Technical Documentation for

Moving Easy

By Team Decimal 6 (Group 110)

Version 1.0 - 18/10/2023

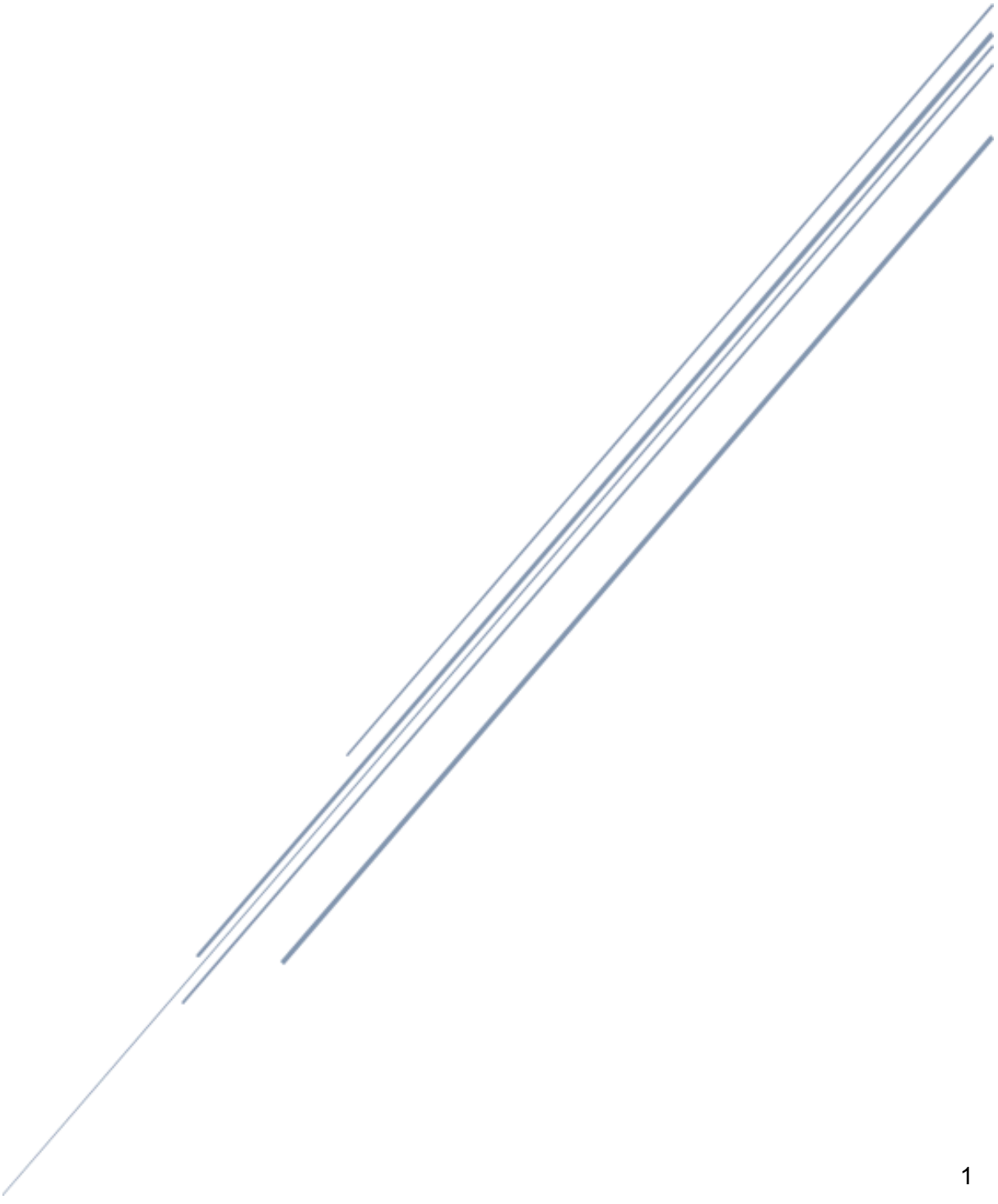


Table of Content

Part 1. About System

1.1 Introduction

This is a web-based system developed by Decimal 6 (Team 110) upon request for the management of Moving Easy. This is a website that helps Moving Easy promote their business, handles customer bookings, and provides a variety of administrative functionalities. The system aims to streamline the customers' experience, and improve efficiency within the business. This system is based on the CakePHP MVC rapid development framework, with customised features and tweekings to fit the needs of stakeholders.

