine.run(engine);

// run the renderer
Render.run(render);
```

4.2 Jigsaw.js

Jigsaw.js is originally part of a class project made by Gavin Macken. It consists of two main JavaScript files, one responsible for the calculations of object placement, defining whether the piece is set in a grid or not. The other file consists of functions cosmetical functions, such as the highlighting of the grid while mouse-hovering. In addition, the mouse and keyboard actions are also defined withing this file. Unlike matter.js, jigsaw.js offers little customization, and was used almost as it is, with small tweak to the appearance. The customizations will be gone through in depth in chapter five.

5 DEVELOPING THE APPLICATION

Prior to the beginning of the actual development cycle, an editor that would fulfil the needs of the project and the standards of the company had to be found. The main requirement was customizability. The company’s plan was to not only create the application but at the same time, introduce a new tool for web development. For this task, two editor were already pre-chosen, and the following chapter explains the comparison done to define the more suitable editor.

5.1 Editor comparison

According to ratings done by Upwork.com[11] and Codementor.io [12] when it comes to customizability, Atom Editor by GitHub [13], and Sublime [14] were both rated among the best editors for web developers. As described in the Atom Editor homepage, Atom is “A hackable text editor for the 21st Century” (), meaning the editor has a wide array of customization possibilities. In the comparison between web editors done by Codementor.io, Atom editors is stated to be a great tool, especially for web developers who want to customize their editor easily (). In addition the comparison made by Upwork.com, describes Atom Editor as the editor to choose when looking “for maximum extensibility” (). Atom is basically a foundation that the developer is able to build the ideal IDE or editor upon. This feature relies on the packages offered by the creator of the editor, GitHub. Using these packages the user is able to customize the look and feel of the editor. The downside of Atom Editor is its performance. Working with multiple files can turn out to be slow.

The second choice, Sublime was rated third most popular Development Environment in the poll made by Stackoverflow in 2016. [15] According to Upwork.com, Sublime being built on C++ makes in naturally more faster than other editor that are built on web tech (). Like Atom, Sublime is also customizable through packages. Codementor.io mentions the downside of Sublime being the lack of a decent GitHub plugin.

As a conclusion, because of the fast connection to GitHub, as well as the customizability, Atom Editor was chosen as the tool to be used in the project.

5.2 Creating the playground

As seen in picture 3, the first setting of the playground contained only simple frames and objects. For testing purposes, a canvas of 800 x 600 pixels was created, and three ball-objects added. The ball-objects are created through the composites.stack[] –command of matter.js, and the actual code can be seen in table 4. The stack, which is essentially multiple objects spawned at the same time, requires a Matter.Bodies object, which in this case was a Bodies.circle. The circle object was chosen, since most of the illustrations were rounded, which lead to the circle-object being the best choice, considering the collision field, and possible problems upon the introduction of textures.

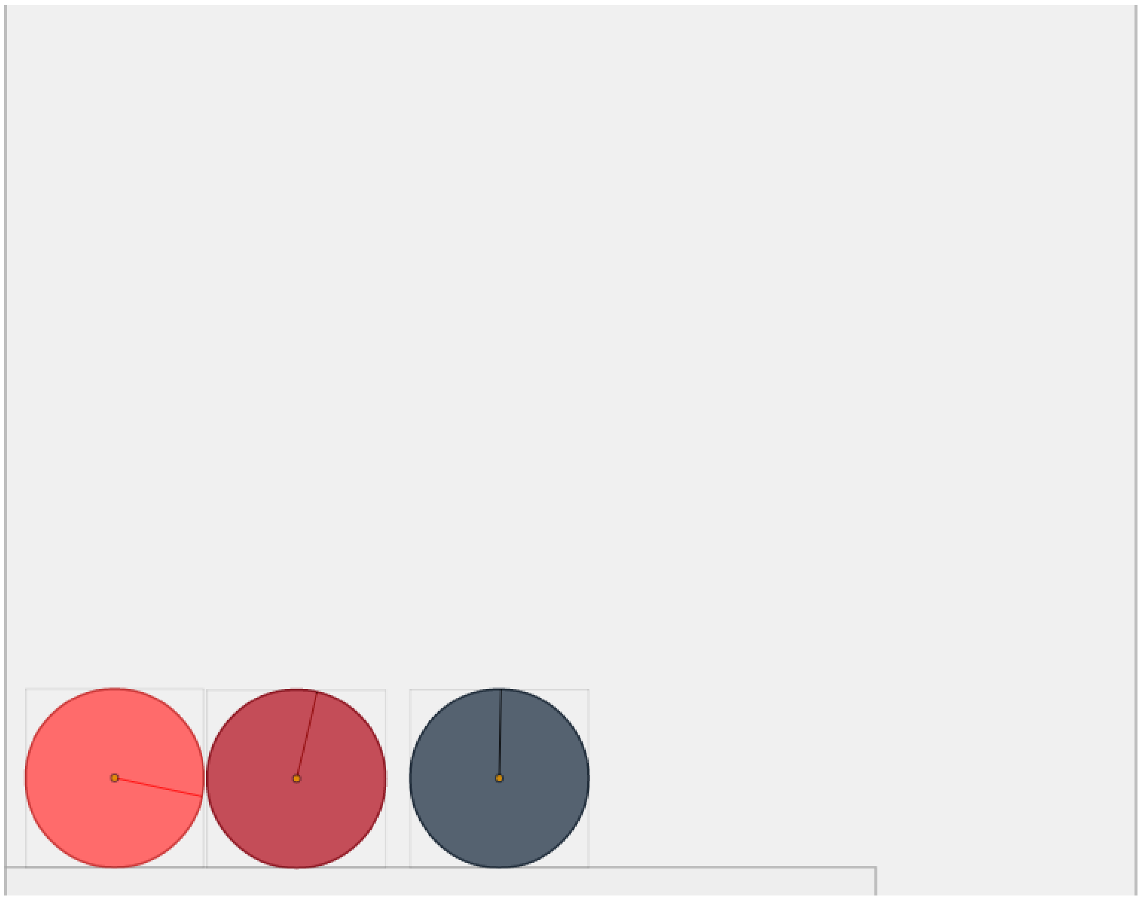
Table 4. Creating the ball-objects.

