COMP0034 Group M

UberKidz Party Planning Website

Hou Chan, Olivier Laben, Dilan Patel, Zhouming Sun, Pok Chan

2019

Catalog

[Introduction 4](#_Toc4372361)

[Project Links 4](#_Toc4372362)

[UberKidz Overview 4](#_Toc4372363)

[Contributions 4](#_Toc4372364)

[Requirements and Deliverables 5](#_Toc4372365)

[Functional Requirements (Based on Interim Report) 5](#_Toc4372366)

[Application Design 6](#_Toc4372367)

[JavaScript Technologies 6](#_Toc4372368)

[Input validation 6](#_Toc4372369)

[DOM manipulation 7](#_Toc4372370)

[Asynchronous JavaScript And XML (AJAX) 7](#_Toc4372371)

[PHP Technologies 8](#_Toc4372372)

[Email Activation Feature 8](#_Toc4372373)

[Remember Me Feature (Via encrypted cookies) 8](#_Toc4372374)

[Cross-Site Request Forgery (CSRF) Defense: Token-Based Form Authentication 9](#_Toc4372375)

[Forgot Password Feature 9](#_Toc4372376)

[Evaluations & Improvements 9](#_Toc4372377)

[Database Design 11](#_Toc4372378)

[Entity-Relationship Diagram 11](#_Toc4372379)

[Design Explanations 11](#_Toc4372380)

[Normalization 12](#_Toc4372381)

[Evaluations and Improvements 13](#_Toc4372382)

[User Interface Design 13](#_Toc4372383)

[Homepage 13](#_Toc4372384)

[Place an order 14](#_Toc4372385)

[Check out page 14](#_Toc4372386)

[Viewing placed orders (customer) 14](#_Toc4372387)

[Viewing placed orders (service supplier) 14](#_Toc4372388)

[Quality Assurance 14](#_Toc4372389)

[Black-Box Test 14](#_Toc4372390)

[User Acceptance Test 15](#_Toc4372391)

[Database Interaction Automated Testing (DBUnit & PHPUnit) 15](#_Toc4372392)

[Automated Testing (Jest & Puppeteer) 16](#_Toc4372393)

[Conclusions 16](#_Toc4372394)

[Appendix 18](#_Toc4372395)

[Appendix 1: Deployment on XAMPP 18](#_Toc4372396)

[Install Composer on Windows 10 19](#_Toc4372397)

[Appendix 2: Deployment on MAMP (Mac) 19](#_Toc4372398)

[MAMP on MAC (method 1) 19](#_Toc4372399)

[MAMP on MAC (method 2) 20](#_Toc4372400)

[Install Composer on Mac 20](#_Toc4372401)

[Appendix 3: PHP Environment Setup 20](#_Toc4372402)

[Appendix 4: User Acceptance Test Form 22](#_Toc4372403)

[Appendix 5: Defect Tracking Log 23](#_Toc4372404)

[References 24](#_Toc4372405)

[HTML Layout Reference (Derived from following templates) 24](#_Toc4372406)

[Composer Libraries 25](#_Toc4372407)

[JS Related 25](#_Toc4372408)

[Testing Related 25](#_Toc4372409)

# Introduction

## Project Links

**GitHub Link:** <https://github.com/billpwchan/COMP0034_GroupM>

**YouTube Link:** <https://www.youtube.com/watch?v=COzMido_RV4&feature=youtu.be>

## UberKidz Overview

Uberkidz is a party planning online platform that will enable users to plan and book parties’ time efficiently. User can either register as a customer or service provider. For service provider, they can offer three types of services: Venue (For hosting party), Menu (For catering) and Entertainment Package (For entertainment). For customer, they can book a service and purchase via UberKidz payment system. The design of UberKidz is based on famous e-Commerce platforms, aiming to provider user a comfortable, efficient and enjoyment shopping journey.

## Contributions

**Bill Chan (Pok Wah Chan)**

* Webpage Framework Implementation
* PHP and HTML implementation for Entertainment Package, menu, and venue-related webpages, including displaying provided services, displaying service details, add chosen service to cart, etc.
* Automated Testing Implementation using DBUnit (PHPUnit) and Jest-Puppeteer

**Matthew Chan (Hou Un Chan)**

* JavaScript implementation (e.g. Validation, Event Handler, etc.) for all form-related code, including but not limited to log in, registration, contactUs, etc.
* MyAccount related webpages, including display account information, display order history, use AJAX for editing account information, etc.

**Zhouming Sun**

* Webpage Framework Design, including webpage layout design, CSS adjustment, and specify functional requirements for HTML elements.

**Dilan Patel**

* Database design and implementation, including ER diagram, MySQL script, database account management, database maintenance, etc.
* Service Provider service management system, including Add/Remove/Edit services.

**Olivier Laben**

* PHP and HTML implementation for Home Page, Contact Page and navigations.
* Deployment-related issues (On Heroko)

# Requirements and Deliverables

## Functional Requirements (Based on Interim Report)

* Registration **(Complete)**
  + JavaScript & PHP form validations **(Complete)**
  + Insert Customer/Service Provider records based on user selection **(Complete)**
  + Email Activation before successful registration **(Additional Implementation)**
* Login and Logout **(Complete)**
  + JavaScript & PHP form validations **(Complete)**
  + PHP session to store user information **(Complete)** -- $\_SESSION[‘userInfo’]
  + Notifications when failed **(Complete)** – SweatAlert2 Library
  + Token-based Authentication **(Additional Implementation)**
  + ‘Remember Me’ via Cookies **(Additional Implementation)**
  + ‘Forget Password’ via Token-based Email Reset **(Additional Implementation)**
* Edit account details **(Complete)**
  + AJAX Calls upon modification **(Complete)**
* Book venue, entertainment package, and menu (customer) **(Complete)**
  + Display Service Details via Pagination **(Additional Implementation)**
  + Display Contained Entertainers/MenuItems **(Additional Implementation)**
  + Select service location via Google Maps API **(Additional Implementation)**
  + Book time slot before adding the service to cart **(Complete)**
  + Unavailable time slots will be indicated **(Not Completed)**
    - **Explanation:** Due to the complexity and limited flexibility of the jQuery DateTime picker we used for displaying booking time, we cannot show unavailable time slots directly on the calendar. The user will get notification via alert box if a specific timeslot isn’t available.
  + Filtering/Refining options by searching service name **(Additional Implementation)**
  + Filtering/Refining options by limiting price range **(Additional Implementation)**
* Provided Service Modification (service supplier) **(Partially Completed)**
  + Add a new Service **(Complete)**
    - Support adding MenuItems/Entertainers as a subset of Menu/Entertainment Package **(Additional Implementation)**
  + Edit a new Service **(Not Completed)**
    - **Explanation:** After discussion, it is not reasonable to allow a service provider to change the detail of a provided service while there exist some customers who already booked it. For instance, after User A booked Venue A, the service provider shall not be able to change the capacity of this venue before notifying users. Thus, our team decided to not implement this operation.
  + Remove a Service **(Not Completed)**
    - **Explanation:** Same concern as the above one. These operations should be performed by admins, who can ensure that all affected customer agrees to the service detail modification.
* Pay function using pre-set account balance **(Complete)**
  + Checkout Page, using default 2000 initial customer balance **(Complete)**
* Apply coupon **(Complete)**
  + AJAX database voucher code retrieval **(Complete)**
* Contact Us form **(Complete)**