1.2 Skills needed

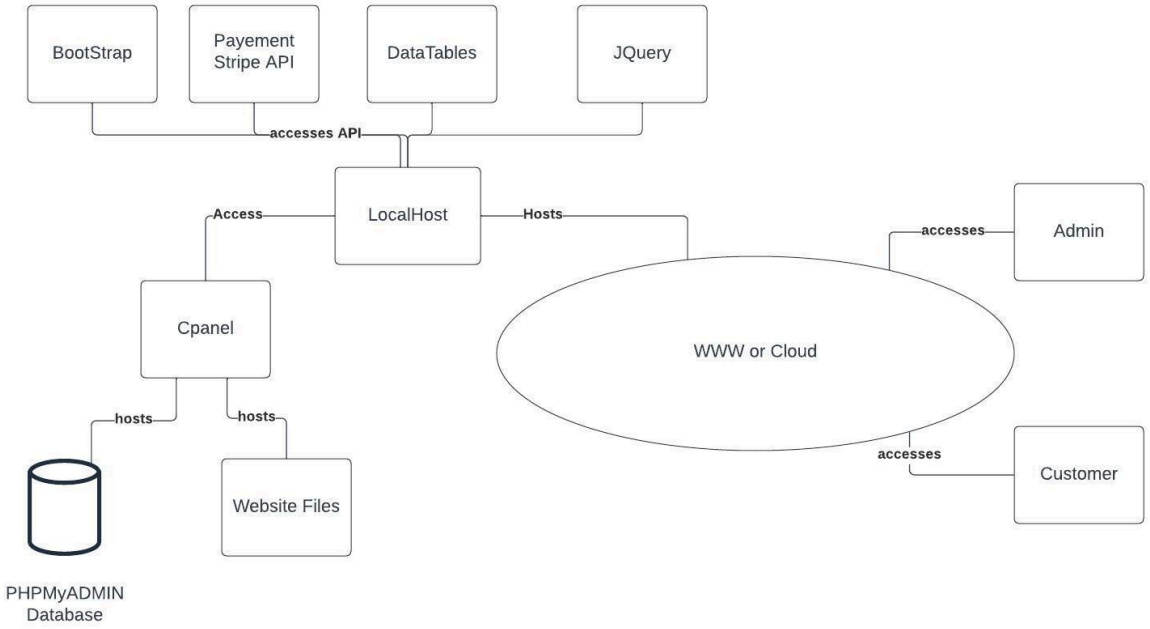
Special technical skills needed to work on this are:

- Web development
- CakePHP
- Sql Database
- API Integration
- Html/Css/Javascript
- GIT

We have used MVC coding design patterns on this project. For summary, it separates the Model, View and Controller. You can see more about MVC herePart 2: System Architecture and Security

Part 2: System and Security

2.1 System Architecture



Above I have shown the system diagram for the current system. Currently the website and Database is hosted through Cpanel. Cpanel primarily creates a localhost for the website to get deployed. All the API requests go through the LocalHost hosted by the Cpanel. Users access the Website through the World Wide Web.

2.2 Security Aspects

2.2.1 Password encryption

Currently all passwords created by the registration of an account are encrypted for security.

2.2.2 Access control

There are currently 3 types of clearance level that determines what sort of content users can see and interact with:

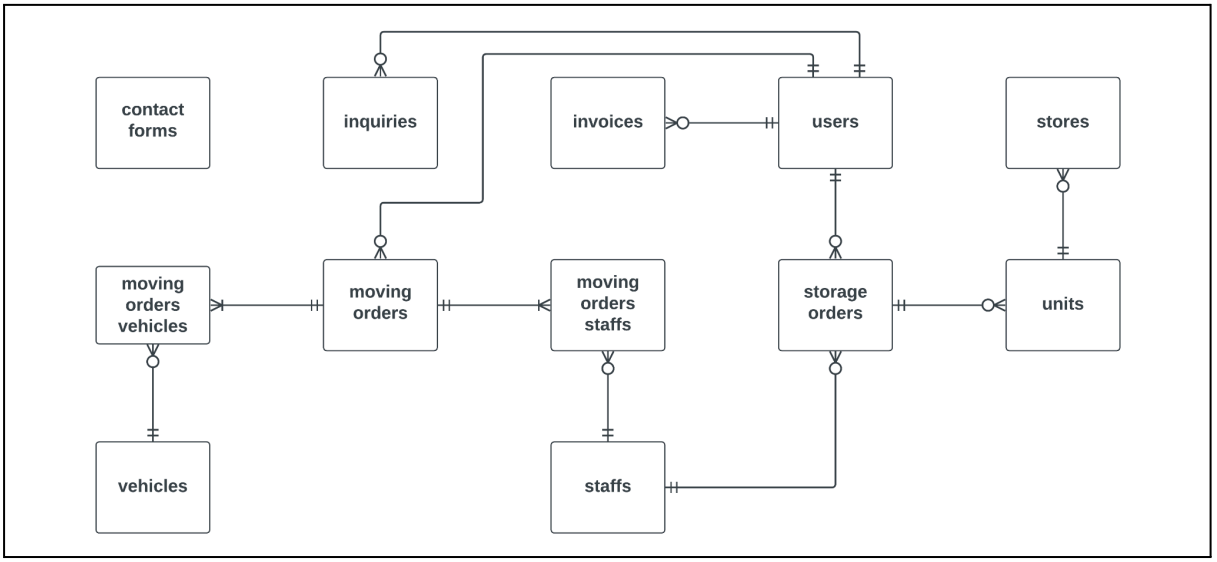
1. **Visitor access:** Visitors will only be able to view pages that are accessible to all users.
2. **User account access:** Users with accounts will be able to view pages as well as the user dashboard and booking pages. Users will be able to make bookings, view order status and make payments.
3. **Administrator access:** Admin accounts will be able to view and edit all pages and information relevant to user accounts such as: Booking orders, Customer Inquiries, View payments and so on.

2.2.3 Data Security

The internal database for the system is isolated from the external database for security purposes. None of the user's personal information is shared with the payment service (only order id and price), and none of the external data is saved in the local database. All inputs are sanitised to prevent injection attacks.

Part 3: Database Documentation

3.1 ER Diagram for Current System



3.2 Data Dictionary

3.2.1 'users' Table

user_name: English name of the client or customer;
email: Used as login credentials of the system;
password: Have to be at least 8 characters with upper, lower and number;
nonce: One time token for password recovery;
nonce_expiry: The expiration time of the recovery token;
user_phone: Australian phone number of the user;
user_type: Used to differentiate individual, business customers and staffs;
user_level: Clearance level to differentiate customer and admins of the system;
user_company: Company's name if it's a business customer;

3.2.2 'staffs' Table

staff_name: English name of the staff;
staff_phone: Australian phone number of the staff;
staff_email: Email address of the staff;

3.2.3 ‘vehicles’ Table

vehicle_rego: registration number of the vehicle;
vehicle_type: enumeration of the 'UTE', 'Small Box Truck', 'Large Box Truck',or 'Dump Truck';
vehicle_model: make and model of vehicle for identification;

3.2.4 ‘contact_forms’ Table

contact_name: name of the visitor;
contact_email: email address of the visitor;
contact_phone: Australian phone number of the visitor;
contact_msg: Messages visitor leave on the form for admin / staff;
contact_replied: Boolean value indicating if the form has been replied by staff;

3.2.5 ‘inquiries’ Table

inquiry_order_time: timestamp recording when the inquiry is made;
inquiry_update: timestamp recording when the inquiry is updated;
inquiry_detail: details customers write about their moving needs;
Inquiry_pickup: pickup location / address provided by the customer;
Inquiry_dropoff: dropoff location / address provided by the customer;
inquiry_start_time: preferred starting time of the move;
Inquiry_end_time: finishing time of the move estimated by the staff;
Inquiry_quote: price quoting provided by the staff;
inquiry_vehicle: estimated number of vehicles needed for the order;
inquiry_staff: estimated number of staffs needed for the order;
inquiry_reviewed: Boolean value indicating if the inquiry is reviewed by the staff;
inquiry_confirmed: Boolean value indicating if the inquiry is confirmed by the client;

3.2.6 ‘moving_orders’ Table

mo_order_time: timestamp recording when the order is made;
mo_update: timestamp recording when the order is updated;
mo_detail: details customers write about their moving needs;
mo_pickup: pickup location / address provided by the customer;
mo_dropoff: dropoff location / address provided by the customer;
mo_start_time: preferred starting time of the move;
mo_end_time: finishing time of the move estimated by the staff;
mo_quote: price quoting provided by the staff;