```
//Create a stack of sprites
var stack = Composites.stack(0, 10, 3, 1, 20, 10, function(x, y) {
   return Bodies.circle(x, y, 60, { restitution: 0.9,
  });
});
```

In order to limit the movement of the objects, borders had to be created. This was accomplished by placing Bodies.rectangle –objects in the bottom, and both sides of the canvas, creating the floor and the walls. Differing from the ball-objects, the walls and the ground had to be able to withstand collision without moving, and thus had to be made static. This was accomplished by including the isStatic-attribute in the creation of the objects. This lead to the wall and ground objects being immune to both collision-based as well as mouse-based movement. Table 5 displays the creation of said objects. The walls and the wall can also be seen in picture 3.

Table 5. Creating the wall- and ground objects.

```
var ground = Bodies.rectangle(200, 610, 810, 60, { isStatic: true  });
var wall_right = Bodies.rectangle(800, 610, 40, 1500, { isStatic: true  });
var wall_left = Bodies.rectangle(0, 610, 40, 1500, { isStatic: true });
```



Picture 3. Initial playground setting.

5.2.1 Setting up matter.js environment

Rendering and running the created objects was the next phase of the development. Creating variables for instances containing the attributes and functions of the used Matter.Bodies and Matter.Composites. (See Table 6.)

Table 6. Matter module aliases.

```
var Engine = Matter.Engine,
  World = Matter.World,
  Body = Matter.Body,
  Bodies = Matter.Bodies,
  Constraint = Matter.Constraint,
  Composites = Matter.Composites,
  MouseConstraint = Matter.MouseConstraint;
  Render = Matter.Render;
```

Creating the engine was done as shown in Table 7. The engine is responsible for rendering the objects visible, and defining settings for the visual effects. For debugging purposes, the wireframes-attribute was added. When receiving the value of “true”, rendered figures would only be shown with the most core features, without rendering colors, textures and sprites. This was done to define the actual size of the circle-object in order to size the textures accordingly. The textures are gone through in the next sub-chapter.

The actual size of the rendered canvas was also defined during the creation of the engine. In addition, using the engine and its background-attribute, a background-image for the canvas can be loaded in. This is explained more thoroughly in part 5.2.2 Adding textures.

Table 7. Creating a matter.js Engine.

```
var engine = Engine.create(document.body, {
render: {
  options: {
    width: 800,
    height: 600,
    showAngleIndicator: true,
    wireframes: false,
    showPositions: true,
    showBounds: true
  }
}
});
```

Finally, for the user to be able to interact with the created objects, a mouse constraint had to be created. This was achieved by using the MouseConstraint-variable created during the first stage, as seen in Table 6. Table 8. displays the creation of the mouse constraint, followed by the addition to the engine-variable.

Table 8. Adding mouse controlled constraint.