# Application Design

## JavaScript Technologies

### Input validation

JavaScript was used as the major technique for validation of user inputs. The validation is performed at the web browser and the data will only be sent to the server after they are verified (client-side validation). This can reduce the burden on the server since any incorrect data sent to the server will lead to

User input validation was implemented mainly in logging in, account registration, edition of account details, addition of new service (service provider) and application of coupons (customer). The validations are generally performed by checking the following:

* *Blank input*: This is the most fundamental check to ensure that all the mandatory fields have been entered by the user.
* *Range:* This is applicable to number input such as quantity, price and venue capacity. These kinds of inputs are restricted to a certain range of values.
* *Length:* This is to limit the length of the string input, or to check of the input has meet the minimum length requirement. Example of input which requires this check is the password entered by the user during account registration.
* *Inclusion of specific characters:* This is to ensure that the input has not includedany invalid characters. For example, all name fields should contain no special characters. On the other hand, some inputs are required to contain some specific characters. The password field in the account registration form is an example that requires this check. The password should contain at least one upper case letter, one lower case letter and one number.
* *Regular pattern*: This is applicable to inputs which have regular patterns, such as email address, post code and phone number. This check can be done by using regular expression (Regex), which defines a particular pattern of string. The input is then compared with the regular expression to verify if it matches with the regular pattern.

For form validation, the form can be prevented from being submitted by using onsubmit=“return validateForm()”. A Boolean variable is used in the method validateForm() and any invalid input will change to the variable to false.

### DOM manipulation

JavaScript was also used to manipulate DOM objects, and manage simple CSS variations and animations upon user actions. An example of animation is the sliding of sections of the account registration form. The account registration form is divided into three different sub-sections. Each section corresponds to one fieldset of the form and different sections will be displayed by clicking “Next” or “Previous” buttons.

When a service provider enters the details of a new menu, the menu items and their corresponding quantities must be specified. A new row will be added to the table when the service provider presses “Add another menu item”. This is achieved by appending a new <tr> object containing all the input fields to the table using JavaScript. Likewise, the service provider can delete a row from the table. However, removal of rows increases the complexity since the “id” attributes of the rows will have to be adjusted depending on which row of the table has been removed to avoid errors in subsequent coding.

### Asynchronous JavaScript And XML (AJAX)

Some scenarios which require data exchange after the page is loaded was made possible by using JQuery AJAX methods. This allows a certain part of the page to be updated without refreshing the entire page, which avoids the “click-wait-refresh” pattern.

When the user updates his/her account details (e.g. name, phone number), an XMLHttpRequest object will be created after the new account detail is validated as mentioned in the “Input validation” section. This object then posts the user’s input for the new account detail to a PHP file called “updateUserProfile” which can update the account detail of that particular user.

AJAX was also employed in the application of coupon and removal of items from shopping cart. The coupon code entered by the customer cannot be validated using normal input validation. AJAX allows the verification of coupon code with the database data associated with coupons and retrieve the corresponding discount if the coupon code exists in the database. When the user removes a cart item, an ajax() method will be called to post the id of the deleted cart item to the “updateCart” PHP file which can update the shopping cart database. It is also worth mentioning that the “updateCart” PHP file contains methods which handle the application of coupon discount and removal of cart items. Therefore, an addition parameter called “method id” will also be posted by the XMLHttpRequest object to identify which method will be used in the “updateCart” PHP file.

When a new entertainment package or a new menu is being added, the service provider will need to specify the entertainers or the menu items which are included in the new service by choosing items from a <select> drop down list. The options in the drop down list are retrieved from the database by using the getJSON() method ( a shorthand AJAX function). This method can get JSON-encoded data from the “getProvidedServices” PHP file using an AJAX HTTP GET request, which in this case are the entertainers or menu items. The retrieved data in the form of an array can then be appended to the drop down list using the $.each() iterator function.

## PHP Technologies

### Email Activation Feature

After user registration, an activation email is sent to the registration email inbox. Initially, the user account stored in the ‘user’ table will have status = 0 and email\_activation\_key = md5 encrypted string of user’s email, first name, and last name. By using all three fields that are unique in ‘user’, it ensures the activation key will not be duplicated when sending to each registered user.

After the user clicked the email activation link, it will be redirected to UberKidz website to update his/her status from 0 to 1 and clear the email\_activation\_key field. One benefits of this approach are to prevent massive bot account created by automation software. Another benefit is ensuring that the user isn’t using a fake email to hack inside the UberKidz system.

### Remember Me Feature (Via encrypted cookies)

When the user submits the login data, the posted details will be evaluated and validated with respect to the User database. For successful login attempt, users with Remember Me checkbox selected will store their logged in status and user information such as email address inside PHP session and cookies. In order to prevent security leak, such as directly storing a plain password inside Cookies (in which malicious user can manually change cookie content to bypass authentication system), a selector-validator system is used to ensure data security.

For users that clicked Remember me, three cookie entries will be stored: User Email Address $\_COOKIE["member\_login"], a random generated password $\_COOKIE["random\_password"] and random generated selector $\_COOKIE["random\_selector"] based on crypto random number generator. The selector will then be hashed and stored into the loginAuth database, as a validator. A default expiry period of 1 month is also stored into the database so that the Remember Me login option will by default have one month of validity. After user closes the browser and reopen UberKidz website, the PHP script embeds in the login page will first examine whether there are corresponding cookie entries stored in the user’s browser. If there exists, it will firstly put user email and a random password on the login form (similar to the autofill function), then hash the $\_COOKIE[‘random\_selector’] and use password\_hash to verify the entry stored in the loginAuth table. Once successfully authenticated, it will directly allow the user to login without the need of entering the password manually.