3.2.7 ‘moving_orders_staffs’ Table

mo_staff_status: enumeration of the status of the staff for this order;
Mo_staff_update: timestamp recording when the order is updated;
staff_id: id of the staff assigned;
Moving_order_id: id of the order the staff is assigned to;

3.2.8 ‘moving_orders_vehicles’ Table

mo_vehicle_status: enumeration of the status of the vehicle for this order;
mo_vehicle_update: timestamp recording when the order is updated;
vehicle_id: id of the vehicle assigned;
moving_order_id: id of the order the vehicle is assigned to;

3.2.9 ‘stores’ Table

store_name: branch name of the self-storage store;
store_address: street address of the store branch;
store_open: opening time of the store branch;
store_close: closing time of the store branch;
store_phone: contact number of the store branch, Australian number;

3.2.10 ‘units’ Table

unit_no: unit number with in a store branch;
unit_area: unit’s area in square metres;
unit_type: ENUM('Locker', 'Small', 'Medium', 'Large', 'Other');
unit_price: one-month price of the unit;
unit_desc: short description of the unit;
unit_avail: Boolean value indicating if the unit is available;
store_id: database id of the store the unit belongs to;

3.2.11 ‘storage_orders’ Table

so_order_time: timestamp recording when the order is made;
so_update: timestamp recording when the order is updated;
so_start: start date of the storage order;
so_duration: duration of the order, calculated in month;
so_price: price of the storage order, automatically calculated;

3.2.12 ‘invoices’ Table:

reference: The database id of the order linked to this invoice;
reference_type: Enumeration showing if it’s a moving or storage order;
Invoice_amount: The amount paid for the order;
invoice_date: The date this order is paid

PART 4: Setup, CakePHP, and Baking:

4.1 Introduction:

CakePHP is an open-source web framework written in PHP. The team decided to develop the system using CakePHP because it follows the Model-View-Controller (MVC) design pattern and the concept of "Convention over Configuration," meaning that the team can get started quickly (integrated CRUD operations) without needing to configure everything from scratch. CakePHP provides built-in tools for input validation, CSRF protection, form tampering protection, and SQL injection prevention, helping us enhance security. (link to cake php page)

4.2 CakePHP Version

This project is based on CakePHP version 4.4.17, released on August 20, 2023.

4.3 Development Environment

During the development process, the team used a local development environment. The website is hosted / updated on the server when Milestones are reached. Some smaller scale bug fixes and improvements are done directly on the server end. It is recommended that developers make the changes locally and test it out on the local or Review server before pushing it on the main site.

4.3.1 PhpStorm / Visual Studio Code Setup

For setting up the visual studio or PhpStorm, you will have to pull the code from visual studio first, after that you will have to run composer commands. Keep in mind to setup up the folder to be www if you are using mac and htdocs if you are using windows. Then you will be good to go.

4.3.2 Localhost & phpMyAdmin Setup

For setting up phpMyAdmin you will have to download xampp for windows. For mac, you can directly download phpMyAdmin without xampp. phpMyAdmin main use in the project is to provide a mysql database. Here is the full step by step procedure which you can follow.
https://kinsta.com/blog/install-phpmyadmin/

4.4 Setup Steps - in a new environment

Step 1: Go to root\tests\schema.sql
Step 2: Run the schema code in MySQL (or other DBMS) to create all the tables
Step 3: Go to root\config\app_local.php and change the ‘username’, ‘password’, and database’ in ‘Datasource’ to the details of the database you’ve created

Step 4: In the terminal, navigate to the root folder and run the following prompt to include the payment plugin:

```
composer require stripe/stripe-php
```

Step 5: Run the following prompt line-by-line to include the Content Management System plugin:

```
composer require ugie-cake/cakephp-content-blocks
bin/cake plugin load ContentBlocks
bin/cake migrations migrate --plugin=ContentBlocks
bin/cake migrations seed --seed ContentBlocksSeed
```

