**What I did**

**Database Design**  
I have created migration files for a MySQL database that includes two tables: tasks and users. The tasks table comprises fields such as id, title, description, status, user\_id, and due\_date. The users table contains fields like id, name, email, and password. I have also established relationships between the tables, where a user can have multiple tasks.

**Authentication**

For user authentication, I utilized Laravel's built-in authentication features. This includes using Auth::attempt($credentials) to manage user sessions.

**CRUD Functionality**

I developed a simple user interface to manage tasks, incorporating Create, Read, Update, and Delete operations. This interface is built using Laravel Blade Templates for the frontend presentation.

**API Endpoints**

I created RESTful API endpoints for task management, including listing all tasks, displaying a single task, creating a new task, editing a task, and deleting a task. These endpoints leverage Laravel Resources and Validations for efficient data handling. To secure the API, I implemented Laravel Sanctum for authentication, where users receive a bearer token upon logging in. This token is required to access other API functions.

**Unit Testing**

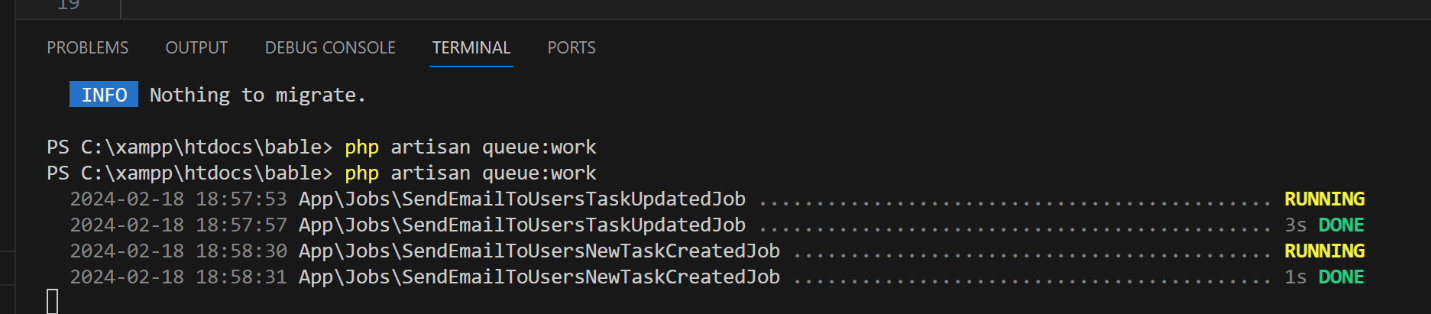
I wrote basic unit tests with PHPUnit to verify the application's functionality. This includes tests for test\_user\_login\_with\_email\_password, all functions in TaskController, and all functions in ApiAuthController and ApiTaskController. To run the tests, simply execute php artisan test. To run the tests

php artisan test

**Middleware Implementation for Role-based Actions** To address the requirement for role-based access control, I created a middleware that distinguishes between normal users and admin users. This middleware ensures that only admin users have the ability to delete tasks, enhancing the application's security and functionality.

**Utilize Laravel Queues**

I set up a SendEmailToUsersNewTaskCreatedJob and SendEmailToUsersTaskUpdatedJob in my Laravel application to send email notifications to all users whenever a task is created or updated. By leveraging Laravel's queue system, I ensured that these notifications are processed in the background. This approach keeps the application responsive by offloading the email sending process to a queue, allowing the main application to operate without delays. The end result is a seamless experience for users interacting with the application, while ensuring all users are kept informed about task updates efficiently and effectively.



**How to setup the system**

**1. Extract the Zip File**

First, extract the zip file to your desired location. This will be the root directory of your Laravel project.

**2. Install Composer Dependencies**

Open a terminal or command prompt and navigate to the root directory of your extracted project.

Use the cd command to change your directory to the project folder:

cd /path/to/your/project

Once you're in the project directory,

run the following command to install the PHP dependencies specified in the composer.json file:

composer install

**3. Create an Environment File**

Copy the .env.example file to a new file named .env. This file contains environment-specific variables. You can do this with the following command:

copy .env.example .env

**4. Generate an Application Key**

Run the following Artisan command to generate a new application key.