```
var mouseConstraint = MouseConstraint.create(engine);
World.add(engine.world, mouseConstraint);
```

5.2.2 Adding textures

After setting up the test-version of the playground (See Picture 3.), the next step was to add textures and illustrations in order to make the setting more appealing to it's core audience, mainly day-care-aged children and their parents. Since the web site of the day care already had a certain character theme, a choice was made to use the same illustrations for the interactive elements as well.

Development-wise, this was achieved by making use of the Matter.Render-module[]. According to the official matter.js github page, (brm.io 2016) "The Matter.Render module is a simple HTML5 canvas based renderer for visualising instances of Matter.Engine. It is intended for development and debugging purposes, but may also be suitable for simple games. It includes a number of drawing options including wireframe, vector with support for sprites and viewports." In this thesis, both the debugging and drawing options were used.

Previously introduced Matter.Composites stack would incorporate a new attribute through the render-module, enabling the usage of textures in place of the standard colors. This was done by adding the render-module, accompanied by the sprite- and texture child-modules. The texture-module can have a image file as a value, which will load the provided image instead of the standard color-based render. The upgraded creation of the ball-object can be seen in table 9.

Table 9. Ball-object with added texture-module.

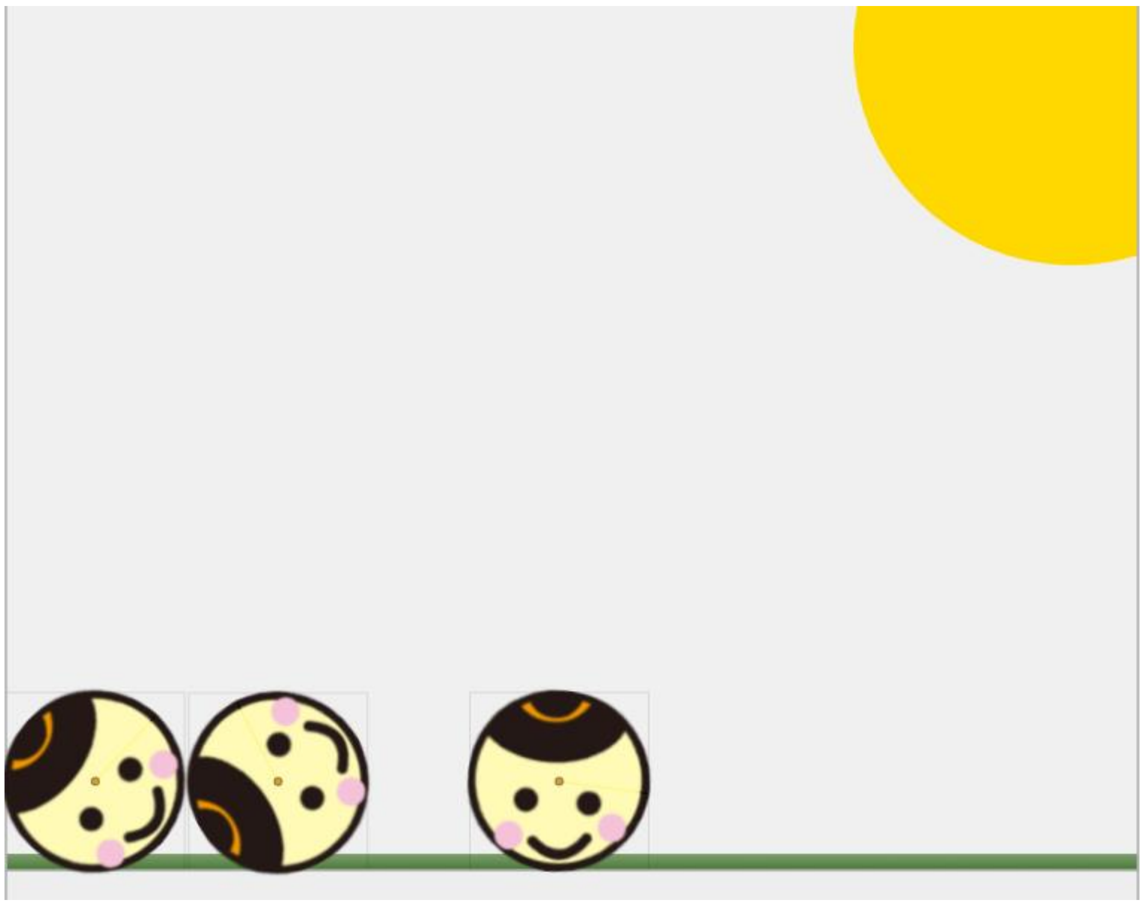
```
//Create a stack of sprites
var stack = Composites.stack(0, 10, 3, 1, 20, 10, function(x, y) {
  return Bodies.circle(x, y, 60, { restitution: 0.9,
    render: {
      sprite: {
        texture: 'img/hoiku01_head.png',
       }
      }
    });
});
```

Next, the currently blank background was upgraded. By using the background-attribute of the Matter.Engine, a image file could be assigned as the background of the rendered canvas. Imitating a landscape, a .png file was created and loaded into the canvas. Due to the fact that the full design of the web site was not yet completed, the background image would be replaced upon the implementation to the day care web site, and thus did not contain the character theme. The test image displayed a simple landscape-setting, with a sun on the right upper corner, and a green ground, mimicking a grassy ground. Table 10 displays the updated table with the added background-attribute.

Table 10. Updated Matter module aliases.

