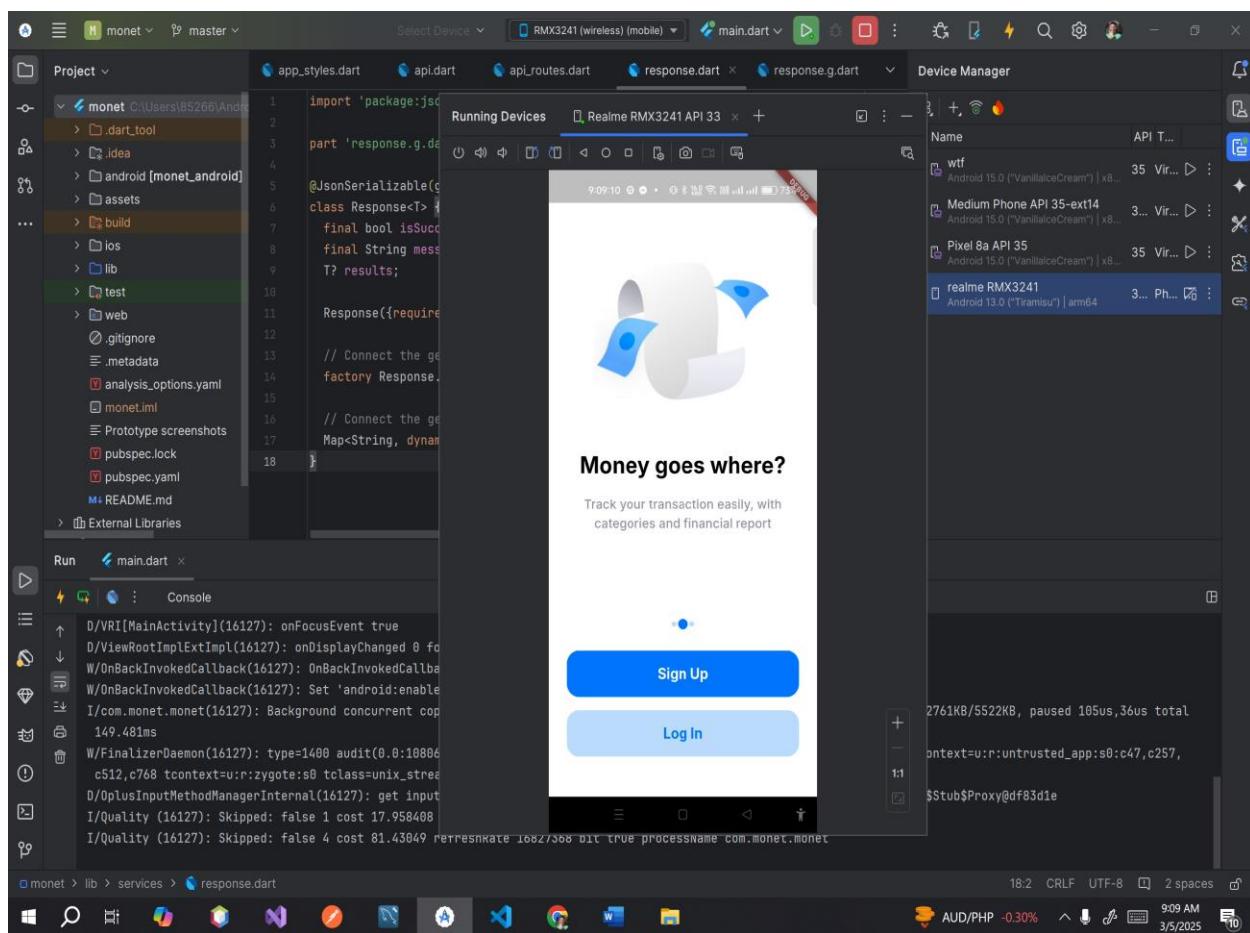