4.5 Migrating the database

If you want to migrate the entire database to a new location, follow these steps:
Step 0: Lockdown the website so that no data would be altered;
Step 1: Select the database in phpMyAdmin, and select the Export tab, create a new template, select format as SQL and click on Export;
Step 2: Navigate to the new, empty database, select the Import tab, select the file you exported, select Character Set as UTF-8 and format as SQL and click on Import;
Step 3: Go to root\config\app_local.php and update the ‘username’, ‘password’, and database’ in ‘Datasource’ to the details of the new database;
Step 4: Test out various functionalities and methods of the website before deleting the the old database or the exported file;

Part 5: File Structure

5.1 Overall Structure

The overall structure follows the MVC structure's principles.
The /config folder consists of configuration files for the system, where the app_local.php handles the connection to the database and a variety of local configurations.
The /logs folder stores all the debugging and error messages that occur while running the system.
The /src/Controller folder stores all default as well as built controllers that receives user input from the view and implements the business logic, calls methods from the model to either retrieve or update data, then updates the view accordingly. Controllers act as an interface between the Model and the View, and will be discussed in more details below.
The /src/Model folder contains the data structure and business logic of the application, the /Table folder has a set of php files each corresponding to a table in the database.
The /templates folder has a set of sub-folders that each corresponds to a Controller. Each sub-folder has a set of files that handles the task of rendering a page for the the user (CRUD+).
The /templates/layout folder has a default.php file that is used to set styling that applies to all pages of the website, the navigation bar and footer is also located in this file.
The /templates/Pages folder contains a set of front-end pages that are mainly used to display information.
The /test/schema.sql contains the up-to-date database schema for the system, including table generation and test data.
The /webroot folder contains assets used on the website such as images and videos.

5.2 Controllers Introduction

As we are following MVC design patterns, we have used controllers to generate the views and other business logic. Main purpose of the controller is to convert the request parameter into the browser response. You can read more about it here.

5.3 Ways to add a new Controller or Page.

Let's say as a developer you are required to add a new controller. Here are the steps and points to keep in mind while doing that.

- **Step 1:** Go to the src/controller folder and create a new controller
- **Step 2:** Import the library and tableregistry which will be used.
- **Step 3:** The function name should correspond to the file name of linked .php
- **Step 4:** Send the required data from the function

Part 6: Content Management System

6.1 Introduction

CMS Plugin is mainly used to make the static part of the website dynamic, so that administration can change certain parts. It is written in CakePHP and uses composer. Through CMS Admin can make some crucial changes like Website Name, Welcome Message, Contact Details, Customer Review etc.

6.2 Guide on using CMS

The full guide for CMS can be found here. Here is some summary of how it works. CMS uses ContentBlocks library for it to work, therefore we need to load this library inside the AppView.php file. Once it's added, it needs a seed class to create the table to store the static part of the website. For your specific requirements, create a child class of AbstractSeed. The run function should contain all the columns which need to be added inside the table. The table will then automatically be created according to the provided seed. After that, the plugin will create a content-blocks page to change the data of the static website. This is a very brief guide, please go through the guide shared above for more details.

6.3 Guide to converting new static field dynamics.

If you want to add a new field and make it dynamic in CMS. Here are the steps in summary.
Step 1: Add a new field inside the Seed for the new field. Give the default value inside the Seed file.
Step 2: The seed file is linked to html with a unique label or key, use that label in the html, and replace the static field.
Step 3: Rerun "`bin/cake migrations seed --seed ContentBlocksSeed` so that the table is generated again.

Part 7: Payment Gateway

7.1 Introduction

All payments on the website are handled by Stripe. Stripe is PCI-compliant and uses advanced techniques like tokenization to ensure secure transaction processing. The team has chosen Stripe for its robust set of APIs and extensive documentation, which allows for a wide range of custom integrations.

7.2 Integration of Payment

For security reasons, the Stripe API and its remote database is in no way connected to the system's internal database. A middle-file is introduced called link-stripe.php that handles the job of passing data and information between Stripe and the system. Limited data is shared between the internal and remote database. Only id of the order (in UUID format), order_type and pricing / quote is shared with the external source. This is passed on through a query in the MovingOrdersController and StorageOrdersController, with a dedicated query handling the action.
Due to the design structure of Stripe, an order should only have one entry in the remote database, but one entry can have multiple payment details. Inside the link-stripe.php and check-out.php files, the query code that retrieves the order price will automatically choose the newest, most up-to-date pricing and jump to the external page to process payment. After a successful payment, the system would be notified and use the query information to built a receipt for that order, and store it within the system.

