
# Support BRANCH

CS 4624 (Multimedia, Hypertext, and Information Access)

Virginia Tech, Blacksburg, VA 24061

5/6/2024

Authors: Akshath Majumder, Luke Marks, Nihar Satasia, Shreyas Sakhalkar

# Table of Contents

# Table of Tables

# Table of Figures

# Abstract

Support BRANCH, formerly known as Mobile Parenting App, is the outcome of a project initially planned for migrating the Treks mobile app to AWS. Financial challenges at Thrust Interactive led to the transfer of data of the Treks App to Virginia Tech. However, further challenges at Thrust Interactive rendered the project defunct. So, Support BRANCH came about to attempt to achieve the same goals as Treks.

Support BRANCH is a web application that helps parents manage behavioral disorders in children, including ASD, ADHD, ODD, and general behavior problems. Parents complete weekly educational adventures that reward parents with badges upon completion.

The technical setup includes a front-end TALL stack—Tailwind CSS, Alpine.js, Laravel, and Livewire—and a back-end LAMP stack composed of Linux, Apache, MySQL, and PHP. Laravel is used as the main PHP framework, with Laravel Blade for building web pages. The environment runs on Docker Compose.

# Requirements

1. Adventures for Users:
    a. Users go through four adventures, unlocked weekly.
    b. The first is unlocked upon registration.
2. Personal Rewards:
    a. Users set a personal reward when they start, which they give themselves after finishing all the adventures.
3. Email Whitelist:
    a. Only certain people can sign up, based on an approved list of email addresses.
    b. There needs to be a way for admins to update or change this list.
4. Daily Check-ins:
    a. If the user has not answered a Daily Check-in in over 24 hours, then they are prompted to answer one before they proceed.
5. Adventures:
    a. Each adventure has five parts, called explores.
    b. Each explore shows information in HTML format.
    c. Explores have prompts, to which users respond with: multiple choice, free response, or free select.
6. Prompts:
    a. Users answer one prompt at a time.
    b. Some questions have correct answers.

# General Approach

Our approach revolves around Visual Studio Code for development, Docker Compose for the application environment, and GitHub [1] for version control. We went with docker-compose because it allows for good portability between development environments. However, the downside is that it runs incredibly slow on Windows machines, but this was remedied by doing most of the development on Linux machines.

Next, we researched content management systems. Given Drupal's reputation for quickly getting functional websites up and running, we initially went with Drupal. However, Drupal is made for passive content consumption (e.g., blog posts), and user interactivity is primarily designed for making content rather than interacting with content. So, we transitioned to a framework called Laravel [3], which provides many features right out of the box, such as its own account management system with registration, login, dashboard, and profile pages.

Once we had our initial workflow set up, we began the development process. For our Docker Compose Application, we initially pulled from a public repository for the LAMP stack [2]. For the containers, we used PHP 8.3 on Apache (Web Server with public ports 80 and 443) [2], MySQL 8 (Database with private port 3304), PHPMyAdmin (DBMS with private port 8080), and Redis (Unused Redis service on private port 6379). Next, we installed Laravel. More specifically, we installed Laravel Breeze, which gave us instant access to user authentication and account management. Laravel uses a Model View Controller design pattern, which makes development much easier. For the front-end, we decided to use Blade [3] and Alpine JS [4]. Blade is used to dynamically create HTML and JavaScript data by using PHP data sent from Controllers. Alpine JS makes elements reactive by having them and their child elements reference data variables. These points of data can be sent to Blade components. Alpine is also

responsible for handling requests to and from the web server for things like prompt submission and data retrieval. Finally, we decided to use Tailwind [5] for styling, which gave us much more flexibility than classic CSS.

While our team mainly handled app development, the development of the actual content was handled by Stephanie Pham, a doctoral student on the client team at the Child Study Center (CSC), who worked on most of the artwork assets, and Benz Huynh, who worked on getting the initial HTML set up, creating the JSON files for importing adventure data into the database, leading the AWS deployment, and putting together releases.

Our AWS deployment approach is very straightforward. We are using AWS for deploying our website, and we have an EC2 instance. The EC2 instance has an email server for us to facilitate email-related functions such as password reset, so we launch an Amazon EC2 instance that will serve as the website's server. This involves selecting an appropriate instance type based on the expected traffic and resource usage. We then use a shell script to package the website into a zip file, which is transferred to the EC2 instance and unzipped and deployed. Finally, we configure the EC2 instance's security group to allow traffic only on necessary ports—typically HTTP (80) and HTTPS (443) for web traffic.

# Design

The client has been primarily in charge of the design of this project. They have designed all of the web pages that will be implemented, including the HTML files, CSS files, and some JavaScript files to add more functionality.

As primarily used for educating parents, the website has modules for them to complete weekly. Modules will be time-locked so that the user can only complete one module per week. The modules will be a group of questions that can be either free response, free selection, or multiple choice. The user's responses will be saved, and if they answered incorrectly (for the questions with correct answers), there will be a gentle message saying which one was correct. Badges will be awarded upon module completion.

When the user logs in, there will be a quick daily check-in pop-up they must answer before proceeding to the module selection page. The questions will be the same every time but will appear every time the user logs in if they have not answered it already that day.

Aesthetically, the website's design will resemble a tree. The module selection page will be a tree; each module will be a branch on the tree, and each question will be an acorn on the branch. The module selection page will have a parallax background depicting a tree rooted on the banks of a pond, with a hill in the back.

# Implementation

The project is implemented using a Docker Compose application of the LAMP stack [2]. LAMP stands for Linux, Apache, MySQL, and PHP. It's a very common stack for web applications. To make the web server more robust, composer was installed to manage dependencies, and Laravel was used as the main content management system and PHP framework.