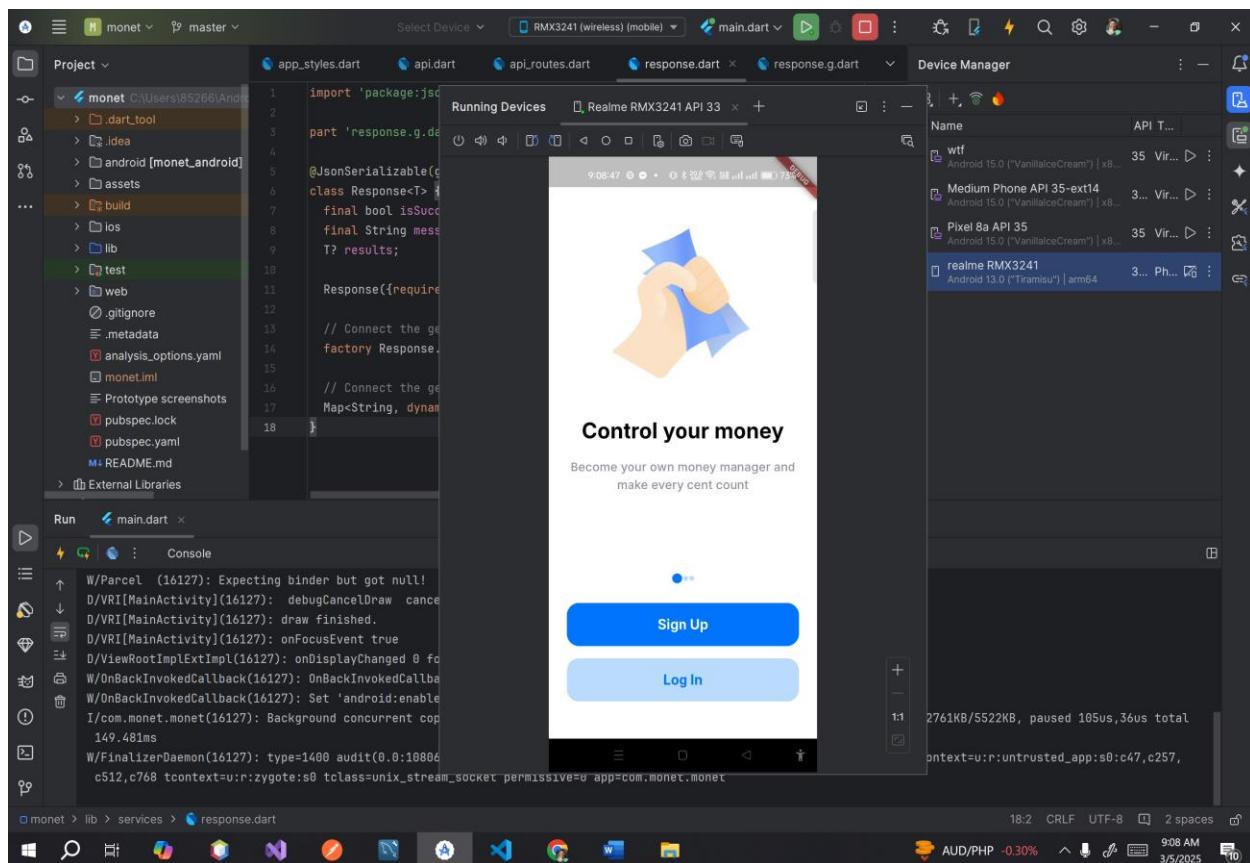
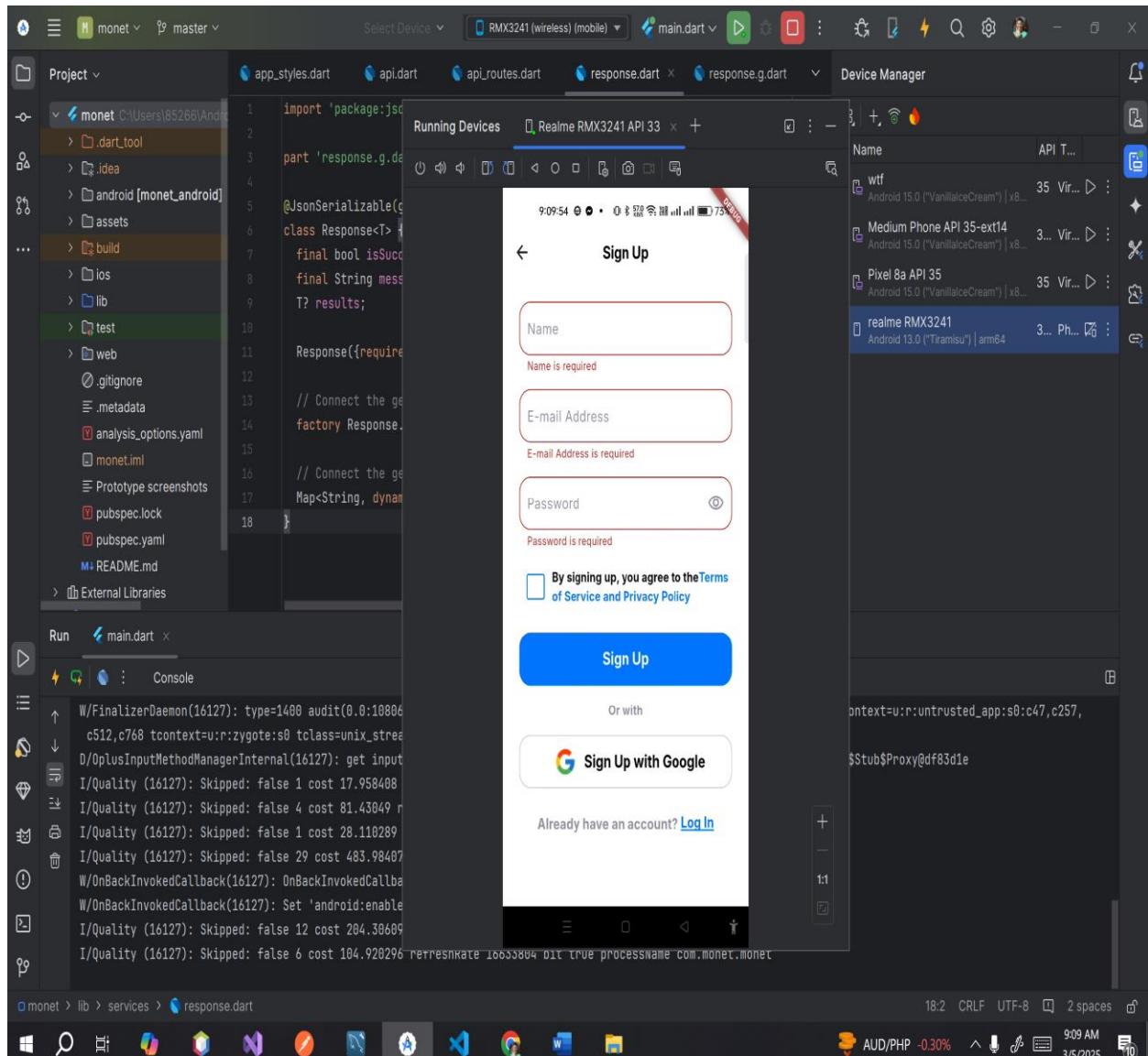
