

# IF3E Project Report

GIRAUX Maxime

HAUTEFAYE Corentin

JOURDAN Lorna

November 17, 2024

## Introduction

During this part of the semester (*i.e.* *A24*) we chose to undertake the first subject which was designing a management system for a travel agency. Thus we decided to create a fictive company called *~Martine Travels<sup>1</sup>~* as a reference to those books in which the latter goes to different places such as the beach, mountain, and so on...

The goal of this document is to explain every feature, and choice made along the way as precisely as possible. However please keep in mind that this document is not "color-blind friendly" as there are many different colored links between tables on the diagrams.

---

<sup>1</sup>Copyright ©Martine Travels 2024

# Contents

<b>1</b>	<b>Subject Presentation</b>	<b>2</b>
1.1	Main Objectives . . . . .	2
1.2	Key Features . . . . .	2
<b>2</b>	<b>Database Design</b>	<b>3</b>
2.1	Entity-Relationship Diagram . . . . .	3
2.2	Database Designer View . . . . .	4
<b>3</b>	<b>Front &amp; Back-end Design</b>	<b>5</b>
3.1	Login & User Authentication . . . . .	5
3.2	Administrator Access . . . . .	5
3.2.1	Overview . . . . .	5
3.2.2	Code explanation . . . . .	6
3.3	Reservation System . . . . .	6
3.4	Other features . . . . .	6
3.4.1	Inactive users deletion . . . . .	6
3.4.2	Payment page . . . . .	6
3.4.3	Reviews . . . . .	7
<b>4</b>	<b>Organization</b>	<b>7</b>
4.1	Version Control System . . . . .	7
4.2	Connection to the Database . . . . .	7
<b>5</b>	<b>Conclusion</b>	<b>8</b>
<b>A</b>	<b>Software used</b>	<b>9</b>
<b>B</b>	<b>How to connect to the database?</b>	<b>9</b>

# 1 Subject Presentation

## 1.1 Main Objectives

The primary goal of this project is to design a *relational database* in SQL in order to manage various daily tasks processed in a travel agency. Staff members have indeed many things to deal with such as clients management, accommodations and transportation bookings (*as packages or not*), payment processing, ... If not organized properly a simple task such as creating a client account could become a hassle and lead to several issues (*e.g efficiency loss, time delays, losing records, upsetting clients, losing contracts with other companies, ...*)

Thus designing a first-rate database system to organize and sort data would allow a company such as *Martine Travels* to be outstandingly fast, efficacious and versatile which would consequently help in gaining customer approval and improve sales.

## 1.2 Key Features

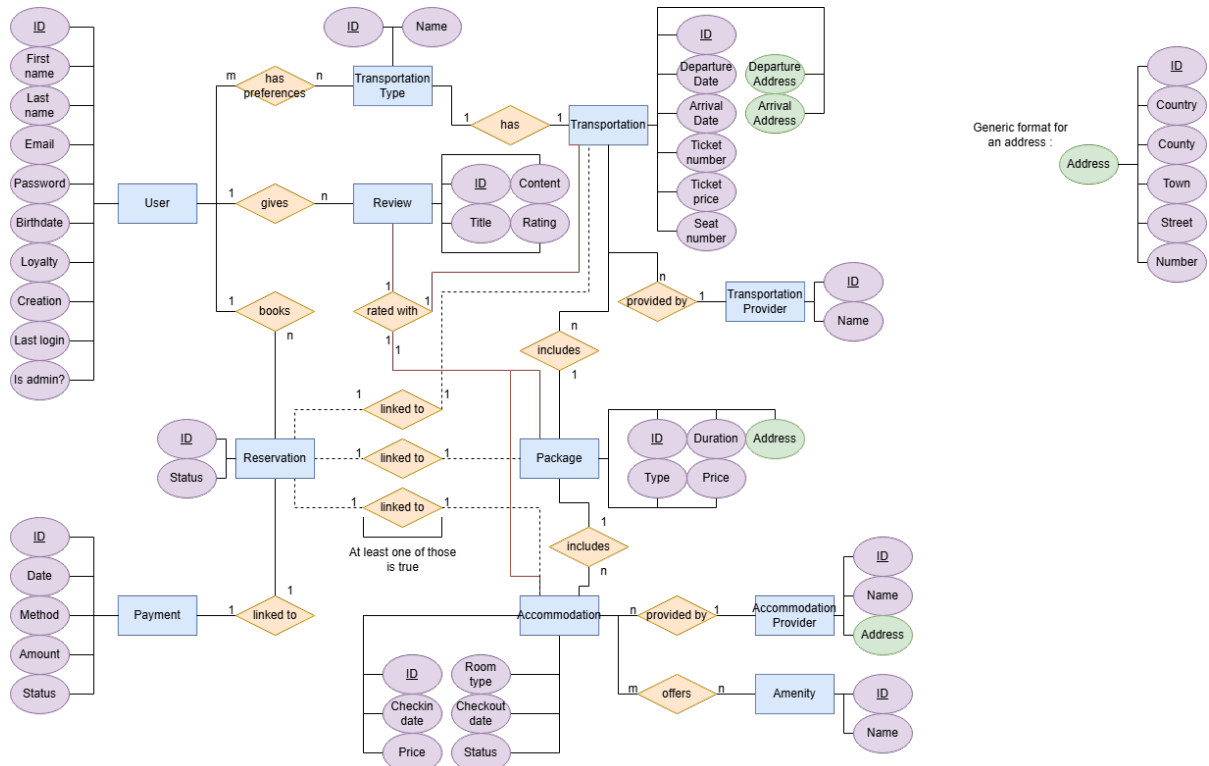
In order to respect the subject specifications we had to design a database model which **would** allow the developer to implement the following:

- Login & User Authentication
- Client Management
  - Detailed record of clients & their information (*e.g name, email, phone, travel preferences, ...*)
  - Account creation, update or deletion
  - Loyalty points based on client history (*i.e allow to offer discounts and/or personalized package*)
- Travel Package Management
  - Various travel package types (*e.g vacation, business trip, safari, ...*)
  - Details about destination, duration, price and itinerary
  - Ability to assign  $n$  transportation and/or accommodation options within a single package
- Transportation Booking
  - Various ways of transportation (*e.g train, car rentals, bus, airplane, ...*)
  - Details about departure and arrival dates, ticket numbers and seat
  - Track transportation providers
- Accommodation Booking
  - Various types of accommodations (*e.g hotels, hostels, resorts, inns, ...*)
  - Details about room types, provided amenities, check-in/out dates
  - Room availability
- Reservation System
  - Ability to book entire packages or individual services (*i.e transportation or accommodation*)
  - Generate booking confirmations

- Allow the user to modify or cancel a reservation
- Payment Management
  - Track payment status (*i.e pending, completed or refunded*) for reservations
  - Include multiple methods (*e.g credit/debit card, bank transfer, cash*)
  - Track client invoices and generate receipts after payment
- Feedback and Reviews
  - Ability to submit reviews for packages, transportation and accommodations
  - Save feedback

## 2 Database Design

### 2.1 Entity-Relationship Diagram



## 2.2 Database Designer View



## 3 Front & Back-end Design

Another important point of this work was to design and program a simple web interface in order to do at least "Login & User Authentication", "Client Management", "Travel Package Management" and "Reservation System" pages. The latter are all explained down below.

The home page (*a.k.a the homepage*) is `site/index.php`. Also in order to offer a wonderful browsing experience to the user on our website, we decided to use CSS Styles to make the website more enjoyable. These can be found under `site/css/`.

### 3.1 Login & User Authentication

A user can sign into their account using the corresponding button within the header bar (*which redirects them to main form of `site/signin.php`*). If one does not have an account they can create one by *signing up*. Then all required information are asked via a form. When trying to submit the latter, verification for data format coherence is performed by looking for regular expressions. For instance we make sure that a password is at least 8 characters long and contains at least one number, upper and lowercase letter. Also we make sure that an email address is only used once (*i.e two users cannot have the same email address*). Yet we do not verify if it truly exists. Also for security purposes passwords are not directly saved into the database but rather a hash string ; we indeed use the `PASSWORD_BCRYPT` to encrypt it.

When the user has been successfully log into their account they are either sent to the admin page if they happen to be one, or are redirected to their account page. Moreover the header bar updates itself and allows the user to log out and modify their settings.