### Cross-Site Request Forgery (CSRF) Defense: Token-Based Form Authentication

For most forms that relate to sensitive information process (i.e. Login, Registration, etc.), a token-based form authentication is implemented to prevent potential CSRF attack. A random number encrypted via md5 is generated before a critical form is submitted, and this encrypted token is stored into the session variable (i.e. $\_SESSION[‘token’]) and a hidden input field inside form named ‘token’. After the controller receives a form submit a request via POST or GET, it will first check whether $\_SESSION[‘token’] is set already or not. If not, a new random number encrypted via md5 is generated and stored into this session variable. Followed by this check, it will then check whether $\_POST[‘token’] equals to $\_SESSION[‘token’]. If the request is sent via the actual UberKidz website, these two variables must match, whereas if the request is sent via other websites using CSRF attack, it will not be able to set identical values in $\_SESSION[‘token’] and $\_POST[‘token’] as it will not execute the token initialization before form submission.

### Forgot Password Feature

The token-based password reset system is enhanced via the usage of selector and validator. The selector is a unique ID to allow search in the table ‘passwordreset’, to prevent directly using database id field which will leak the number of users on this platform potentially. The validator consists of two part: 1) plain text stored in the password reset email 2) hashed string via SHA256 stored in the database. After user passes the email validation at the first step, a password reset link will send to user’s email inbox with the attached selector and plain text validator. After clicking on that link, the user will then be able to redirect to the reset page to enter a new password. The validator will be compared with the hashed string stored in ‘passwordreset’ table using hash\_equals() function. The password reset link also comes with an expiry feature, in which this link is only valid within 1 hour after the email was sent.

## Evaluations & Improvements

1. HTML text translation should be implemented to support the usage of end-users with different language preference. In particular, if UberKiz is implemented based on Node.js, it is very convenient to use i18n library for establishing a mapping between database record and displayed texts.
2. The search function for entertainment packages, menus and venues can be implemented in a way that smoother user experience can be provided. For instance, instead of refreshing the page after the user specifies a search criterion, an AJAX call can be used to retrieve the information in the background and update the view correspondingly. However, this implementation is much easier in other programming frameworks such as Angular, in which the view will update itself after the data stored in a variable changed (i.e. using two-way binding). Due to limited time and knowledge about implementation for a similar approach in PHP, this improvement to user experience shall be considered as one primary objective in the future.
3. For entertainment packages, menus and venues product display page, the layout and CSS are basically identical. The best practice, from the team’s perspective, is to use only one PHP file to display all three different types of services based on user selection. However, problems caused by this approach are difficulties in maintenance and debug, as well as service customization. For instance, if the platform host wants to change the layout only for entertainment packages, having separate files for different types of services is a better approach since it is easy to change for a specific type of service while not affecting others.

## Database Design

### Entity-Relationship Diagram

### Design Explanations

UberKidz database design consists of three parts in total, which are account management, product management, and others.

1. **Account Management**
   1. The account management consists of tables including User, Customer, Service supplier, OrderHistory, and OrderDetail. Each user registered inside our system can be either a customer or a service provider. Common information including name, gender, email address, and social information, etc. will be recorded in the user table to avoid duplications. For features that associate with one’s role like the account balance of a customer or company name of a service provider, it is stored in the corresponding table.
   2. For each order placed by a customer, detailed information is stored into two tables named order history and order details, respectively. The order history consists of fields such as order\_ID, customer\_ID, which are designed to map actual items a customer purchased at a given order. The order detail table stored the related information such as price, address and other relevant information customer input for a purchased product. Since this is a 1-to-N situation, the foreign key is placed at the order detail table so to map purchased items to a specific order.
2. **Product Management**
   1. The product management consists of tables including event, menu, minimap, menuitem, entertainmentpackage, entertainmentpackagemap, entertainer, venue. In UberKidz, there are three types of events (disjoint and total constrains), so an event recorded in the database only be one of menu, venue or entertainment package. The event table stores common information shared among all types of events, including name, price, description, image, etc.
   2. A menu can consist various dishes, for instance, two salad for Menu A. However, the item salad might also be a part in Menu B and C. Therefor, the menu and menuitem is N-to-M relationship, an additional mapping table is required to avoid duplications. Similar logic is applied to entertainment package, where an entertainer might be involved in several entertainment packages. Both entertainmentpackagemap and menumap are used for mapping entertainers/menu items to entertainment package/menu.
3. **Others**
   1. **Authentication Management:** LoginAuth, passwordReset.
   2. Voucher, cart and message tables are separate from the main systems. These tables are responsible for storing information that is not connected to other two parts of the main system.

### Normalization

Normalization decomposes unsatisfactory relation schemas into fragments, so it can eliminate data redundancy and update anomalies, preserves dependencies and is lossless. A relation schema, by default, is in First Normal Form (1NF) as all attributes are atomic (single-valued), which means UberKidz is achieving 1NF already. Ideally, a 3NF should be achieved in order to achieve a lossless-join, dependency preserving database design. However, due to limited time and knowledge, the UberKidz database design only achieved 2NF, which means all non-prime attributes are fully functionally dependent on every candidate key. In other words, no partial dependency should be detected in this database design.

In particular, assume a table R consists a set of functional dependencies as F. Then this table R is in 2NF if and only if every functional dependency X🡪A 1) A belongs to X (A trivial functional dependency) 2) X is not a proper subset of a candidate key for R **or** 3) A is a prime attribute for R.

For instance, consider tables of User, Customer, and ServiceSupplier, which are primary tables for holding all account-related information. In the User table, which is considered as the parent of Customer and Servicesupplier, the primary key is ‘user\_ID’ and ‘email\_address’, and there exists one functional dependency (user\_ID) 🡪 (first\_name, last\_name, gender, email\_address, password, contact\_number, registration\_date, avatar, status, email\_activation\_key) and (email\_address) 🡪 (user\_ID, first\_name, last\_name, gender, password, contact\_number, registration\_date, avatar, status, email\_activation\_key). The candidate keys of this table are (user\_ID) and (email\_address), as both of these fields can uniquely identify a row inside this table. For the above two functional dependencies, both user\_ID and email\_address is not a proper subset of a candidate key, therefore achieving 2NF as a result. Similar logic can be applied to other tables, in which all of those tables satisfy the constraints of 2NF.

### Evaluations and Improvements

1. 3NF can be achieved if all transitive dependencies are removed. Specifically, there are some tables like User that are still having transitive dependencies (e.g. Non-Primary Key ‘email\_address’ can also uniquely identify a row), causing the UberKidz database design only at 2NF state. Due to limited time and insufficient knowledge regarding normalization, it is difficult for the team to adjust database structure after already completing most PHP logic and website development. Therefore, to design a better database structure before implementing the website is one lesson learned from this UberKidz project.