```
var engine = Engine.create(document.body, {
    render: {
      options: {
        width: 800,
        height: 600,
        showAngleIndicator: true,
        wireframes: false,
        showPositions: true,
        showBounds: true,
        background: 'img/bg.png'
      }
    }
    });
```

After adding textures to the ball-objects and defining a background image for the canvas, the test version included the core functionalities, accompanied with the possibility of replacing the textures upon implementation to the actual web site. The textures scale with the actual size of the ball-object, as can be seen from the collision-range rectangles, displayed in picture 4. Picture 4. displays the test-environment with added textures and background.



Picture 4. Playground with added textures

5.2.3 Variation

After the basic functionality was completed, new functions were planned for implementation. The current playground was built around one key feature; the possibility of experiencing gravity-based objects in web-environment. By harnessing this core feature, a new, goal-oriented feature was designed. The next step in developing the playground was adding a building block simulation, using the existing engine and world. In addition, smaller features such as adding new bodies and changing the gravity with a click of a button, were also added.

The first new feature added was the possibility to add a new body to the canvas. This was done by sticking a simple JQuery onClick-function to a HTML-button. The function would consist of a return value of previously used Bodies.Circle-method. This way, upon receiving the click, the function would return the value of one Bodies.Circle –object, included with the textures explained in the previous chapter, and finally as a part of the World.add –function, the object would be rendered on the canvas. Table X displays the add-function.

Table 11. Add a new body, onClick function.

```
var addHoiku = function () {
  return Bodies.circle(30, 0, 60, { restitution: 0.9,
    render: {
        sprite: {
            texture: 'img/hoiku01_head.png',
          }
        }
      });
}
$('.add').on('click', function () {
  World.add(engine.world, addHoiku());
})
```

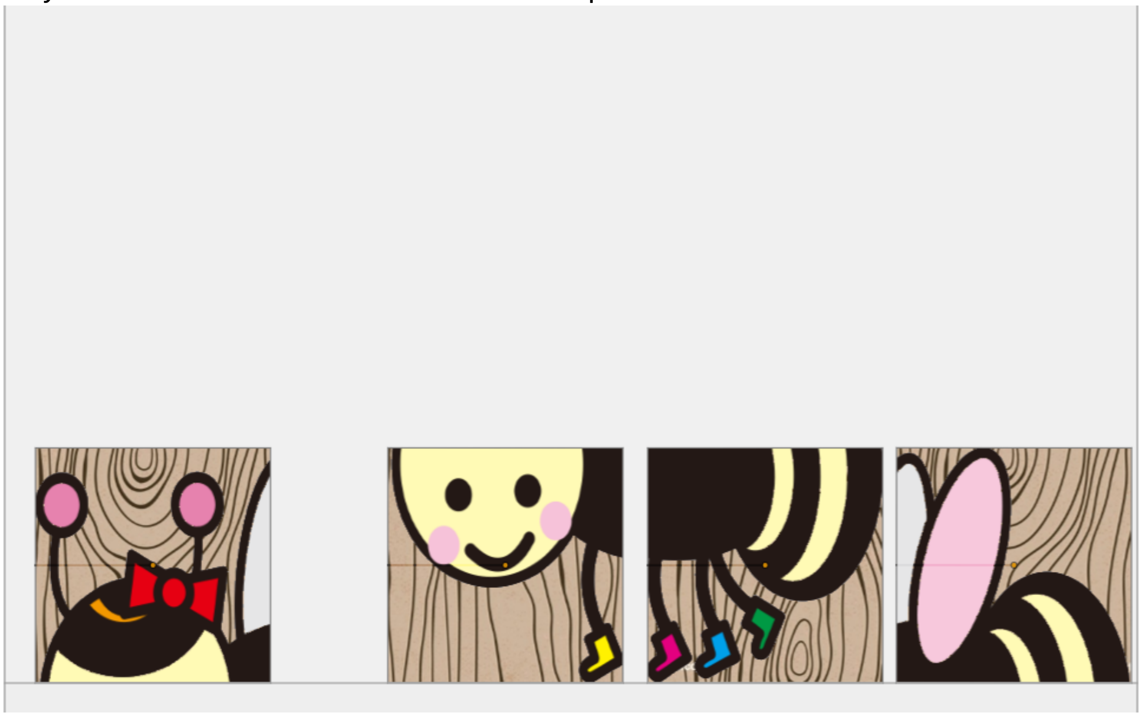
Following this, a new play-form was developed. Instead of circle-objects, rectangular-ones were chosen to work as building blocks, which would enable a more goal-oriented environment.

The rectangular-objects were created using the matter.js Bodies.Rectangle –module, which takes in the x and y coordinates, and the width and height of the to be created object. In addition, using the same render method that was used with the circle-objects and their textures, a picture of one of the day care characters was cut in four pieces and inserted into individual objects. Table X displays the creation of one of the four objects.

Table 12. Creating a Bodies.Rectangle object.