This key is used to secure your session and encrypted data:

php artisan key:generate

**5. Run Migrations**

If your project uses a database, run the migrations to create the database schema:

php artisan migrate

**6. Install Node.js Dependencies (if necessary)**

If your project uses Node.js dependencies (for frontend assets like Vue.js, React, or just to compile assets with Laravel Mix),

make sure you have Node.js and npm installed. Then run:

npm install

And compile your assets (if needed) with:

npm run dev # For development

**7. For Laravel Queues**

In .env File set this mailtrap details that i am using for sending email

MAIL\_MAILER=smtp

MAIL\_HOST=sandbox.smtp.mailtrap.io

MAIL\_PORT=2525

MAIL\_USERNAME=f9b9a3653b8ec9

MAIL\_PASSWORD=9a8a37c5f8226a

**8. Setting QUEUE\_CONNECTION to database**

To use the database queue driver, I set the QUEUE\_CONNECTION environment variable in your .env file to database:

QUEUE\_CONNECTION=database

**9. Preparing the Database for QUEUE**

Before the use of database queue driver, you need to prepare your database to store the job queue:

Create the Jobs Table: Run the following Artisan command to create a migration for the jobs table:

php artisan queue:table

**10. Migrate: Apply the migration to create the jobs table in your database:**

php artisan migrate

**11. Running the Queue Worker**

After setting up the database queue connection and migrating your database, you need to start a queue worker to process the queued jobs. Run the following command to start a worker:

php artisan queue:work

**12. Serve the Application**

Open a new terminal or command prompt and navigate to the root directory of your extracted project.

Use the cd command to change your directory to the project folder

cd /path/to/your/project

Finally, you can serve your Laravel application using the built-in PHP server:

php artisan serve

**API Endpoint**

**API for login**

http://127.0.0.1:8000/api/login

{

"status": "success",

"message": "User is logged in successfully.",

"data": {

"token": "1|BHlSqS9h8S6dmDswaFEWUEm4dq6cMoGaE04mtwZ9c9363c63",

"user": {

"id": 3,

"name": "Shoaib",

"email": "shoaib@hotmail.com",

"user\_role": "Normal User",

"created\_at": "2024-02-17T17:46:49.000000Z",

"updated\_at": "2024-02-17T17:46:49.000000Z"

}

}

}

**API for create task**

Post http://127.0.0.1:8000/api/tasks/store

{

"status": "success",

"message": "Task is added successfully.",

"data": {

"title": "C23",

"description": "C23",

"status": "pending",

"due\_date": "2024-02-29T00:00:00.000000Z",

"user\_id": 3,

"updated\_at": "2024-02-17T22:15:00.000000Z",

"created\_at": "2024-02-17T22:15:00.000000Z",

"id": 4

}

}

**API for to show by id**

Get http://127.0.0.1:8000/api/tasks/show/taskid

{

"status": "success",

"message": "Task is retrieved successfully.",

"data": {

"id": 3,

"title": "B23",

"description": "B23",

"status": "pending",

"user\_id": 3,

"due\_date": "2024-02-29T00:00:00.000000Z",

"created\_at": "2024-02-17T17:47:11.000000Z",

"updated\_at": "2024-02-17T22:09:25.000000Z"

}

}

**API for to show all task**

Get http://127.0.0.1:8000/api/tasks/

[

{

"id": 3,

"title": "B23",

"description": "B23",

"status": "pending",

"user\_id": 3,

"due\_date": "2024-02-29T00:00:00.000000Z",

"created\_at": "2024-02-17T17:47:11.000000Z",

"updated\_at": "2024-02-17T22:09:25.000000Z"

},

{

"id": 4,

"title": "C23",

"description": "C23",

"status": "pending",

"user\_id": 3,

"due\_date": "2024-02-29T00:00:00.000000Z",

"created\_at": "2024-02-17T22:15:00.000000Z",

"updated\_at": "2024-02-17T22:15:00.000000Z"

}

]

**API for update task by id**

Post http://127.0.0.1:8000/api/tasks/update/taskid

{

"status": "success",

"message": "Product is updated successfully.",

"data": {

"id": 3,

"title": "D23",

"description": "D23",

"status": "pending",

"user\_id": 3,

"due\_date": "2024-02-29T00:00:00.000000Z",

"created\_at": "2024-02-17T17:47:11.000000Z",

"updated\_at": "2024-02-17T22:16:48.000000Z"

}

}

**API for delete task by id**

Post http://127.0.0.1:8000/api/tasks/destroy/taskid

{

"status": "success",

"message": "Product is deleted successfully."

}