The benefits of using Docker Compose are portability, security, and simplicity. It's easy to know what libraries, applications, etc., are in the container because of the specifications in its respective dockerfile. Then, Docker Compose allows multiple containers to communicate with each other in an isolated manner. Each container is considered a service, and they can connect to each other by using their service names as hostnames. Additionally, persistent volumes can be stored in the application's main directory. The MySQL server and Apache server both have persistent volumes. Altogether, this makes deployment very straightforward and allows the development environment to be different from the AWS environment.

Laravel, as opposed to Drupal, provides a robust set of web development tools that we can use right away. This includes built-in support for account management, user authentication, and event handling. We took full advantage of this, which allowed us to speed up the development process immensely.

Security was a major priority before deploying the website on AWS. Laravel and its dependencies must be kept up to date, the passwords must be only privately accessible, any default passwords must be changed, and the application's security must be properly vetted before deployment.

# Testing/Evaluation/Assessment

Our testing strategy was carefully thought out and shaped for the specific task at hand. Since the success of our Mobile Parenting Website is largely based on user experience, the core of our approach involved creating sample adventures with dummy data that would simulate real-world scenarios. This method ensured that we were thorough and that the key functionality was implemented as intended. To further elaborate, we created two types of test accounts with different levels of access. The first was a normal user account that we would use to interact with the dummy data and ensure the website was running as intended. The second account was for an admin user, which would be used to test features like the activity dashboard, and registration controls such as adding and removing users from the whitelist.

On the other hand, our debugging approach was not as methodical, and primarily involved strategic print statements throughout our code. Given that our turnaround for this website was short, we felt it was best to prioritize key functionality over a more robust and structurally sound codebase. Although it was tedious, it was also very simple, providing us with immediate feedback on the specific feature we were working on. This allowed us to isolate and resolve issues quickly, which was our highest priority.

To gain more feedback, another integral part of our testing phase was meeting with our clients and having weekly demo sessions to showcase the website's current state. These sessions were some of the most valuable forms of feedback because we got a different perspective on certain features and design decisions. In particular, our clients had a specific styling vision, allowing us to adjust based on their expectations before it was too late.

Overall, we found that the most effective approach was the continuous process of testing/debugging, gaining feedback, and revising. Ultimately, this method proved to be quite

effective as we continuously refined and improved our website into a stable and functioning platform.

# User's Manual

## Website Walkthrough

Users are first directed to the login page shown in Figures 1 and 2. Upon successful login, a home screen pop-up and a daily report pop-up for inactive users are displayed, shown in Figures 3 and 4 respectively. After submitting these pop-ups, the user will see the home page depicted in Figures 5 and 6. The top of the home page has several clickable buttons, the account settings button takes the user to a new settings page shown in Figure 7, the greyed-out tree button shows the user's progress, as shown in Figure 8, and the last button at the top is the Support BRANCH banner button which will show our mission statement detailed in Figure 9. Now, at the bottom of the home page, clicking an Adventure will result in three potential pop-ups explaining the Explore's status as shown in Figure 10. Once on the Adventure page, clicking an Explore Acorn triggers a pop-up for the user to answer questions as shown in Figures 11 and 12. Upon submission, the user will see the Adventure page again with a new Explore acorn indicating the next Explore is ready to be completed which is highlighted in Figure 13. Finally, admin users will also be able to view and modify settings within the Admin Dashboard shown in Figure 14.

# Figures

Figure 1 – Login page (users are redirected here if they are not logged in and try to use any other route)