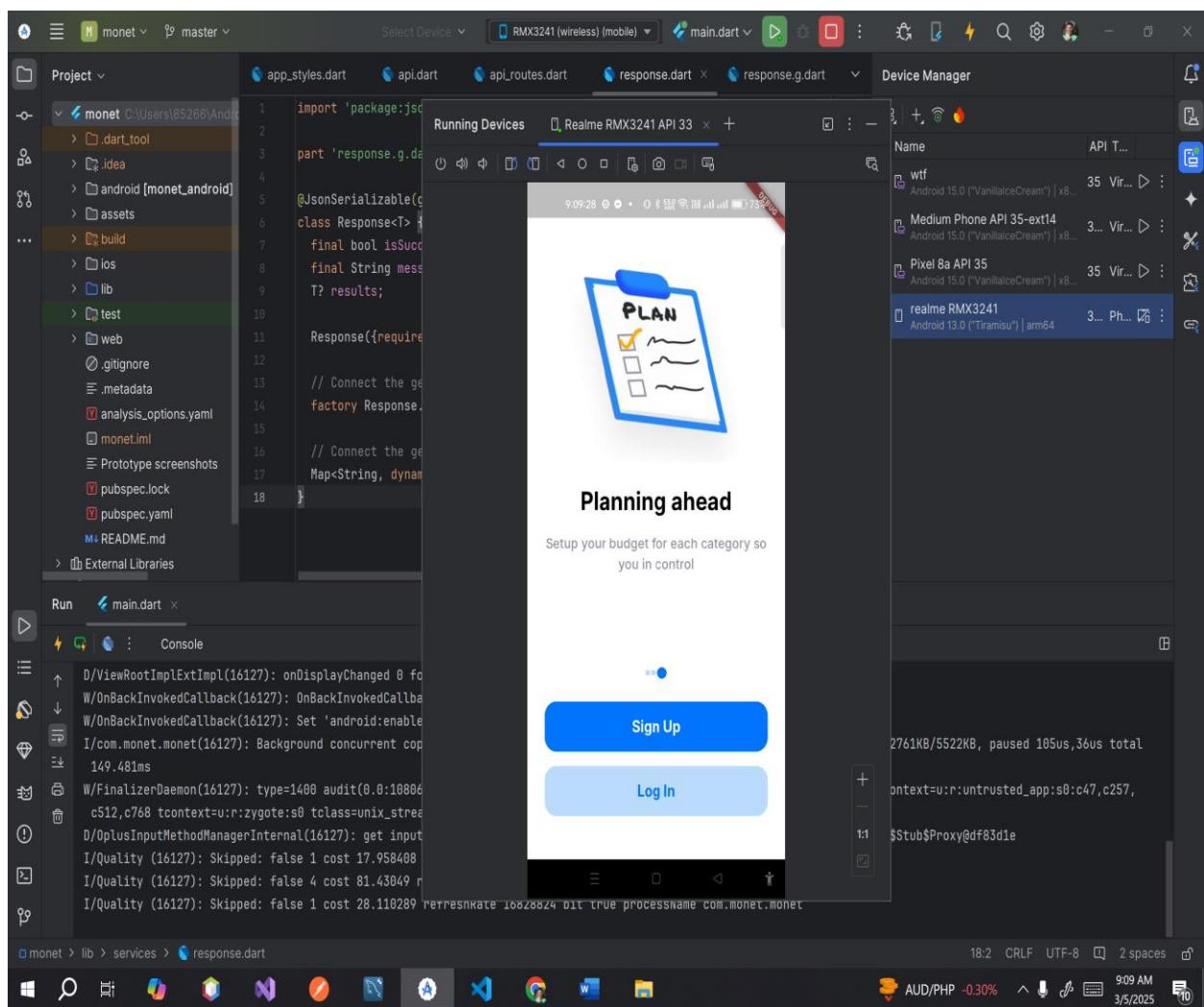