## User Interface Design

The final user interface design of the web app highly depends on the wireframes from COMP0035. However, there are some changes compared with the original one. The most significant changes and reasons are justified as following:

### Homepage

The original wireframe contains too many elements in the homepage, including introduction information, 2 navigation bars, sign in and register buttons. The new UI is designed according to minimalism. Only a navigation bar, a simple introduction, and 2 buttons are shown. One button will lead a new customer to the create an account page and another button will slide down the page and demonstrate more introduction.

### Place an order

Originally, all bookable services and their details are put into one single page. In the final design, all bookable services are classified into 3 groups: Venues, Entertainment Packages, and Menus. Each category is accessible through the navigation bar on the top. Secondly, we take the UI design of some other famous online shopping websites such as Amazon as a reference, all services are packaged to be a commodity and only their name, price and a figure are present on each category page. Further details and options are available by clicking the service. The new design improves visualization as it prevents too much information to be presented and accumulate in one single page. In addition, the breakdown design provides more flexibility because a customer may only require one service.

### Check out page

The original wireframe is designed to enable credit or debit card payment. It is not possible to really implement this function. Instead, we give a certain number of virtual currency to each new account for testing. The final checkout page contains simplified details of all purchased items and price. A new function is that on the bottom, there is an input text box which enables the customer to enter voucher and apply coupon.

### Viewing placed orders (customer)

This page is similar to the previous wireframe. One difference is that viewing saved order function is removed because in the new UI design, when the customer chooses a service, it will appear in the shopping cart automatically.

### Viewing placed orders (service supplier)

In the original wireframes, 2 pages were designed for service supplier. For example, as a service supplier, he can log in, view placed orders in details, accept proper ones, and start to prepare for the party. In the final web app, the functions and UI design mainly focus on the customer side. So the service supplier viewing page is simplified. There is only key information shown on the viewing page.

## Quality Assurance

### Black-Box Test

Most functionalities are tested by peers via black-box test, including

1. Registration and Login with extreme cases (i.e. Invalid phone number, email address, password combinations, etc), SQL injections (e.g. " or true--) and XSS attacks (e.g. <script>alert(‘123’);</script> for input)
2. Booking with overlapped timeslot, with duplicate products with same service quality, with no address provided, etc.
3. Checkout with the insufficient fund.

### User Acceptance Test

As the last phase of the software testing process, our team invites three students with experience in e-commerce previously to test our website on its workflow, functionality, and user-friendlessness. Two participants will register as the role of the customer, while one participant will register as the service provider. The service provider will firstly upload three products to UberKidz platform, one for each category (i.e. menus, entertainment packages, and venues). Then the two customers will try to book and add their preferred service to the cart. As the final step, two customers need to perform the checkout process and check the order history in My Account page.

During the UAT, several unexpected scenarios happened, including overlapped booking time, duplicate items in the cart, and invalid login credentials, etc. However, our UberKidz system did manage to catch and handle all these exception cases, either via notifying users or terminate user’s invalid operations. Moreover, the entire workflow of selecting, booking and purchasing services is rated by our participants ‘Comfortable and easy to learn’, which shows the learnability and user-friendlessness of our website. Overall, it is reasonable to conclude that UberKidz passes the UAT and ready to deploy into the real market.

### Database Interaction Automated Testing (DBUnit & PHPUnit)

The first part of automated testing is via a form of database interaction testing, which is mainly used for testing database-related functionalities. Due to limited time and knowledge on DBUnit, our team only developed few automated testing on database-related modules. A sample of the testing is in ./tests/contactTest.php, which will compare the live database record before and after inserting a feedback message to our database.

To perform a round of testing, it is necessary to firstly specify the table structures currently exists in the database (message\_fixture.xml) in a .xml file. For instance, the following code block represents that there exists one record of “thomas” in the ‘message’ table.

*<dataset>*

*<message name="thomas" email="testing1@mailinator.com" message="Message 1"/>*

*</dataset>*

Meanwhile, also specify the table structures that is expected to be after performing the test in the file ‘message\_expected.xml’. In our case, the .xml file should specifies the result after inserting one line of generated message into the database. The total row count in the message table should be 2 after the insertion.

### Automated Testing (Jest & Puppeteer)

The second part of the automated testing is via a form of usability and Interface testing, which is mainly used for testing front-end and PHP logics related functionalities. To achieve such automation, our team decided to use

1. Puppeteer, a Node Library which provides a high-level API to control Chrome or Chromium over the DevTools Protocol (<https://github.com/GoogleChrome/puppeteer>)
2. Jest, a JavaScript Testing Framework allows parallel execution and assertion validations
3. Jest-Puppeteer, a MIT Open-Source Project used for establishing automated testing environment on Node.JS

Our test focused on simulating user behavior and how our website handle each extreme case. For instance, when testing on the Add to Cart feature, we tried cases including adding a service to the cart for the first time, adding duplicate service into the cart, and adding a service with overlapping time with OrderDetail, etc. All these test cases are served as a supplement to the previous User Acceptance Test, to ensure all extreme cases are tested and proved can be properly handled by the website.