```
var block1 = Bodies.rectangle(100, 400, 200, 200, { friction: 0.9,
  render: {
    sprite: {
      texture: 'img/box1.png'

      }
    }
  });
// create the ground
var ground = Bodies.rectangle(400, 605, 1200, 60, { isStatic: true
  });

// create the walls
var wall_right = Bodies.rectangle(1000, 610, 40, 1500, { isStatic: true,  });
var wall_left = Bodies.rectangle(0, 610, 40, 1500, { isStatic: true });

// add all of the bodies to the world
World.add(engine.world, [block1, block2, block3, block4, ground, wall_right,
wall_left]);

// run the engine
Engine.run(engine);
```

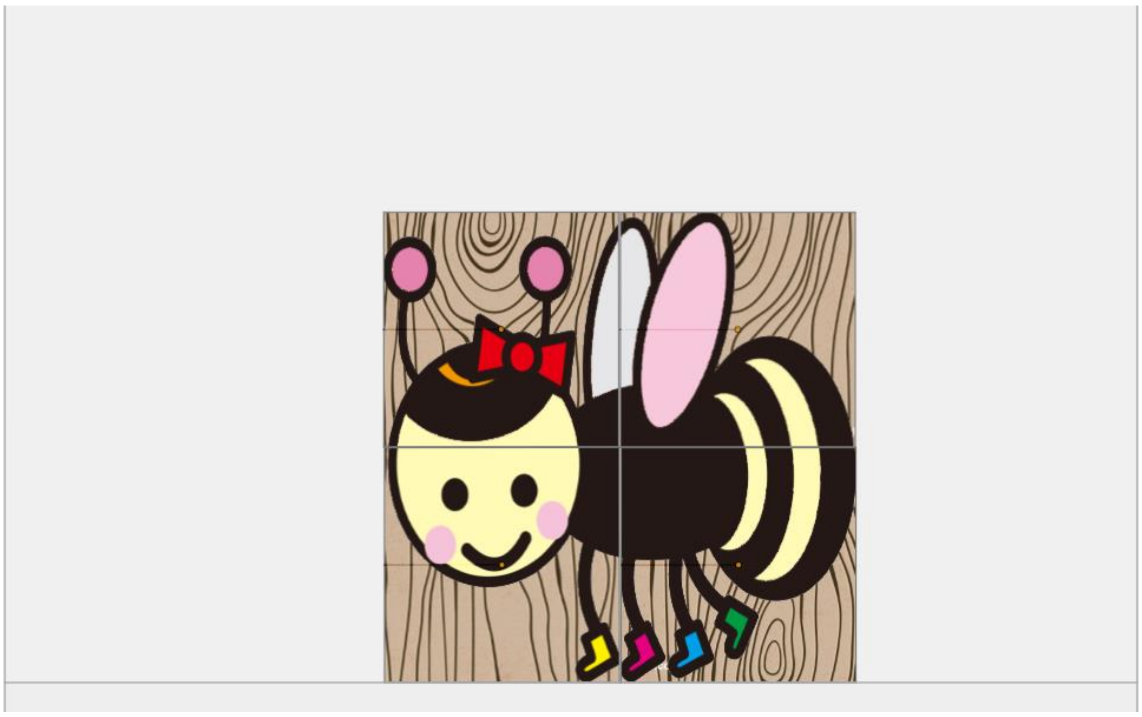
After this, all of the four objects were added one by one to the world with the World.add command. The walls and the ground created during the development of the first function stayed the same. The results can be seen in picture 5.


Picture 5. Building block test scene.

During the testing, a problem with the rotation of the blocks was noticed. When the user grabbed the object, rotating it would be difficult with forcefully making the object collide with other objects, thus making it rotate. This was fixed by adding an attribute to the Bodies.Rectangle-object. When set to infinite, the rotation would be disabled, thus enabling the objects to be placed on top of each other more easily. The updated Bodies.Rectangle object can be seen in table 13 and the test setting in picture 6.

Table 13. Rectangle-object with Inertia.