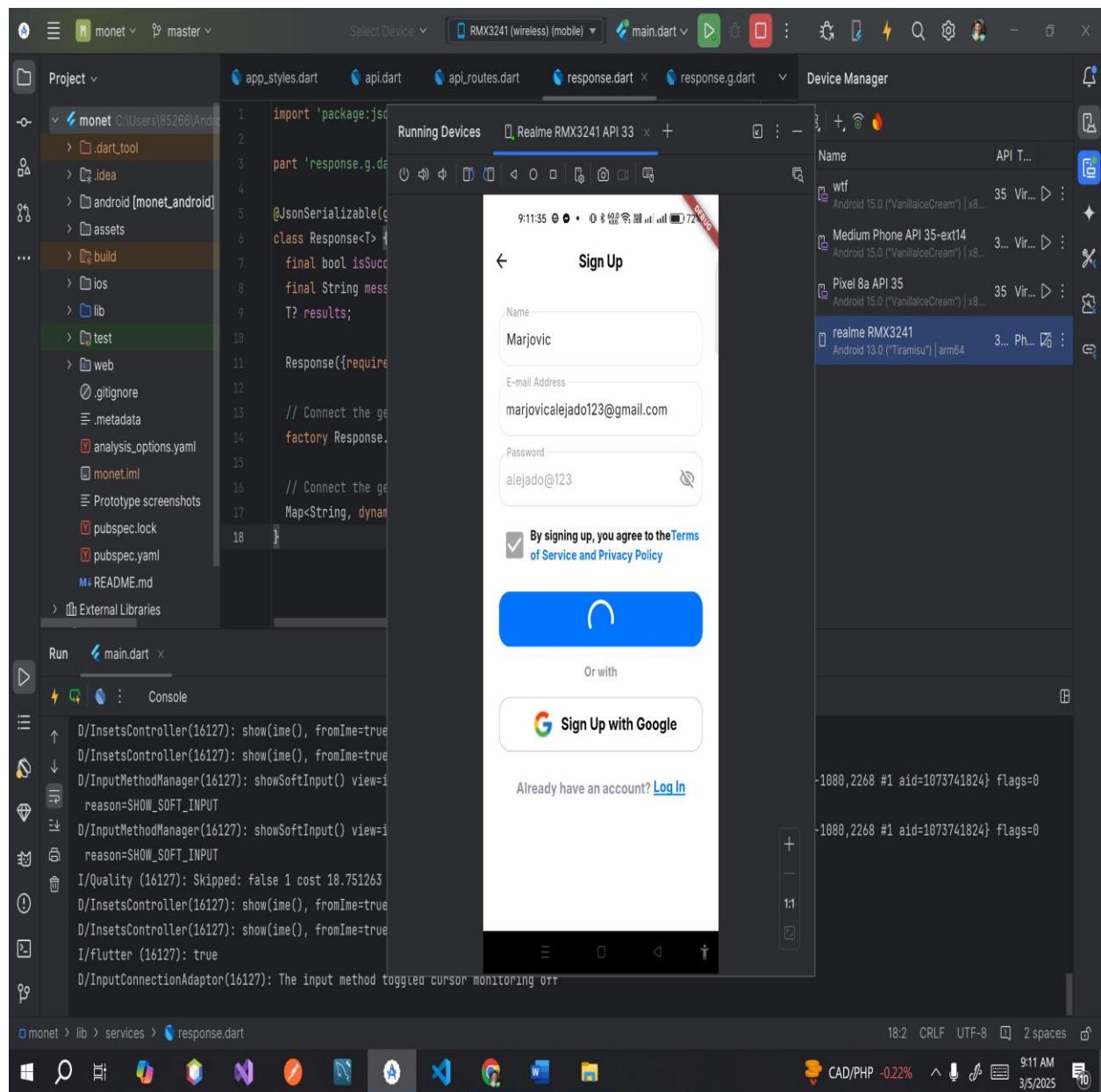