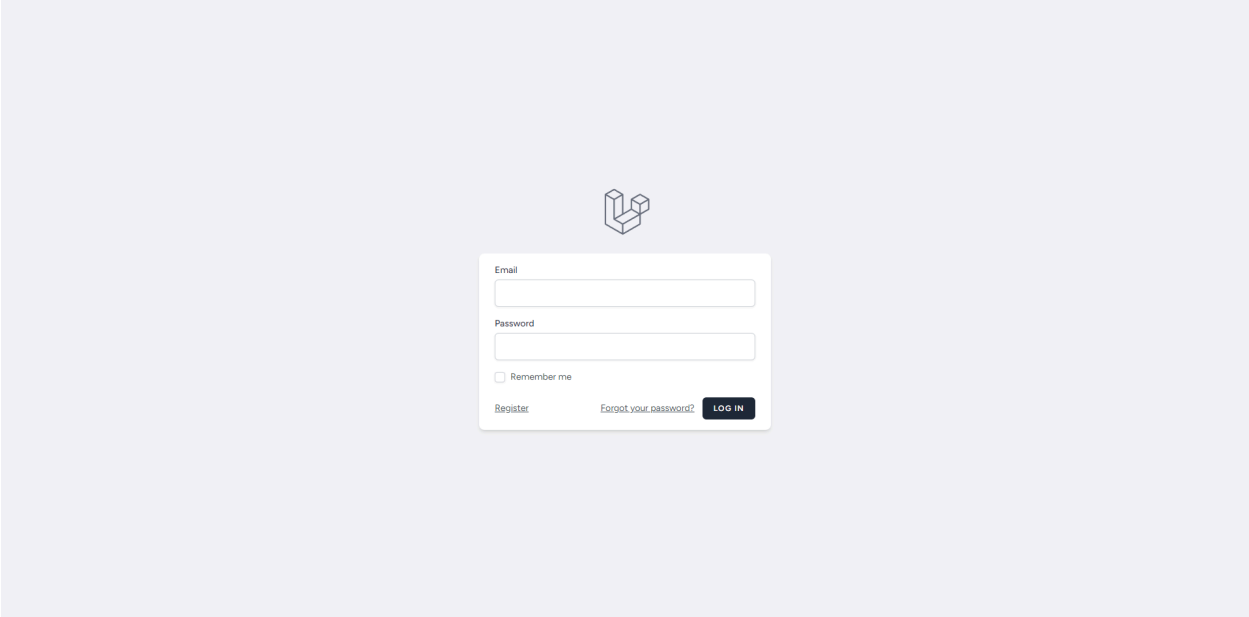



Figure 2 – Registration Page

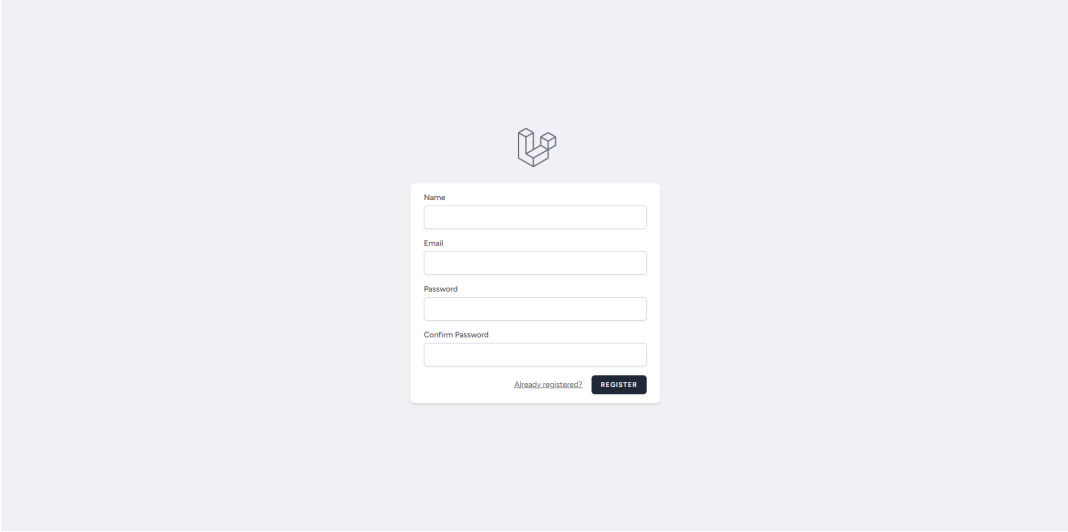


Figure 3 –Home Page with Initial Pop-up

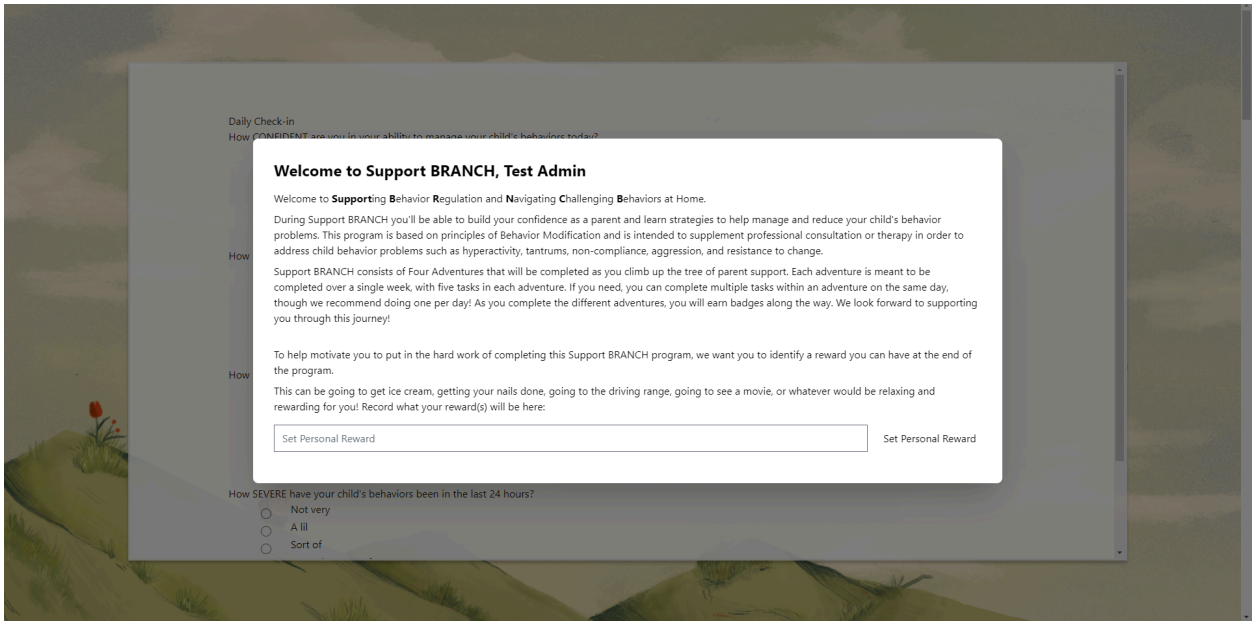



Figure 4 – Home Page with Daily Report (pop up that shows up every time the user has not responded to the daily report for more than a day)