# Conclusions

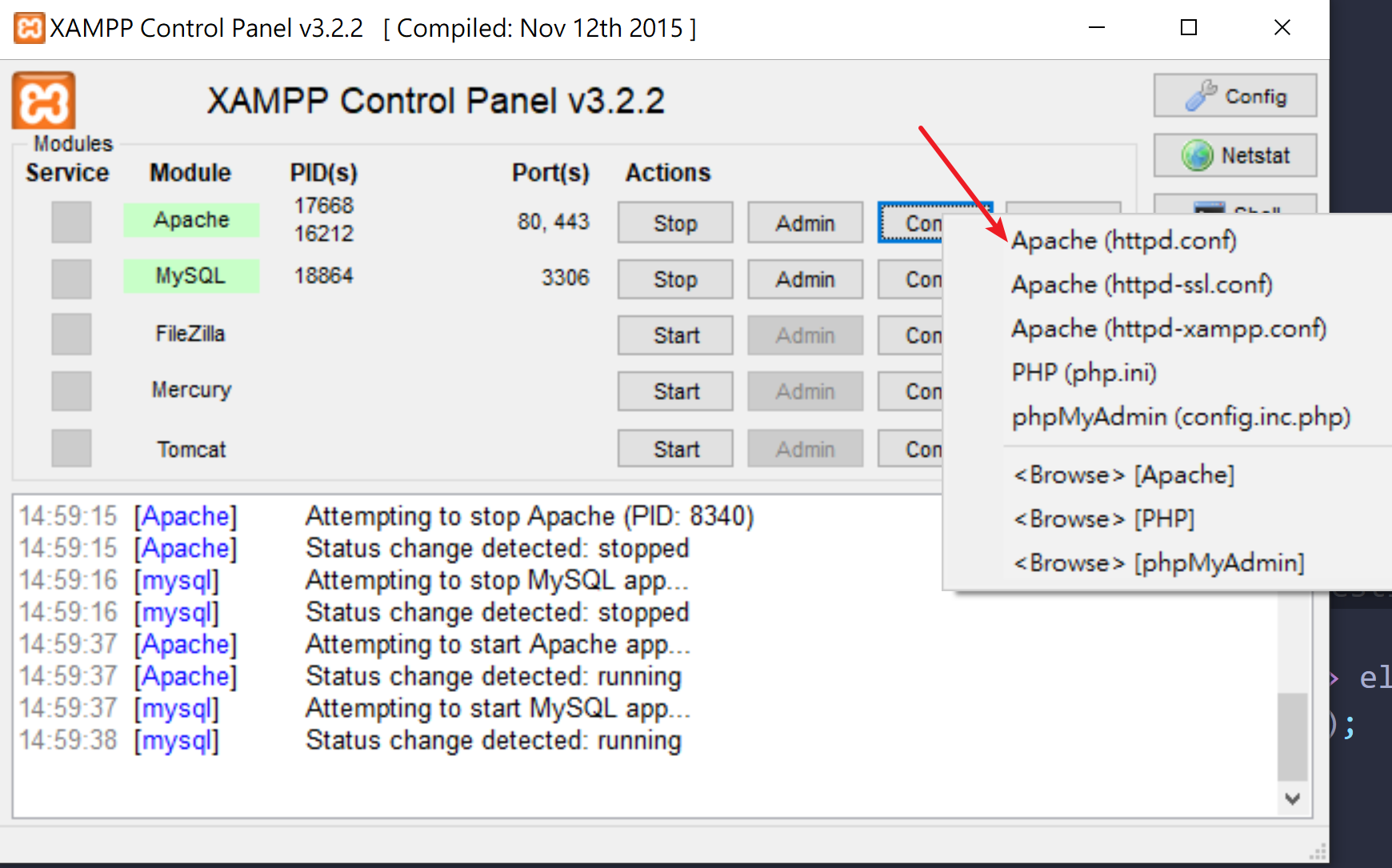
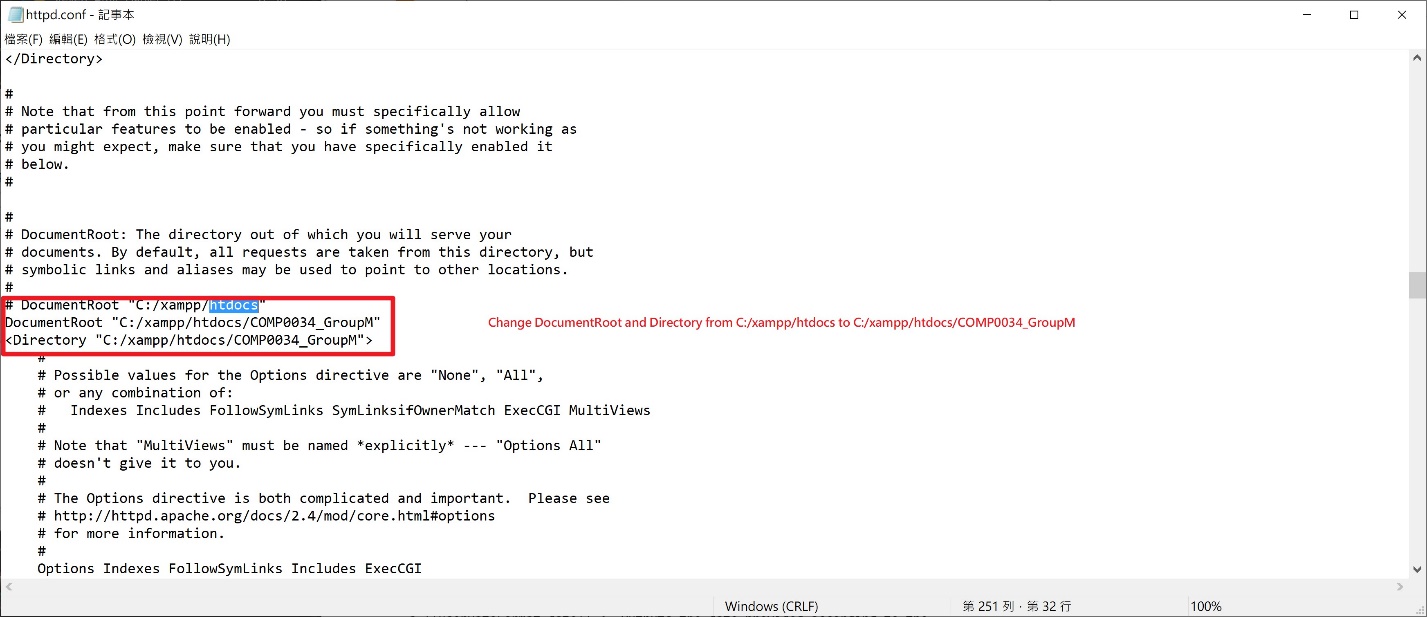
Undoubtfully, our website still requires a lot of effort before it can actually be a commercialized web solution that can be sold into the market. In particular, there are following limitations that can be improved in the future.

1. XSS attack guard. Currently, the system only using the built-in filter function to prevent SQL injection, while the malicious user can insert JavaScript code as the name of the added service to our database. Once this malicious code is parsed by the browser, it will hijack other users’ computer to perform hacker’s tasks, such as stealing sensitive credentials or using their computing power to perform DDoS Attacks. Our UberKidz system can prevent this kind of attack via escaping HTML entities.
2. Comprehensive testing packages, including a fully automated testing script for all main features.
3. CodeIgniter Framework. Although the team tried its best to implement this website in an MVC way, the maintainability and efficiency are still not good enough compared with using professional PHP frameworks like CodeIgniter.
4. PHP Routers. In the current implementation, most redirect is performed directly in the corresponding controllers. However, via using the concept of Router (i.e. similar to the Angular Router concept), it will be easier to maintain redirection behavior in the future when the project is expanding to a larger scale.
5. Only refresh a certain part of view when switching pages in Entertainment Packages/Menus/Venues. Due to limited knowledge of PHP, the team failed to find a way to achieve the effect of ‘Two-Ways Binding’ that is implemented in most other languages (ReactJS, Angular, etc.). There are tutorials about the PHP implementation of this by using pure JavaScript, but the workload is too large for such a small feature.