```
var block1 = Bodies.rectangle(100, 400, 200, 200, { friction: 0.9, inertia : Infinity,
  render: {
    sprite: {
      texture: 'img/box1.png'


      }
    }
  });
```



Picture 6. Finished building block -test setting.

5.3 Developing the puzzle

Moving away from the matter.js environment, a feature for more grown up children was planned and developed. Making use of the jigsaw.js, introduced in chapter 4.4, a jigsaw puzzle game was put under construction. The basic features already inside jigsaw.js were deemed suitable, so little changes were made to the functionality.

Using the jigsaw.js, two five-times-five grids were made. The left-side grid acts as the base where the puzzle pieces would be placed whereas the right-side grid contains the actual pieces. The user is able to pick up a piece by clicking on it, which attaches the piece temporarily to the mouse. By hovering the piece above the base grid, the slot which the user is trying to insert the piece becomes darker, and upon clicking the mouse, the piece will lock into the highlighted area. The initialization of the puzzle can be seen in table 14.

Table 14. The initial settings of the Puzzle.

```
function init() {
  var allElem = document.getElementsByTagName("*");
  for (var i = 0; i < allElem.length; i++) {
    if (allElem[i].className == "grid") grids.push(allElem[i]);
    if (allElem[i].className == "pieces") pieces.push(allElem[i]);
  }
  var randomIntegers = randomArray(pieces.length);

  for(i = 0; i < pieces.length; i++) {
    pieces[i].style.backgroundImage = "url(img/puzzle/piece" +
randomIntegers[i] + ".jpg)";
    pieces[i].style.top  =  getStyle(pieces[i],"top");
    pieces[i].style.left  =  getStyle(pieces[i],"left");
    pieces[i].style.width  =  getStyle(pieces[i],"width");
    pieces[i].style.height  =  getStyle(pieces[i],"height");
    pieces[i].style.cursor = "pointer";
    addEvent(pieces[i], "mousedown", mouseGrab, false);
  }
  for  (var  i  =  0;  i  <  grids.length;  i++)  {
    grids[i].style.top  =  getStyle(grids[i],"top");
    grids[i].style.left  =  getStyle(grids[i],"left");
    grids[i].style.width  =  getStyle(grids[i],"width");
    grids[i].style.height  =  getStyle(grids[i],"height");
}
  document.onkeydown  =  keyGrab;
  keyPiece  =  pieces[0];
  keyIndex  =  0;
  document.getElementById("jumble").onclick  =  jumbleIt;
  document.getElementById("solve").onclick  =  solveIt;
  document.getElementById("next").onclick  =  nextPuzzle;
}
```

5.4 The mobile site

The last step before implementation was to create a mobile-friendly version of the playground. The day cares web sites already contained a version used when the page was viewed in a smaller device, and by using this already existing function, the mobile-site was created.

Using sections, inside one html-file, both the pc and the smartphone versions of the page could be constructed. Under the section with the class of “mainspace sponly”, the code for the mobile-friendly site was placed. By including the code inside this class, the page would automatically resize the elements to fit smaller screens. This way, no further modifications of the style-files were necessary. The smartphone-version included the same elements as the pc-version, with changed class-names for easier style-manipulation. The site-structure can be seen in table 15.

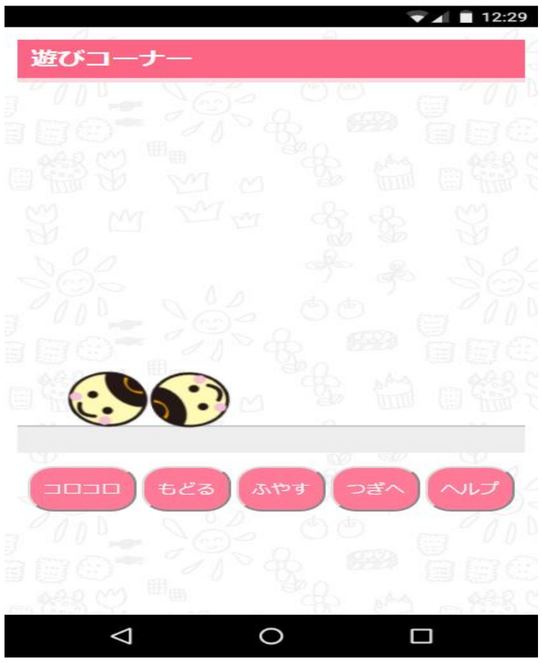
Table 15. Site structure.