Part 8: Deploying website on a new server

8.1 Deployment Overview

Currently are using CPanel as a deploying platform. It provides us with various functionality inorder to deploy any website. The parts which we have used while deploying on the server are PHPMyadmin, FileManager, Terminal, Domains and Git. I will talk more about git in section 9.

8.2 How to Deploy or Change any Deployed Component

Below I have explained how to make any changes or add a new deployment.
Changes to PHPMyadmin: PHPMyadmin is used inorder to manage MySql Database. Inorder to deploy all the tables, you will need to run the table schema here. To host a new website, you will have to create a new username and password.Changing a table field, or any other changes here will also mean you will have to rebake that particular table. You can find more information here.

Changes to FileManager: CPanel has provided FileManager to manipulate, add or remove any hosted file. I won't recommend you to directly modify files in File Manager, as a hosted platform will be linked to the production branch. Changes here will most likely cause merge conflict. Instead, you can change the code in the editor and pull the file to FileManager through the terminal. If you want to host a new website, create a new folder inside the FileManager, and then pull the branch code of the website inside through the terminal.
Changes through Terminal: CPanel terminal will play a very important role, most importantly you can access both git and FileManager through it. If you want to change any code in a hosted website, you will need to update the production branch. After modifying that, come inside the Terminal, and pull all updated code on the production branch.

Changes to Domain: Let's say at a certain point of time you want to change the domain of the website. You will need to go to CPanel, and click on Domain. Follow these instruction:

- Click "Create A New Domain".
- Enter the new domain name.
- Deselect the "Share document root (/home/username/public_html) with "domain.tld"." option.
- Enter the directory where you want the files for this domain to exist.
- Click the "Submit" button

Part 9: Git Code Repository

9.1 Git SetUp And Overview

Currently we are using infotech as a git repository. You can directly clone the main branch and have access to the latest code. Some parts of the code might differ between the production branch and the main branch, as the main branch is mainly suited for localhost, while the production branch is suited for deployment.
Current process for deploying code.
Step 1: Add code in feature branch and test it
Step 2: Merge the code on main branch if feature is complete and tested
Step 3: Merge the code on the review branch for the client to see if the feature is what they are expecting it to be. Pull it on CPanel.
Step 4: Merge the code on the production branch for the customer to use the deployed feature. Pull it on CPanel.

Therefore, we have categorised the branches into 5 different kinds for our development

9.2 Important Branches to keep in Mind

As discussed above in the process, we have categorised the branches into 5 categories during our development.

- **Feature Branches:** These branches are mainly for adding any new feature from the side of the developers. Any client requirements are worked on these branches. Examples of these branches are dev_nishant_cms, dev_kristin, dev_allen, dev_nishant_profilepage etc.
- **Main Branch:** This branch should have all the features which the developer has worked on. It might not be deployed to the server at this point of time.
- **Bug Fix Branch:** Even after testing, there are many bugs which we occasionally find, we separately create a branch for such bugs and work on them. This branch can be copied from any branch. Creating a separate branch ensures that our changes won't affect if we find bugs in the main branch or production branch. Examples of this will be dev_nishant_bugfix, dev_nishant_fixing_conflict.
- **Review Branch:** We have kept this branch to deploy the code from the main branch to review cpanel. Once the client gives thumbs up to achieve functionality, we merge this code into the production branch.
- **Production Branch:** This branch should always contain those code which are bug free and thoroughly tested with the client. The production deployed code is the copy of this branch, so any changes made here will directly impact the look and feel of the website for the customers.

9.3 Some Major Commits

If you have a look at our git repository, there are a total of 127 commits. Going through each will be impossible, here I will briefly discuss some of the major commits which you can revert to if such situations arise.

- Baked Cakephp structure (`9cf5cfc4`):
  This commit is very important because you can see all the initial baked tables which we started with. We have added some more tables, but most of it is baked in this commit

- Updating all the code with new schema (`f2123dd6`)
  After our first table design, we made some major schema changes and baked it again, all the new baked code will be seen here.

- Added Inquiry Part (`dd3528d6`)
  There are many similar commits as above, what this specific commit says is that adding inquiry functionality is complete. So all the code for Inquiry can be found here.