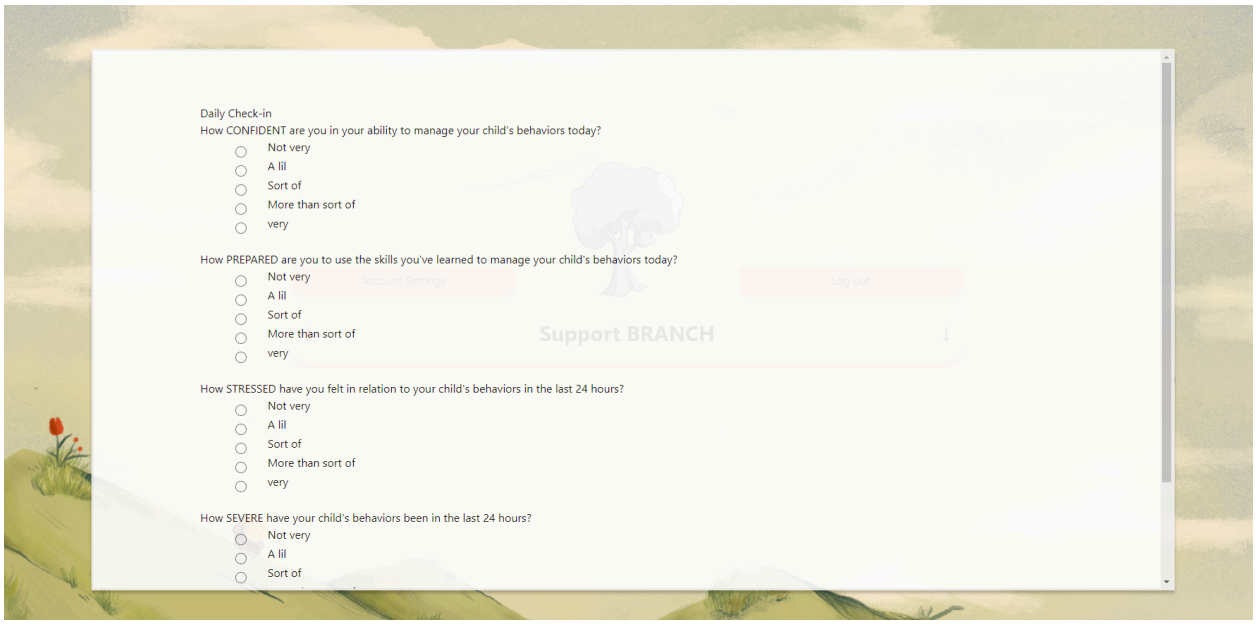


Figure 5 – Home Page Top - The tree becomes more colorful as the user completes more adventures