```
<main class="mainsize maincontents clearfix" id="playground_main">
        <section class="mainspace pconly">
          <article>
            <div class="">

            </div>
          </article>
        </section>
        <section class="mainspace sponly">
          <article>
            <div class="">

            </div>
          </article>
        </section>
</main>
```

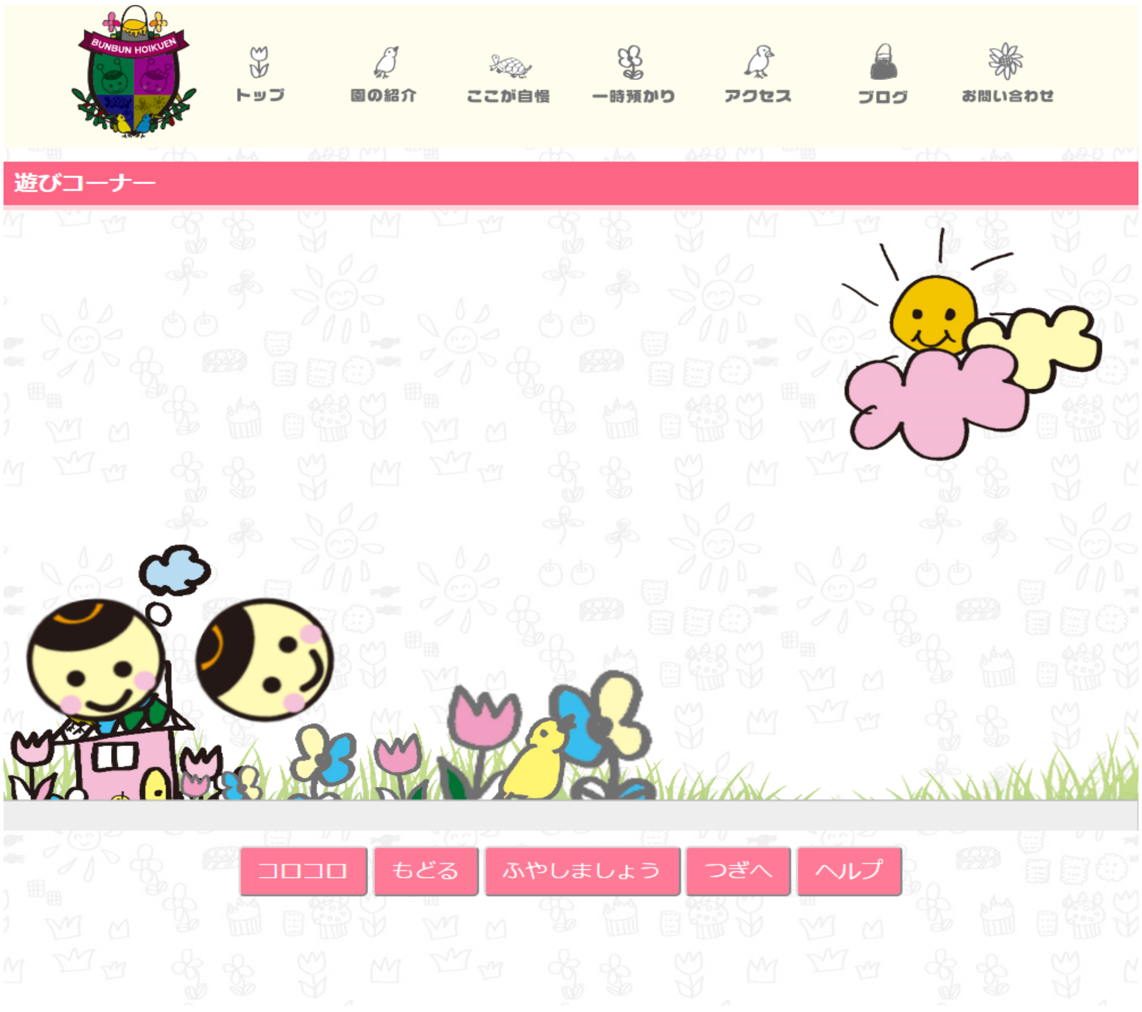
6 IMPLEMENTATION

Upon the completion of the test environment, the next step was to upload the software to the company's servers and begin the finalization. A connection to the company's servers was established through FTP, using FileZilla [16]. The test environment files occupied one folder and the modified CSS and index.html files were replaced with the ones containing code required by the playground. Upon implementation, a problem with the puzzle emerged. The style-settings used during the testing and which the puzzle's placing was mainly depending on were not compatible with the home page's styling, leading to difficulties maintaining the responsive layout of the puzzle. After consideration, the decision was made to leave out the puzzle-section until a working solution was found.

The playground required only few changes to the functionality and the implementation was completed without complications. Buttons were added in order to control the gravity and amount of the spawned characters. In addition, buttons for navigating between the different types of playgrounds and displaying the instructions were added. The appearance underwent multiple modifications in order to fit in to the overall theme of the main page. The background image was replaced with a picture featuring characters used in the day care's advertisement and the web page. The finalized page can be seen in picture 7 and the mobile-friendly version in picture 8.



Picture 7. Finalized mobile version.



Picture 8. Finalized page.

7 CONCLUSION

The increasing difficulty of finding a place in day care centers has affected not only the families, but also the economy and industries. The amount of organizations trying to establish new centers is going up rapidly and in the hunt for customers, a unique selling point is what can make a difference.

The main purpose of this thesis was to introduce one company aiming to establish a day care center, and show how web pages can be used as a virtual playground for both parents and their children.

Using open source JavaScript –libraries, a gravity-based web-playground was developed. The playground is based on a gravity engine and works on all browsers. In addition, the playground is responsive and fits different screen sizes. The playground was also implemented and is currently live on the day care web site.