## Sign up / Log In System Implementation

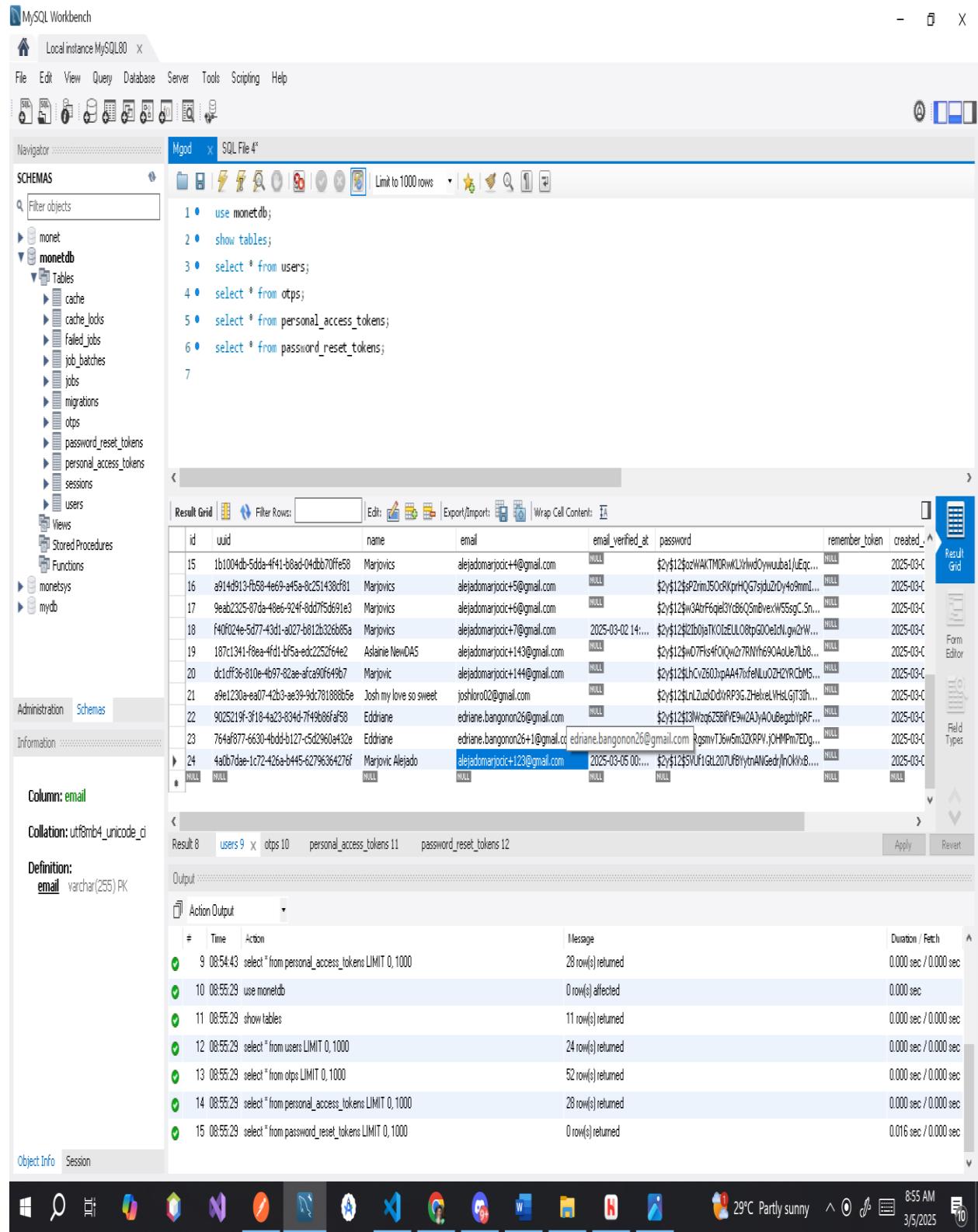




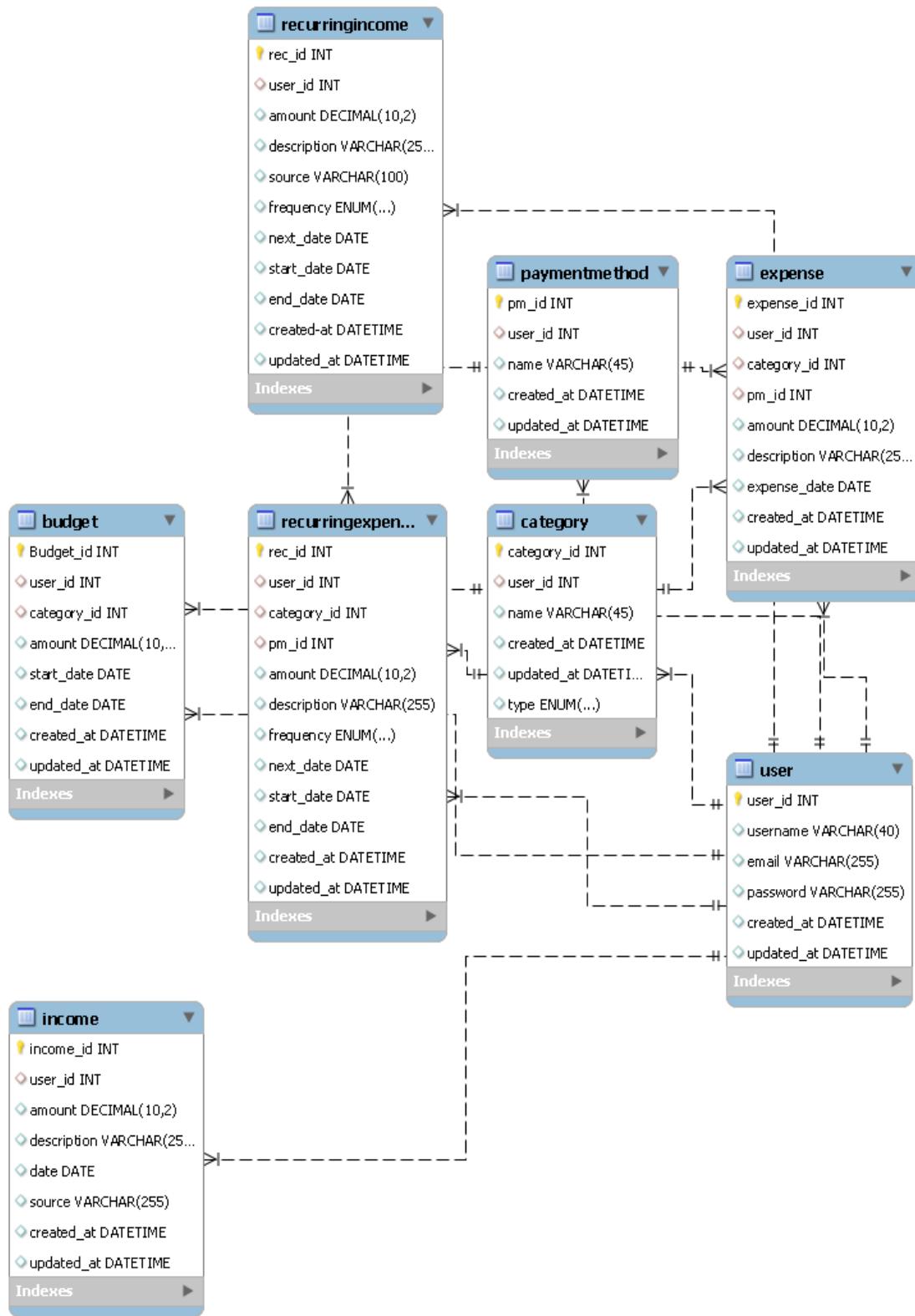


**Dashboard Implementation –** ps. we were working for it