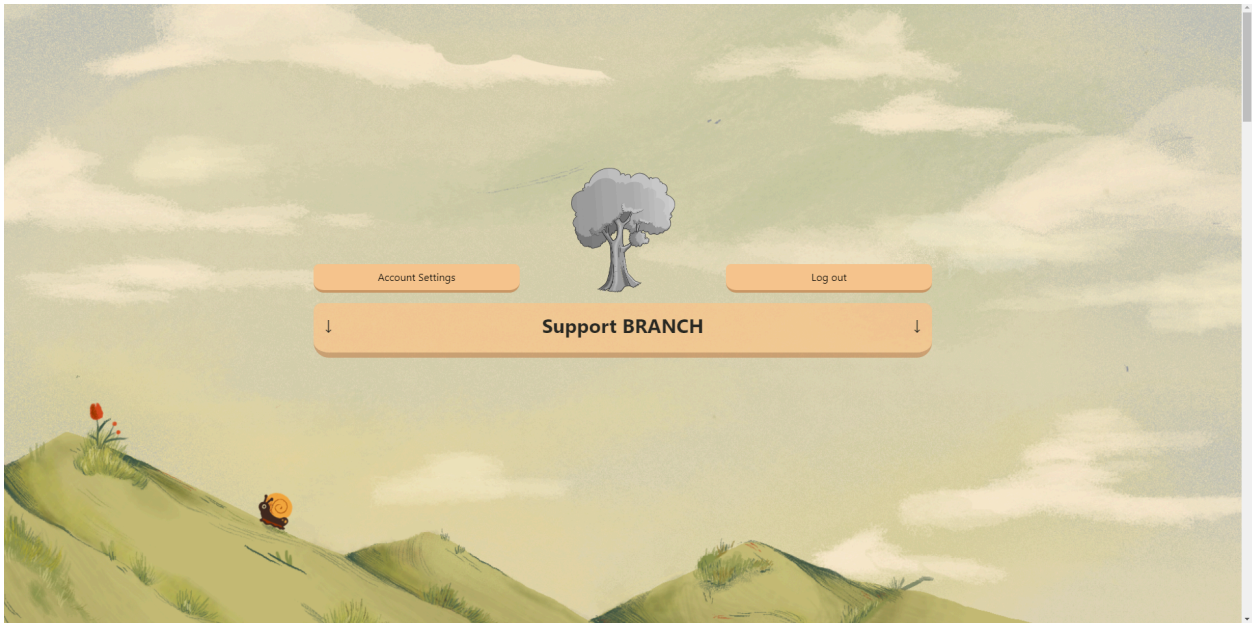



Figure 6 – Home Page Bottom

Figure 7 – Account Settings

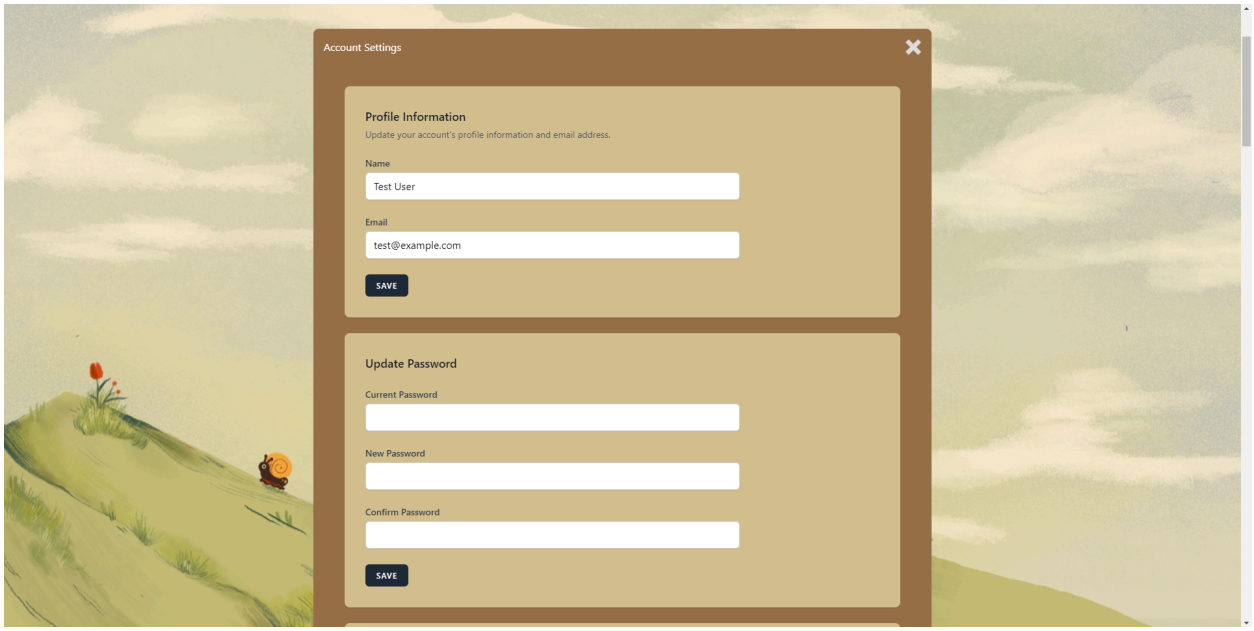



Figure 8 – Progress Pop-Up (opens when you click on the grayed-out tree)

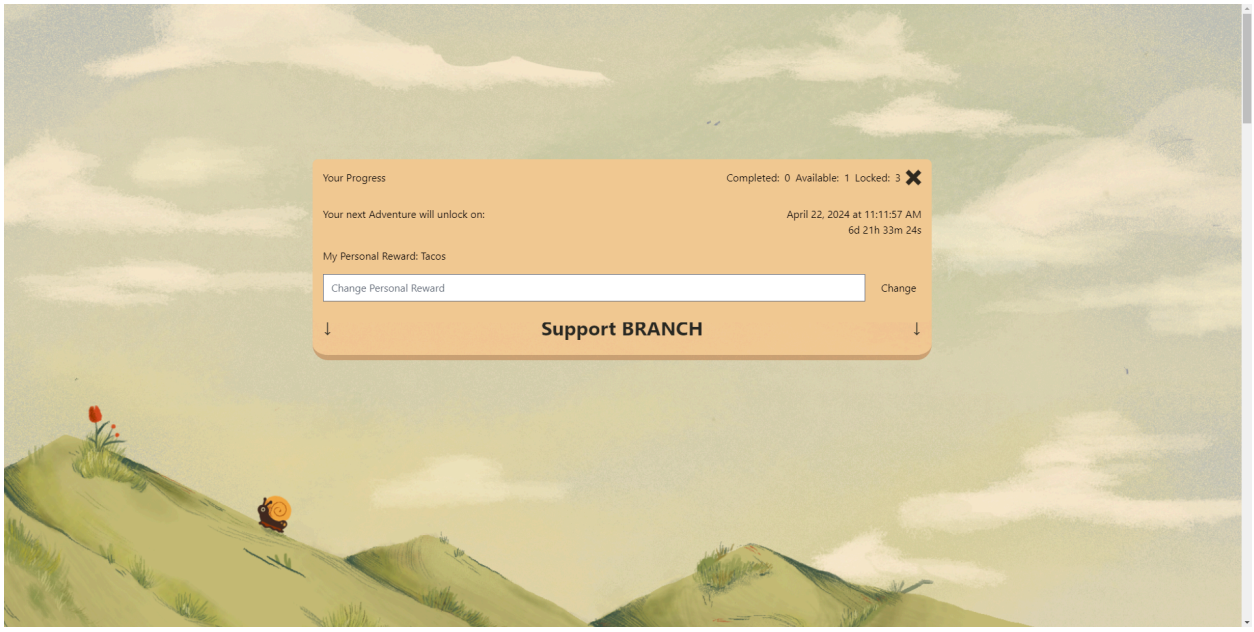


Figure 9 – Support Branch Banner (opens when you click on the banner)

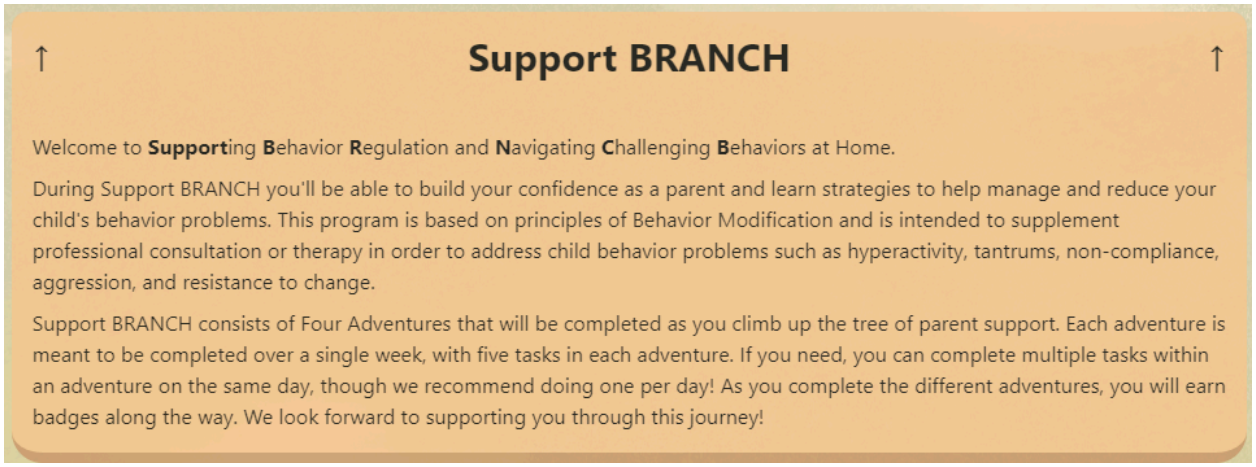



Figure 10 – Adventure Badge Pop-Up (opens when you click on an adventure)

Figure 11 – Adventure Page - Explore Badge acorn one is pulsating, and the arrow on the left says “Go Home” when hovering over it and sends the user to the home page when clicked


Figure 12 – Explore Pop-Up (opens when you click the Explore Acorn badge)