### 3.2 Administrator Access

#### 3.2.1 Overview

At the very beginning, we wanted to make an admin page such that every single table could be modified, be it adding a new record, updating lines with the right datatype, or deleting records. Very soon we discovered that this was not going to be possible for there is quite a number of tables, all of which are joined together. This led to chunks of duplicated code which could not be broken down into `PHP` functions ; this was indeed due to the fact that almost every attribute had a different name whence unlike `SQL` requests.

Thereby we remade the entire page from scratch trying to write functions as it would ease our work. However this soon led to another issue. There are tables in which "secondary" IDs can reference others such as **Package** or **Address**. Thus a request will only return an ID, not the contents it is pointing onto. Hence we wrote a function which would display the latter (*e.g we must show the first and last names of any user, and not just their ID*) and allow the administrator to easily modify any kind of record.

We've also decided to add an extra ID attribute for tables intervening in many-to-many relationships, which we will refer to as "*Join Tables*". This enabled us to perform operations such as *deleting or modifying* on any table without having to take care of every single case.

In order to prevent any staff member to enter invalid data or modify read-only attributes we set some rules down. First administrators are not allowed to directly modify any primary ID. Next forms will prevent anyone from sending data that is either invalid or in the wrong format/datatype (*e.g text, timestamp, integer, ...*). Nevertheless there is no constraint on dates validity and coherence since scheduling is quite the task.

Finally there is a "*super-admin*" which can be logged into using the following credentials: `admin@gmail.com`, `admin`. By the way the reader must keep in mind that the password was so trivial to make testing easier, and to **never** use such weak passwords for any user with higher permissions. Moreover only an administrator can grant a user admin rights.

### 3.2.2 Code explanation

Only administrators can enter the `site/admin/admin.php` page. When one first arrive on that page they are asked to choose a table to modify. This can be done by clicking on the corresponding link. Then they are redirected to a page which displays all records from a table. This is because of a PHP page of which two arguments are passed: the name of the table and the action to perform. The first one is used in `DisplayTableData` which as its name suggests, displays the table. Then the second one is used in `DisplayForm` which allows the admin to modify a record and checks for data format coherence.

## 3.3 Reservation System

Any client can choose to make a reservation (*in their Account page*) whether pre made or personalized. The first one is the easiest as every information has already been entered by a staff member. The second one is trickier because everything must be done by the user and currently there is not enough data in the database to make a whole batch of different new packages.

The page `site/add_reservation.php` asks for all those information and displays dynamically buttons and possible choices (*according to the previous ones*) using JavaScript. This is done through another PHP file which queries data from the database then creates a Json file from it. Next the latter is recovered and processed by a script which displays the new options.

At the end when all required data has been acquired a button labeled "Next" enables the user to finalize the reservation. Finally they can pay it by going on the "Payment" page.

## 3.4 Other features

### 3.4.1 Inactive users deletion

We have added a small scheduled event (*every five minutes*) which looks for users who have not used the website in two years, then deletes the latter.

### 3.4.2 Payment page

When at least one reservation has been made the client may choose to pay one with any of the supported payment methods. When those choices have been made a simple click on the button labeled "Pay" triggers the process.

In real life an other organization such as a bank takes care of money-related troubles ; our website must not store bank details as those are sensitive data. Indeed we should only receive a confirmation of the bank (*or accommodation/transportation provider if using cash*) to mark the payment as completed. Thus we decided to simulate all these steps by a "pay" button.

When it has been done, another button to download the receipt appears. Also because we do not verify for dates coherence, the corresponding reservation is directly marked as "completed" instead of "pending".



### 3.4.3 Reviews

When a travel is marked as "completed", it appears on the home page and users are allowed to leave a review. However there isn't yet any code to display reviews.

## 4 Organization

In order to successfully complete our project and to be more efficient we adopted a simple rule: we made sure that everyone could work on the same file while being at home and without relying on a single person.

### 4.1 Version Control System

Using a version control system such as *Git* was quite useful in our project as it enabled us to track and manage changes to the source code easily. It also allowed us to keep backups of the codebase in case something bad happened (*e.g data corruption or loss, hardware malfunction*) or in case we wanted to try out new features without the fear of breaking something. Besides it made collaboration easier by allowing every member of the group to work on different aspects simultaneously.

We chose to host our code base on *GitHub* as it is free and easier to set up in *PhpStorm*. To access our project repository go [here](#).