## MySQL Database

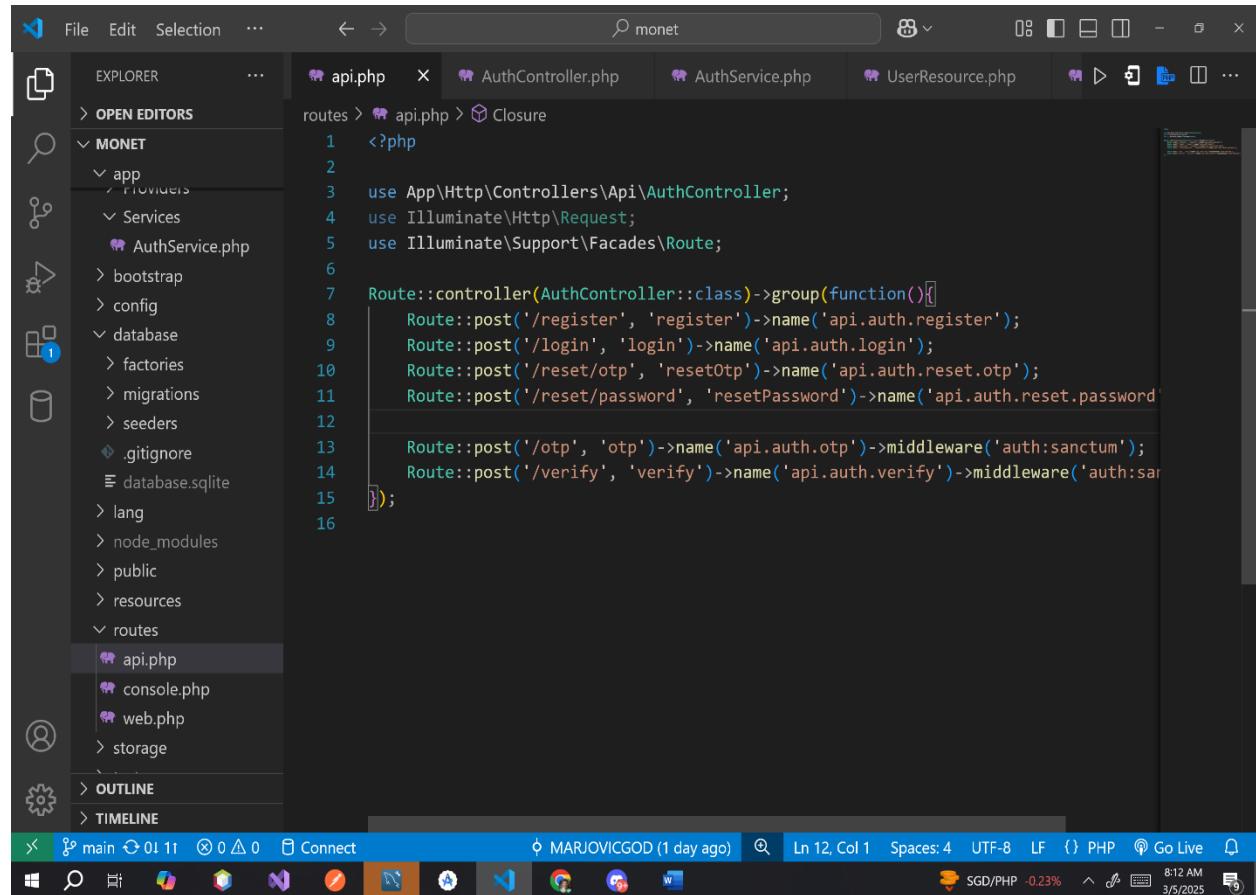


## Monet Schema Diagram



## Code Snippets

### routes/api



The screenshot shows the Monet IDE interface with the following details:

- File Bar:** File, Edit, Selection, ...
- Search Bar:** monet
- Editor Area:** The file "api.php" is open, showing PHP code for API routes. The code includes imports for App\Http\Controllers\Api\AuthController, Illuminate\Http\Request, and Illuminate\Support\Facades\Route. It defines a group for the AuthController and various POST routes for registration, login, password reset, OTP, and verification.
- Explorer Bar:** Shows the project structure under "MONET". The "routes" folder contains "api.php", "console.php", and "web.php". Other folders like "app", "Services", "config", "database", "migrations", "seeders", ".gitignore", and "database.sqlite" are also listed.
- Bottom Bar:** Includes tabs for main, Connect, and Go Live, along with system status indicators like battery level, time (8:12 AM), and date (3/5/2025).

```
<?php
use App\Http\Controllers\Api\AuthController;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;

Route::controller(AuthController::class)->group(function(){
    Route::post('/register', 'register')->name('api.auth.register');
    Route::post('/login', 'login')->name('api.auth.login');
    Route::post('/reset/otp', 'resetOtp')->name('api.auth.reset.otp');
    Route::post('/reset/password', 'resetPassword')->name('api.auth.reset.password');

    Route::post('/otp', 'otp')->name('api.auth.otp')->middleware('auth:sanctum');
    Route::post('/verify', 'verify')->name('api.auth.verify')->middleware('auth:sanctum');
});

});
```

## Auth\_Controller/Auth\_Service

### Register



```
1 <?php
2 namespace App\Http\Controllers\Api;
3
4 use App\Http\Controllers\Controller;
5 use App\Http\Resources\UserResource;
6 use App\Models\User;
7 use App\Services\AuthService;
8 use Illuminate\Support\Facades\Auth;
9 use Illuminate\Http\Request;
10 use Illuminate\Http\Response;
11
12 class AuthController extends Controller
13 {
14     protected AuthService $authService;
15
16     public function __construct(AuthService $authService) {
17         $this->authService = $authService;
18     }
19
20     public function register(Request $request) : Response [
21         // Validate user
22         $request->validate([
23             'name' => 'required|string|max:255',
24             'email' => 'required|email|unique:users,email|max:255',
25             'password' => 'required|min:6|max:255',
26         ]);
27         // Create user
28         $user = $this->authService->register($request);
29         // Create Access token
30         $token = $user->createToken('auth')->plainTextToken;
31         // Return
32         return response([
33             'message' => __('app.registration_success'),
34             'result' => [
35                 'user' => new UserResource($user),
36                 'token' => $token
37             ]
38         ], 201);
39     }
40 }
```

```
{} package.json      🐘 api.php      🐘 AuthController.php      🐘 AuthService.php X      🐘 User.php
app > Services > 🐘 AuthService.php > 🐘 AuthService > 🕒 register
1  <?php
2
3  namespace App\Services;
4
5  use App\Models\Otp;
6  use App\Models\User;
7  use Illuminate\Support\Facades\Hash;
8  use Illuminate\Support\Str;
9  use Nette\Utils\Random;
10 use Illuminate\Support\Facades\Mail;
11 use App\Mail\OtpMail;
12 use Carbon\Carbon;
13
14 class AuthService {
15     public function register(object $request) : User []
16     {
17         // Create user
18         $user = User::create([
19             'uuid' => Str::uuid(),
20             'name' => $request->name,
21             'email' => $request->email,
22             'password' => $request->password,
23         ]);
24
25         $this->otp($user);
26
27         return $user;
28     }
29
30     public function login(object $request) : ?User
31     {
32
33         // Create user
34         $user = User::where('email', $request->email)->first();
35         if($user && Hash::check($request->password, $user->password)) {
36             return $user;
37         }
38         return null;
39     }
40 }
```

## Log in

```
41     public function login(Request $request) : Response {
42         $request->validate([
43             'email' => 'required|email|max:255',
44             'password' => 'required|min:6|max:255',
45         ]);
46         // Login user
47         $user = $this->authService->login($request);
48         if(!$user) {
49             return response([
50                 'message' => __('auth.failed'),
51             ], 401);
52         }
53         // Create Access token
54         $token = $user->createToken('auth')->plainTextToken;
55         // Return
56         return response([
57             'message' => $user->email_verified_at ? __('app.login_success') : __('app.login_success_verify'),
58             'result' => [
59                 'user' => new UserResource($user),
60                 'token' => $token
61             ]
62         ], 200);
63     }
64 }
```

```
30     public function login(object $request) : ?User
31     {
32
33         // Create user
34         $user = User::where('email', $request->email)->first();
35         if($user && Hash::check($request->password, $user->password)) {
36             return $user;
37         }
38         return null;
39     }
40 }
```

## Otp

```
65  public function otp(Request $request) : Response {
66      /** @var User $user */
67      $user = Auth::user();
68
69      /** @var Otp $otp */
70      $otp = $this->authService->otp($user);
71
72  return response([
73      'message' => __('app.otp_sent'),
74  ]);
75 }
76
```

```
40
41  public function otp(User $user, string $type = 'verification') : Otp {
42
43      $tries = 3;
44      $time = Carbon::now()->subMinutes(30);
45
46      $count = Otp::where([
47          'user_id' => $user->id,
48          'type' => $type,
49          'active' => 1
50      )->where('created_at', '>=', $time)->count();
51
52      if($count >= $tries) {
53          abort(422, __('app.to_many_requests'));
54      }
55
56      $code = random_int(100000, 999999);
57
58      $otp = Otp::create([
59          'user_id' => $user->id,
60          'type' => $type,
61          'code' => $code,
62          'active' => 1
63      ]);
64
```