Regardless the fact that this website still contains certain defects that can be improved, the overall quality of the UberKidz website is, from our team’s perception, fairly good. We provided a professional e-commerce solution to kids’ party planning company, and implement several advanced technologies such as Google Map Location search and display function to facilitate user convenience. Moreover, we provided comprehensive testing solutions on both database and front-end implementations, which can be further served as a regression test to ensure future changes will not affect the core functionality of this website. In addition, protection measures are implemented to prevent most common hacking strategies, including SQL injections, CSRF attacks, etc. Overall, the UberKidz website should be considered as a fully-functional e-commerce website.

# Appendix

## Appendix 1: Deployment on XAMPP

1. Place the entire COMP0034\_GroupM folder inside /htdocs
2. Click the ‘config’ button for Apache, and open file httpd.conf
3. Change the DocumentRoot and Directory from *C:/xampp/htdocs* to *C:/xampp/htdocs/COMP0034\_GroupM.* This step is **necessary** since the way Uberkidz Website include/require files are mostly using $\_SERVER[‘DOCUMENT\_ROOT’] + Path to the File. However, by default, the XAMPP set the DocumentRoot and Directory to the parent folder of the project, which caused incorrect file reference.
4. 

### Install Composer on Windows 10

1. Please follow the instruction listed on <https://getcomposer.org/download/> for downloading composer and set up your PATH environment variable.
2. Use your command line to redirect to C:/XAMPP/htdocs/COMP0034\_GroupM, and type ‘composer install’ for installing all required dependencies for opening UberKidz.

## Appendix 2: Deployment on MAMP (Mac)

### MAMP on MAC (method 1)

**Reference:** *(*[*https://ruifeio.com/2014/08/08/mamp-how-to-change-the-root-directory-on-apache-server/*](https://ruifeio.com/2014/08/08/mamp-how-to-change-the-root-directory-on-apache-server/)*)*

1. Open MAMP
2. Select *Preferences*
3. Select *Apache*
4. You will be able to see the current location of your document root
5. Click on the object next to the ‘*Document Root:*‘ text
6. You will now have a new *Finder* window opened
7. Select the new location for your document root **(/htdocs/COMP0034\_GroupM)** and click on the ‘*Select*‘ button
8. You have now setup the new document root location
9. Select *OK*
10. MAMP automatically stops and starts the Apache and MySQL servers