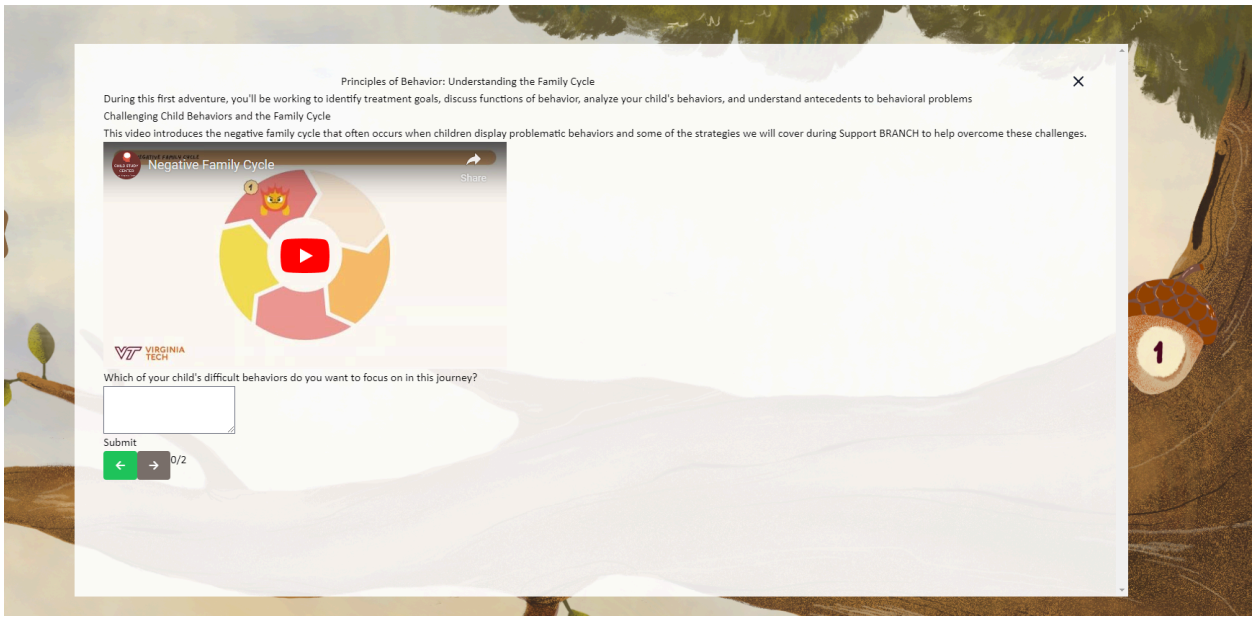
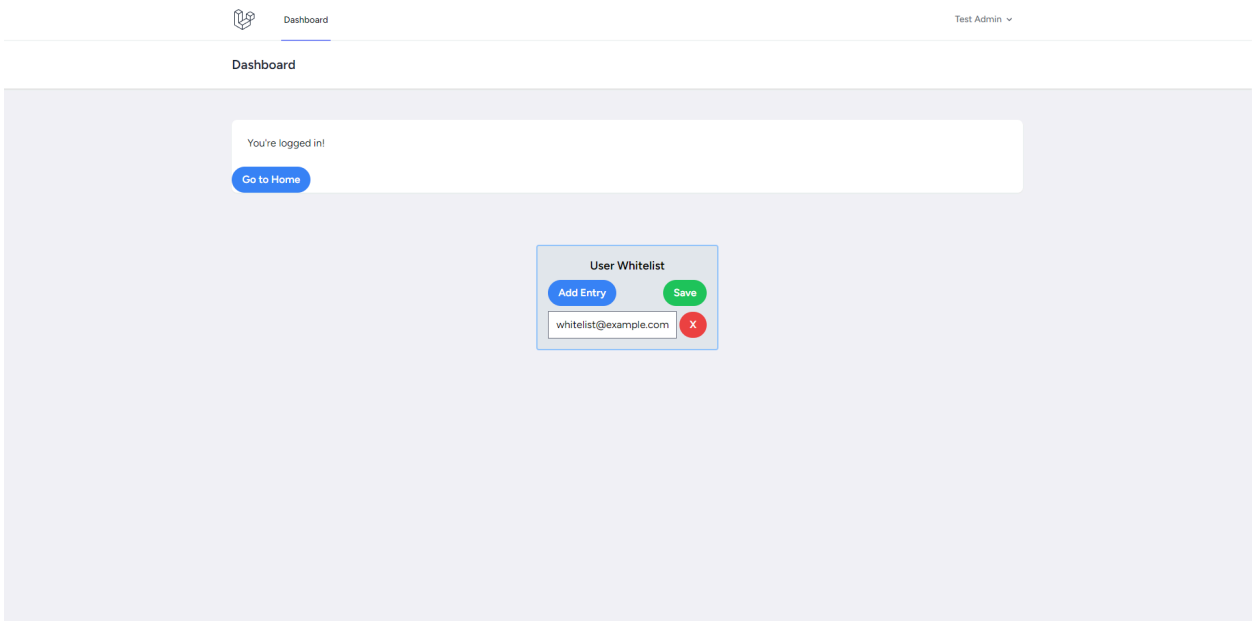


Figure 13 – Adventure Page after Explore completion. Now, the Explore Badge Acorn 2 has appeared and is pulsating instead of the first acorn.