The thesis project will act as a base for future updates and the playground will be updated with new variations. In addition, the development for finding a solution for the problems faced with the puzzle –game will also continue.

The playground –environment provides parents and their children with a way to get familiar with the characters and theme of the day care, making it a unique way to tie together the customer and the provider. Interacting with something can help to remember, which again can prove to be the deciding factor when choosing a day care center for example.

REFERENCES

[1] Concern as Japan's 2014 birth rate falls to record low *BBC* [Online] [Cited: 28 January 2017] http://www.bbc.com/news/world-asia-30653825

[2] Desperate Hunt for Day Care in Japan *The New York Times* [Online] [Cited 28 January 2017] http://www.nytimes.com/2013/02/27/world/asia/japans-mothers-in-hokatsu-hunt-for-day-care.html

[3] ぶんぶん保育園 *Bunbun Day Care* [Online] [Cited: 2 February 2017] http://bunbun1936.com/

[4] Matter.JS. Available from <https://github.com/liabru/matter-js>. [15 January 2017]

[5] Jigsaw.JS. Available from <https://github.com/gavmac/jigsaw>. [28 January 2017]

[7] Node.JS. Available from <https://nodejs.org/en/>. [15. January 2017]

[6] Matter.JS Matter Tools. Available from <https://github.com/liabru/matter-tools> [15. January 2017]

[8] Gulp.JS. Available from <http://gulpjs.com/>. [15. January 2017]

[9] Les métamorphoses de Mr. Kalia lab212 [Online] [Cited: 28 January 2017] http://lab212.org/Les-metamorphoses-de-Mr-Kalia

[10] Adobe Analytics Livestream Adobe Firehose [Online] [Cited: 28 January 2017] http://adobefirehose.mediarain.com/#/dashboard

[11] Github Atom Editor. Available from <https://atom.io/>. [4 January 2017]

[12] Sublime Text. Available from <https://www.sublimetext.com/> [4 January 2017]

[13] Battle of the Text Editors: Atom, Sublime & Brackets *UpWork* [Online] [Cited: 2 March 2017] https://www.upwork.com/hiring/development/text-editors-atom-sublime-brackets/

[14] Best Text Editor? Atom vs Sublime vs Visual Studio Code vs Vim *Codementor* [Online] [Cited: 1 March 2017] https://www.codementor.io/mattgoldspink/best-text-editor-atom-sublime-vim-visual-studio-code-du10872i7

[15] Developer Survey Results 2016 *Stackoverflow* [Online] [Cited: 2 March 2017] http://stackoverflow.com/research/developer-survey-2016#technology-development-environments

[16] FileZilla. Available from <https://filezilla-project.org/>. [27 February 2017]