- Auto generated invoice upon payment (`c5c7b12c`)
  We have added the payment and generation of invoice at the end of this commit. One thing to note, some part of payment is achieved in other commits too.

There are many such commits, more than 100, so these were some of the most important ones.

Part 10: Design Specification Followed

10.1 Theme Used

Throughout our website, our main motive was to keep it consistent with the theme, so that customers have a good experience. As a result we have used standard themes throughout our website. Here are the things to keep in mind while extending the website.

Similarity To keep In mind while making any design change.

- All the form buttons have a class called form_button, so any design changes to these buttons can be made here.
- All the Views should be inside container1 class, to keep the bluish background with corner edges.
- Bootstrap is used in all the form for its current look
- We have kept the theme of the website using primarily three colours black, white and yellow. These can be changed from cake.css file
- All the tables use the datatables javascript library to add search functionality.

10.2 Design Software Used

To create our website mockups, we have used figma. Here is the project link.
https://www.figma.com/file/VK6Rcf6IJ5KX3REsUg3Ysp/FIT3047-mid-level-prototype-iteration-2
You can change any design or add any mockup here.

10.3 Design Library And Software.

In our website looks we have used several libraries. Here are some important ones and its uses.

- **Bootstrap:** It is mainly used for making the form and other pages responsive. Here is the tutorial of how to use it. https://getbootstrap.com/

- **jQuery:** It has been used in various button interactions and forms interactions. Here is the full guide to use it. https://jquery.com/

- **DataTables:** It has been used for adding search functionality, sorting functionality in all the tables, Here is the full guide to use it. https://datatables.net/

We have used some templates and design guidelines from various online platforms. Above discussed are the most important ones.

Appendix

A.1 Updating a table

When you want to update a table to include new attributes or modify existing ones, it's highly recommended NOT to rebake the table, as it would overwrite the code and create errors. Instead, after the database is updated, first run "bin/cake cache clear all" in the terminal to clear the cache from the older version. Then go to the Controller \src\Controller\YourTableController.php, Entity \src\Model\Entity\YourTable.php, and Table \src\Model\Table\YourTable.php and implement the changes manually. Make sure you refer to the CakePHP cookbook for guidelines and conventions. If foreign keys are altered or updated, you also need to update the files for the foreign key table.

A.2 Adding a new table

If you want to add a new table to the system, follow these steps after you designed the table:
Step 1: Go to root\tests\schema.sql and paste the CREATE TABLE sql code at the bottom of the file;
Step 2: Run the newly written part of the schema code in MySQL (or other DBMS) to create the new tables;
Step 3: In the terminal, navigate to the root folder and run the following prompt:
bin\cake cache clear all;
bin\cake bake all <your_new_table>;
Step 4: If the new table contains foreign key linking them to existing tables of the system, you have to manually update the Controller \src\Controller\FKTableController.php, Entity \src\Model\Entity\FKTable.php, and Table \src\Model\Table\YourTable.php and implement the changes manually. Make sure you refer to the CakePHP cookbook for guidelines and conventions.

A.3 Backup

Although we aim for delivering a smooth running system that's free of fatal errors. Backups are still planned as part of the maintenance process to prevent data loss and recover from attacks.

A.3.1 Partial Backup

The most crucial information that would directly affect the business, such as the database, should be backuped weekly. The admins of Moving Easy have been trained on how to backup the database on MySQL during their training sessions

A.3.2 Full Backup

This includes everything – website files, databases, configurations, etc. This can be done every month or twice a month. Use FTP tools such as FileZilla to connect to your server and download all the files. In case the admin or IT professionals tasked with this project forgets about it, we recommend that an automated backup should be put in place to prevent human error. Many hosting providers offer automated backup services. Check your hosting control panel or consult with your hosting provider. You can use cloud services like Amazon S3, Google Cloud Storage along with tools or scripts to automate the backup process.

A.3.3 Additional Information and Recommendations

Use off-site backups: Always store a copy of your backup in an offsite location (different from your hosting provider). Cloud storage services, or even physical storage media such as external SSD or USB drives can serve as offsite backup locations. This provides an extra layer of protection against server failures.
Maintaining the backups: Keep multiple versions of your backup. If a problem is discovered after a backup has been made, having versions allows you to revert to a previous state. Keep a reasonable count of copies, clean up very old backups, and periodically test your backups by trying to restore them in a separate environment. This ensures that your backups are both valid and complete.