Figure 14 – Admin Dashboard - only accessible to admin users; all others get redirected to the home page



# User Guide

- When you navigate to the Support BRANCH website, you will be directed to the Login page if you haven't already logged in. If you are logged in, you will be directed to the Home page.
- If you have not registered yet, click the Register link on the Login page. Here, you will create your account. Make sure the email (address) you use is the address you gave to the Administrators, as only certain addresses can be used to create an account with Support BRANCH.
    - If you are running into issues, please contact an administrator.
- You will be prompted to enter your personal reward upon logging in for the first time. You may not proceed with the application until you've done so.
- Then, you'll be prompted to answer the Daily Check-in. You must answer all questions to proceed. This pop-up will appear if you open the website and you haven't answered the Daily Check-in in over 24 hours. Note that you do not need to answer the Daily Check-in every 24 hours unless you would like to. This is just here to get you thinking about your progress in Support BRANCH.
- Now, you'll see the top banner of Support BRANCH. Here, there is an account settings button, a logout button, the Support BRANCH banner, and an interactable tree button, which will indicate your progress.
    - You can change your name, email, or password in the Account Settings. Note that if you decide to change your email, this email does not need to be in the whitelist. The whitelist only affects emails for account registration.
    - The logout button will log you out as soon as you click it, so be aware of that.

    - The banner will unravel the same introduction text you saw earlier in the initial pop-up, where you set your personal reward. This is here so you can reference that information whenever you'd like.
    - The tree will become more colorful as you complete more adventures. If you click on it, you can see information related to your progress, such as how many adventures you have completed, how many are available, and how many are locked. You may also see the unlock time and date for your next adventure and a countdown to that time. Finally, you may change your personal reward here if you so choose.
- Next, scroll down through the page and descend into Support BRANCH Valley. Here, you can see the badges of the adventure you must complete. Work your way from the bottom of the tree to the top. The first adventure is unlocked, and each adventure is unlocked a week after the previous one.
- If you click on an adventure page, you can see the title, description, and button to open the adventure. The button is only enabled if the adventure is unlocked. Otherwise, it displays the highest time unit of the countdown to the unlock time.
- Clicking on the button will open the adventure page. Here, you will see an arrow to navigate home on the left side and explore badges on either the right or the left, depending on the adventure. Complete the explores by submitting answers to the prompts. Once you complete all the explores in an adventure, go back to the home page and see the tree become more colorful as a reflection of your progress!
- Once you've completed all four adventures, be sure to reward yourself with whatever you entered for your personal reward!

# Developer's Manual

## Environment Prerequisites

- It is highly recommended that you run this on a Linux machine, and if that's not possible, try to get it running on a Linux VM. Running it on macOS is not possible, and running it on Windows is very slow.
- You must have Docker installed on your machine, and the docker daemon must be running in order to run the web server.

## Setting up the environment

1. Clone the repository locally: `git clone` https://github.com/Ldawsonm/docker-compose-lamp.git
2. Navigate to the repository: `cd docker-compose-lamp`
3. Build the Docker Image: `docker compose build`
4. Turn on the Docker Compose Application: `docker compose up -d`
5. Open the Webserver's bash: `docker compose exec webserver bash`
    a. Note: The docker-compose application has the webserver's service name as webserver, so that's why you put that in the command. This will open a bash session inside the webserver container. You'll be working in this bash session a lot, so be ready to use that command often to get into the container.
6. Install dependencies in the bash session:
    a. npm install

   b. npm run build

7. At this point, the website should be working. You can access it from the web with the address *localhost*.
8. Initialize Laravel Database: `php artisan migrate --seed`
   a. Note: This will prompt if it should create the support_branch_vt database. Say yes to this prompt with the arrow keys and the enter key. This command will seed two test users in the website; here are their credentials Test User: email: test@example.com password: testuserpassword Test Admin: email: testadmin@example.com password: testadminpassword
   *b. You must remove the test users before you deploy.*
9. Access Points:
   a. You can access PHPMyAdmin from localhost to localhost:8080 in a web browser. Do NOT make this port publicly accessible.
   b. The website itself can be accessed by just entering localhost into a web browser.
   c. The MySQL database is technically accessible via port 3304, but it's not recommended to interface directly with the database, as the website and PHPMyAdmin already do that for you.

## Docker Compose Application information

- The docker-compose-lamp directory is where all the application code is kept.
- The scripts directory contains shell scripts for creating releases.
- The bin directory is where dockerfiles for the database and webserver services are kept.
- We are using the dockerfiles in the mysql8 and php83 subdirectories.

- The config directory is where configuration files for the Apache and MySQL servers are kept.
- The php subdirectory is where config info for PHP is for the Apache server, the config file being php.ini. The current settings should be fine for the expected load of the application, but feel free to change these settings if need be.
- The ssl subdirectory is where SSL certificates are kept.
- The data directory is where the database data is kept. Do not make any changes to any files in this directory unless absolutely necessary.
- The logs directory is where logs for the Apache and MySQL servers go, as well as xdebug for PHP if you have that enabled (it is not enabled by default). If there are problems with either of these (specifically the MySQL server since that's more prone to failure than the Apache server), then check these log files.
- The www directory is where all the web server code is kept. This will be the primary directory you'll be working in, as this is where Laravel and all its files are located.
- The .env file is used to define the environment settings for the Docker Compose application. This is where you set things like ports, MySQL usernames and passwords, and persistent volume directories.
- The docker-compose.yml file contains the configuration data for the Docker Compose application as a whole.

## Laravel Codebase

- Paths in this section will be relative to the www directory, so keep that in mind. And all commands mentioned in this section must be executed within the webserver bash.

- PHP Artisan Commands
    - Developers will be using `php artisan` commands for a lot of back-end-related things.
    - To view all php artisan commands, just enter `php artisan`
    - To view the usage of any given command, enter `php artisan {command} -h`. This is especially helpful for the `make` commands.
    - The AdventureJSON directory is where adventure JSON files are kept.
        - You can import the adventure directory by using the command `php artisan adventure:import {PATH TO FILE}`
        - Note that {PATH TO FILE} is a file path relative to the www directory.
        - You can delete adventures by using the command `php artisan adventure:delete {ID}` where {ID} is the identifier of the adventure in the database.
        - Adventure JSON files follow a specific JSON schema. The schema file can be found at database/schemas/adventureSchema.json.
        - The source code for the custom commands are found at app/Console/Commands.
            - One can create a command with `php artisan make:command`.