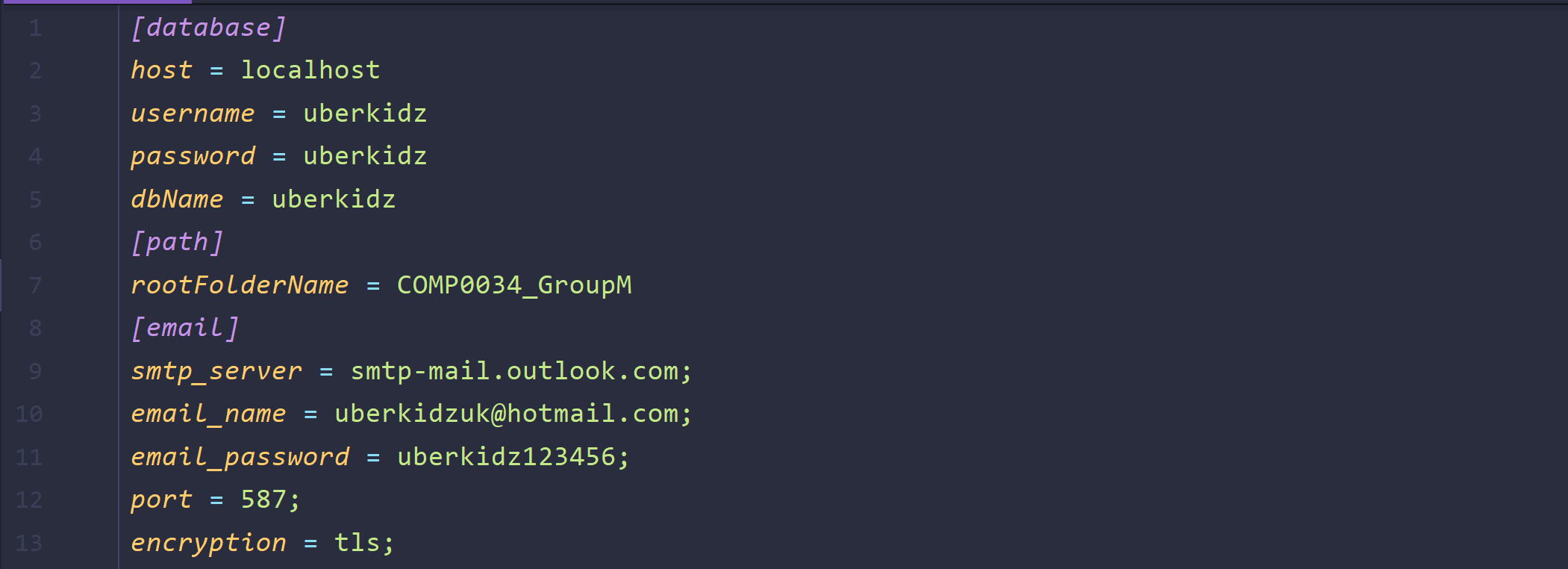
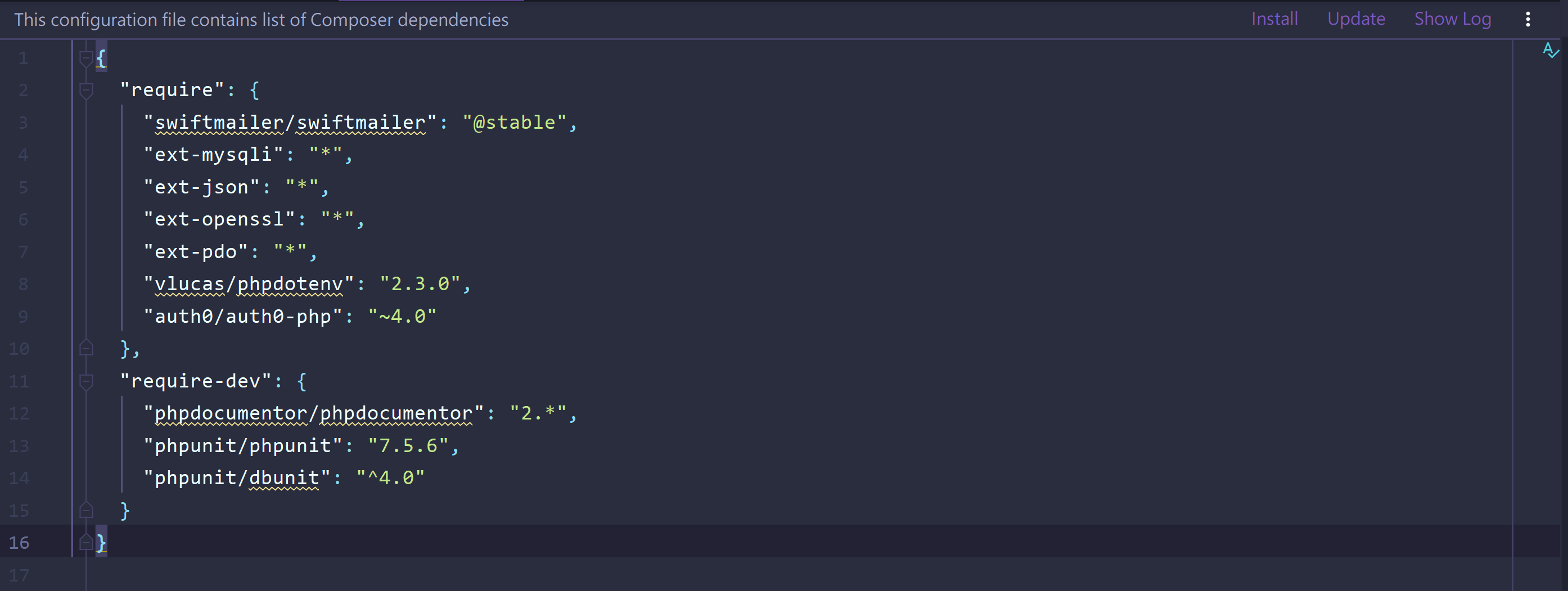
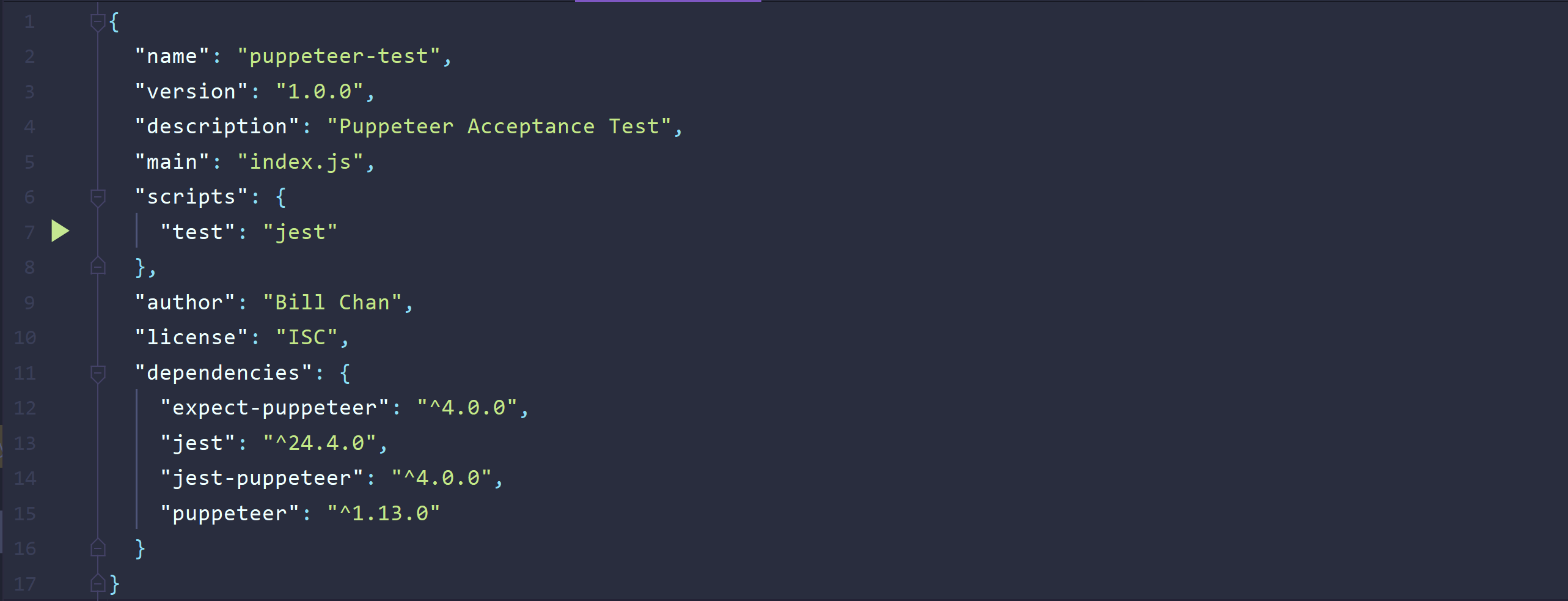
### MAMP on MAC (method 2)

1. Using *Finder*, select your HD device
2. Go to *Applications > MAMP > conf > apache*
3. Edit file  *httpd.conf*
4. Do a find on ‘DocumentRoot’
5. Replace the current path of your DocumentRoot (e.g.: */Applications/MAMP/htdocs*) by your new (e.g.: ***/Applications/MAMP/htdocs/COMP0034\_GroupM***).
6. Do another find on ‘DocumentRoot’ until you get to the line that says “*This should be changed to whatever you set DocumentRoot to.*“
7. Replace the path on DocumentRoot (e.g.:*/Applications/MAMP/htdocs*) by your new path (e.g.: ***/Applications/MAMP/htdocs/COMP0034\_GroupM***).
8. Save the modified *httpd.conf*  file
9. Go to MAMP
10. Select *Stop* *Servers* (if they are up and running)
11. Once the Apache and MySQL servers are stopped, select *Start Servers*

### Install Composer on Mac

1. Open ‘Terminal’, and type ‘brew install composer’. If your machine isn’t equipped with homebrew, please follow the link to install one (<https://docs.brew.sh/Installation>).
2. After successful installed, please open Terminal and use ‘cd’ command to go to /Applications/MAMP/htdocs/COMP0034\_GroupM.
3. Type ‘composer install’ for installing all necessary dependencies required for successfully using UberKidz website.