### 4.2 Connection to the Database

The connection to the database has also been an important problem. Working separately on the database could indeed yield some interesting results to say the least. That is especially true when testing the front-end part ; one might have had a newer database model while another might not and thus would have had to update it (*all records would have been lost*) to test the website. Therefore we chose to host the database on an UNIX Apache server run from a personal Raspberry PI card.

This solved a huge issue as now everyone could work on the project when they wanted to. Yet that simple action led to two sub-issues:

**Security** We opened a port in our firewall to grant us a remote access to both the Php-MyAdmin page and the database. Yet anyone could also go through that port to steal information which is a great flaw. Thus, we first tried to enable SSL encryption, still it was too difficult to set up. So we set rules down in order to lower permissions for remote access. Moreover "suspicious" requests are denied by the Internet box. All in all this is okay for such a small project however keep in mind that those layers of security are not enough by today's standards.

**Static IP Address** Another problem laid in the fact that we did not own a website domain, nor a **static** address as it cost money. Thus we had to use the IP address of the card every single time. Moreover most of the time an IP address will change about every two weeks. Thus someone had to monitor the latter and change it in `connection.php` so as to ensure the connection to the database.

For more information on how to connect to the database, please see Appendix [B](#)

## 5 Conclusion

In conclusion, this project showed us how databases can be used in daily life operations. We have also realized the significance of designing a thorough and normalized database structure ; indeed it must be comprehensive by anyone who reads it and must limit redundancies.

Moreover this subject is a tangible and realistic situation in which anyone can imagine themselves. As future engineers it is our job to ensure that the whole system works, thus we have a huge impact on the company's success. Also this work taught us to work and collaborate together, to express different ideas, compare opinions.

**Difficulties** We have faced four major difficulties during this project. First of all in our view, the bill of specifications was sometimes too long or complicated. There were indeed too many entities and cases to manage at once, especially with transportation and accommodation providers. Secondly so as to implement some features (*e.g the administrator page*), we had to read a big part of the PHP and Javascript documentation, for which we spent a lot of time whereas we could have used it to enhance some features or add more things. Another big issue we have had is using XAMPP as it was (*and still is*) an unstable piece of software. We've had indeed multiple occasions where we've had to uninstall and then reinstall XAMPP from the system, and it still would not work properly. Thus, the importance of using a mini-server on Linux. Moreover we preferred to use PHP as a standalone software and configure the `php.ini` file which took quite some time. Finally, we think it may be better to have more students per group as there are many things to do. Despite it we realize that the more people in a group the more chaotic it can be.

**Suggestions** Finally we have several things that we could have done to enhance the current codebase/database. After multiple tests, we think that splitting clients and staff member (*e.g administrators*) into two tables might have been a good idea, for an admin should not be able to book a package and then mark the latter as "paid" manually (*as of now an admin does not directly have access to the customers' homepage*). Next we could have stored images for travel packages in the database. Yet we did not have enough to add this feature through the back and front-end, and to make it so the database stays normalized. Finally we could have added a system to display reviews for a package.

## A Software used

- MariaDB
  - Server: Localhost via UNIX socket
  - Server connection: SSL not set up
  - Server version: 10.11.8-MariaDB-0ubuntu0.24.04.1 - Ubuntu 24.04
  - Protocol version: 10
  - Server character set: UTF-8 Unicode (utf8mb4)
- Apache/2.4.58 (Ubuntu)
  - Database client version: libmysql - mysqlnd 8.3.6
- PHP 8.3.6
  - mysqli
  - curl
  - mbstring
  - sodium
- phpMyAdmin 5.2.1deb3
- PhpStorm 2024

## B How to connect to the database?

As of November 17, 2024, the IP address of the server is 90.100.91.219. Also the server will be unplugged on December 1, 2024 at noon. Therefore you may try to access it remotely. However if you may prefer to upload the database on your localhost, there are some variables that need to be updated.

First start your Apache Server and import `martine_travels.sql` into your PhpMyAdmin. Then open the file `site/connection.php` and change its content to (*assuming there is a user root with no password*):

```
<?php
function get_dbhandle(): PDO
{
    $host = "localhost";
    $dbname = "martine_travels";
    $dbuser = "root";
    $dbpasswd = "";
    return new PDO("mysql:host=$host;dbname=$dbname", $dbuser, $dbpasswd);
}
?>
```