## Routes, Models, Migrations, Controllers, Views, and Components

- The file routes/web.php contains all the custom routes in this application.
    - To view all routes, use the command `php artisan route:list`

- The directory app/Models contains models made for this application. If a model has the `use HasFactory` statement, it can interface with the database. Laravel uses its own Eloquent system which subdermally links these parts together by using the file naming conventions. For example, the Whitelist.php model file refers to the database table whitelists (which is grammatically weird because the table is really for whitelist entries).
- To create a model, use the command `php artisan make:model`
    - Add the `--migration` option to create a migration file to accompany this model.
    - Add the `--controller` option to create a controller file to accompany this model.
- The directory app/Http/Controllers contains all the controllers used in this application. These are key for facilitating route behavior, as routes call Controller methods for their behavior.
    - To create a Controller, use the command `php artisan make:controller`
    - Controllers are typically associated with Models, though they do not have to be.
- The directory database/migrations contains all migration files for the database.
    - Migration files are used to create the tables in the MySQL database. In this project, you must use these migration files to create MySQL tables. Do not do it via PHPMyAdmin.
    - To create a migration file, use the command `php artisan make:migration`
    - To apply migrations to the database, run `php artisan migrate`

    - To undo the last migration to the database, run `php artisan migrate:rollback`
    - In most cases for development, it's best to use the command `php artisan migrate:fresh` as this will take down all the tables, then build it back up, then apply the seeders (dummy data)  to the database.
        - NOTE: Migration files go in order of their timestamp, from earliest to latest. This is important because if you have a foreign key reference in a table; you must ensure its migration file runs after the table it references.
- Database seeders are found in the database/seeders directory. Most of the database seeding is done in the DatabaseSeeder.php file.
- Views are kept in the resources/views directory. When a Controller returns a view, it uses the syntax 'views.{view file without file extension}'
    - All views are made with Laravel Blade.
    - Views typically use Vite to include app.js and app.css, two files that are used to include JS libraries and Tailwind CSS, respectively.
    - You can find them at resources/js and resources/CSS.
- Components have two files associated with them: the constructor file and the actual view file.
    - The constructor file for a given Component can be found in app/View/Components
    - The view file for a given Component can be found in resources/views/components

- The constructor file is responsible for initializing the data referenced in the component view file.
- The view file is where the HTML and JavaScript of the component are kept.
- Combined with Alpine JS, Blade components can be used to create reactive components and keep your code clean and concise.
- Whenever you use a Tailwind class you haven't used before, make sure to use the `npm run build` command to ensure that the CSS class is included.
- If you need to make changes to the Tailwind config, you can do so in the tailwind.config.js file. Make sure to run `npm run build` after you make your changes, though.
- Laravel has its own .env file, much like the Docker Compose Application does. This is used to set the web application's configuration, such as Application Name, Database Name, etc.

There are many more uncovered parts of the Laravel application. This Developer manual is not meant to teach about Laravel, but instead our codebase. For more information about Laravel, use the online documentation linked in the references.

# Lessons Learned

## Timeline

| **Weekly Timeframe** | **Brief Overview** |
|---|---|
| Weeks 1-3 (1/23/2024 - 2/14/24) | Introduction – Attempted making contact with Thrust and decided to pivot to a website on February 14. |
| Weeks 4-6 (2/14/24 - 3/9/24) | Early Development – We set up the Web Stack and planned on using Drupal. On March 9, we decided to pivot to Laravel. |
| Weeks 7-10 (3/9/24 - 4/12/24) | Proper Development – Majority of the programming work was done. Completed development phase on April 12. |
| Weeks 11-13 (4/12/24 - 4/26/24) | Deployment and Testing – Study participants performed further testing once the website was deployed. |
| Weeks 14-15 (4/26/24 - 5/1/24) | Conclusion – Wrapped up all development and prepared materials for future maintenance. |

## Problems

Throughout the development process, we faced several challenges involving Docker, Drupal, and Alpine JS. More specifically, Docker presented significant performance problems when handling Linux containers. Our initial choice of Drupal proved to be a misstep, resulting in a lot of time being wasted. Additionally, incorporating Alpine JS much later in the development process resulted in having to rely on HTML, which was also time-consuming.

## Solutions

In response to these issues, we moved most, if not all, of our development to be done on Linux machines, which proved to be more effective. Fortunately, Drupal did not set us back too much since we were able to transition to Laravel before any significant progress was made. Eventually, we realized that Alpine JS was a better alternative to the previous HTML forms, resulting in the ability to create adventure forms and dynamically enhance user navigation. Overall, we were able to spot our issues early on, which allowed us to make the necessary adjustments.

## Future Work

Future teams should aim to enhance the features per the client's requests while maintaining the software we have developed. It is vital that all of the software remains up to date and functioning so that the consumers can use this website to the best of their ability.

# Acknowledgments

Rosanna Breaux

- Director of the Child Study Center and CALMER Lab
- rbreaux@vt.edu

Angela Scarpa-Friedman

- Director of the Center for Autism Research
- ascarpa@vt.edu

Benz Huynh

- Web Development Consultant
- bhuynh12@terpmail.umd.edu

Stephanie Pham

- Graduate Student
- phamsn@vt.edu

# References


1. Calmer-Lab-VT. (2024). Support-branch [Software]. GitHub. Retrieved May 1, 2024, from https://github.com/Calmer-Lab-VT/support-branch
2. Sprintcube. (2024). Docker-compose-LAMP [Software]. GitHub. Retrieved May 1, 2024, from https://github.com/sprintcube/docker-compose-lamp
3. Laravel. (2024). Laravel documentation. Retrieved May 1, 2024, from https://laravel.com/docs/11.x
4. Alpine JS. (2024.). Alpine JS documentation. Retrieved May 1, 2024, from https://alpinejs.dev/
5. Tailwind CSS. (2024). Tailwind documentation. Retrieved May 1, 2024, from https://tailwindcss.com/