## Appendix 3: PHP Environment Setup

1. Ensure config.ini consists correct setting to your database. Specifically, ensure the database username and password are correct. The following screenshot is a sample of config.ini used for development purpose. 
2. Ensure both npm packages and composer packages are installed correctly. To install composer packages, type ‘composer install’ in the cmd/terminal at project root directory. To install npm packages, firstly install node.js on the machine, then go to tests/puppeteer-tests, type ‘npm install’ in the cmd/terminal to allow installing necessary testing packages. A more detailed instruction is provided in the video.  

## Appendix 4: User Acceptance Test Form

**TEST ENVIRONMENT:**

**Operating System: \_\_\_Windows 10\_\_\_\_\_ Network: \_\_\_\_\_Localhost\_\_\_\_\_\_\_\_\_\_\_**

|  |  |  |  |
| --- | --- | --- | --- |
| **Form Based Testing Component** | **Pass/Fail** | **Date** | **Initials** |
| 1. Are all fonts, colors, shading and toolbars consistent with standards and project guidelines? | P | 23/03 | MC |
| 1. Is the online help module available? | F | 23/03 | OL |
| 1. Are all date formats correct (DD-MON-YYYY) Are the date edits being correctly applied? Are dates greater than 2000 accepted? | P | 23/03 | OL |
| 1. Does all text wrap when displayed in the text editor? | P | 23/03 | DP |
| 1. Is there a scroll bar on every applicable block? | P | 23/03 | OL |
| 1. Is the Toolbar List button enabled only when a list of values is available for an item? | P | 23/03 | OL |
| 1. Do the window titles correctly identify each module? | P | 23/03 | MC |
| 1. Is there hint text available for all applicable items? | P | 23/03 | MC |
| 1. Do all of the initial 'window display' sizes fit entirely on the screen (assuming an SVGA 800x600 resolution)? | P | 23/03 | DP |
| 1. Are the correct items case sensitive? (i.e. Do fields allow lower case text if they should only accept upper case?) | P | 23/03 | DP |
| 1. Are error, warning and information messages accurate and understandable? | P | 23/03 | OL |
| 1. Do all DELETE operations display a ‘Delete Confirmation’ alert? | P | 23/03 | MC |
| 1. Is the field tab order correct | P | 23/03 | MC |
| 1. Are the appropriate edits done on all fields (range of values, valid values etc.) | P | 23/03 | OL |
| 1. Are defaults appropriate? | P | 23/03 | OL |
| 1. Are the correct fields mandatory? | P | 23/03 | OL |
| 1. Is the tool bar present and appropriate buttons enabled? | P | 23/03 | DP |
| 1. Are screen & field labels appropriate? | P | 23/03 | MC |
| 1. Are fields & buttons ordered appropriately? | P | 23/03 | MC |
| 1. Are all codes valid? | P | 23/03 | OL |
| 1. Are all field labels are consistent across the application | P | 23/03 | OL |

## Appendix 5: Defect Tracking Log

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Defect #** | **Tester** | **Description** | **Date Reported** | **User Level Tested** | **Severity**  **(1,2,3,4,5)** | **Repeated**  **(E,S,O,1x)** | **Date Reported to Developer** | **Status**  **(Open or Fixed)** |
| 1 | DP | NOT ADAPTABLE TO CHANGES REDUCED SCREEN | 15/03 | AD | 2 | S | ES | OPEN |
| 2 | OL | INCORRECT DATABASE CONNECTION | 10/03 | AD | 5 | E | BC | FIXED |
| 3 | ES | INCORRECT PAGE TITLE | 17/03 | AD | 1 | O | BC | FIXED |
| 4 | DP | THE STATEMENT AT THE BOTTOM OF THE PAGE SHOULD BE CHANGED | 09/03 | AD | 2 | O | ES | FIXED |
| 5 | OL | USER NAME CHECK FAILS | 05/03 | AD | 4 | E | DP | FIXED |
| 6 | BC | PASSWORD ALLOWS SQL INJECTION | 10/03 | AD | 4 | O | DP | FIXED |
| 7 | OL | USER NAME ALLOWS HTML INJECTION | 01/03 | AD | 5 | E | BC | FIXED |
| 8 | DP | PASSWORD FIELD IS NOT MANDATORY | 21/02 | AD | 3 | E | BC | FIXED |
| 9 | ES | PASSWORD FIELD TAKES MORE THAN 99 CHARS | 05/03 | AD | 3 | S | DP | FIXED |
| 10 | DP | FAQ PAGE HAS SMALL SIZE TO WRITE DESCRIPTION | 01/03 | AD | 1 | E | DP | FIXED |

# References

### HTML Layout Reference (Derived from following templates)

Bootstrap Navigation Bar <https://getbootstrap.com/docs/4.0/components/navbar/>

Bootstrap Selection Box <https://developer.snapappointments.com/bootstrap-select/>

Colorlib Creative Agency (CC BY 3.0) <https://colorlib.com/wp/template/creative-agency/>

Colorlib Fashe (CC BY 3.0) <https://colorlib.com/wp/template/fashe/>

Colorlib Login Form V1 (CC BY 3.0) <https://colorlib.com/wp/template/login-form-v1/>

Colorlib Product Page Template (CC BY 3.0) <https://colorlib.com/wp/landing-page-wordpress-themes/>

Colorlib Contact Form V12 (CC BY 3.0) <https://colorlib.com/wp/template/contact-form-v12/>

Colorlib Registration Form (CC BY 3.0) <https://colorlib.com/wp/free-bootstrap-registration-forms/>

Colorlib Restaurant Template (CC BY 3.0) <https://colorlib.com/wp/template/restaurant/>

Freebie-footer-templates <https://tutorialzine.com/2015/01/freebie-5-responsive-footer-templates>

Shopping Cart Dropdown <https://github.com/bootstrapstudio/shopping-cart-bootstrap-snippet>

Font-Awesome <https://fontawesome.com/>

### Composer Libraries

SwiftMailer <https://swiftmailer.symfony.com/>

phpdocumentor/phpdocumentor <https://www.phpdoc.org/>

phpunit/phpunit <https://phpunit.de/>

phpunit/phpunit-selenium <https://phpunit.de/manual/3.7/en/selenium.html>

### JS Related

Google Map API (Autocomplete and Embed Map) <https://developers.google.com/maps/documentation/javascript/tutorial>

Modernizer <https://modernizr.com/>

jQuery <https://jquery.com/>

### Testing Related

Puppeteer (<https://github.com/GoogleChrome/puppeteer>)

Jest (<https://jestjs.io/>)

Jest-Puppeteer (<https://github.com/smooth-code/jest-puppeteer>)