## Verify

```
77     public function verify(Request $request) : Response {
78
79         // Validate the request
80         $request->validate([
81             'otp' => 'required|numeric',
82         ]);
83
84         /** @var User $user */
85         $user = Auth::user();
86
87         // Verify the OTP
88         /** @var Otp $otp */
89         $user = $this->authService->verify($user, $request);
90
91         return response([
92             'message' => __('app.verification_success'),
93             'result' => [
94                 'user' => new UserResource($user)
95             ]
96         ]);
97     }
98 }
```

```
71     public function verify(User $user, object $request) : User {
72
73         $otp = Otp::where([
74             'user_id' => $user->id,
75             'code' => $request->otp,
76             'active' => 1
77         ])->first();
78
79         if (!$otp) {
80             abort(422, __('app.invalid_otp'));
81         }
82
83         // Update OTP
84         $user->email_verified_at = Carbon::now();
85         $user->update();
86
87         $otp->active = 0;
88         $otp->updated_at = Carbon::now();
89         $otp->update();
90
91         return $user;
92     }
93 }
```

## Reset using OTP with Password

```
99     public function resetOtp(Request $request) : Response {  
100  
101         // Validate request  
102         $request->validate([  
103             'email' => 'required|email|max:255|exists:users,email',  
104         ]);  
105  
106         // Get user  
107         $user = $this->authService->getUserByEmail($request->email);  
108  
109         /** @var Otp $otp */  
110         $otp = $this->authService->otp($user, 'password-reset');  
111  
112         return response([  
113             'message' => ___('app.otp_sent'),  
114         ]);  
115     }  
116 
```

```
117     public function resetPassword(Request $request) : Response {  
118  
119         // Validate request  
120         $request->validate([  
121             'email' => 'required|email|max:255|exists:users,email',  
122             'otp' => 'required|numeric',  
123             'password' => 'required|min:6|max:255|confirmed',  
124             'password_confirmation' => 'required|min:6|max:255',  
125         ]);  
126  
127         // Get user  
128         $user = $this->authService->getUserByEmail($request->email);  
129  
130         $user = $this->authService->resetPassword($user, $request);  
131  
132         return response([  
133             'message' => ___('app.password_reset_success'),  
134         ]);  
135     }  
136 
```

```
93     public function getUserByEmail(string $email): User
94     {
95         return User::where('email', $email)->first();
96     }
97
98     public function resetPassword(User $user, object $request): User
99     {
100         $otp = Otp::where([
101             'user_id' => $user->id,
102             'code' => $request->otp,
103             'active' => 1,
104             'type' => 'password-reset'
105         ])->first();
106
107         if (!$otp) {
108             abort(422, __('app.invalid_otp'));
109         }
110
111         // Update
112         $user->password = $request->password;
113         $user->updated_at = Carbon::now();
114         $user->update();
115
116         $otp->active = 0;
117         $otp->updated_at = Carbon::now();
118         $otp->update();
119
120         return $user;
121     }
122 }
```

# API

## Register

The screenshot shows the Postman application interface. The left sidebar displays 'My Workspace' with collections like 'Monet API' and 'Authentication'. Under 'Authentication', there are several requests: 'POST Register' (selected), 'POST Login', 'POST Otp', 'POST Reset OTP', 'POST Reset Password with Otp', and 'POST Verify Account'. The main workspace shows the 'POST Register' request details. The method is 'POST' and the URL is 'https://monet.test/api/register'. The 'Body' tab is selected, showing 'form-data' with three fields: 'name' (Marjovic Alejado), 'email' (alejadomarjocic+123@gmail.com), and 'password' (alejado@123). Below the body, the response status is '201 Created' with a timestamp of '20.16 s' and a size of '604 B'. The response body is displayed as JSON:

```
1 {
2   "message": "Account Created Successfully!",
3   "result": {
4     "user": {
5       "id": "4a0b7dae-1c72-426a-b445-62796364276f",
6       "name": "Marjovic Alejado",
7       "email": "alejadomarjocic+123@gmail.com",
8       "email_verified_at": null,
9       "created_at": "2025-03-05T00:32:38.000000Z",
10      "updated_at": "2025-03-05T00:32:38.000000Z"
11    },
12    "token": "27|81rGpl1zbh0vWBG02SM13fBXRNJuIckAQFHyVwed7b35bd2"
13  }
14 }
```

On the right side, there is a 'Code snippet' panel showing Dart code for sending the same POST request. The code uses http.MultipartRequest to build the request with headers and fields, then sends it and prints the response.

```
1 var headers = {
2   'Accept': 'application/json'
3 };
4 var request = http.MultipartRequest('POST', Uri.parse
5   ('https://monet.test/api/register'));
6 request.fields.addAll({
7   'name': 'Marjovic Alejado',
8   'email': 'alejadomarjocic+123@gmail.com',
9   'password': 'alejado@123'
10 });
11 request.headers.addAll(headers);
12
13 http.StreamedResponse response = await request.send
14   ();
15
16 if (response.statusCode == 200) {
17   print(await response.stream.bytesToString());
18 }
19 else {
20   print(response.reasonPhrase);
21 }
```

## Log In

The screenshot shows the Postman application interface. The left sidebar is titled "My Workspace" and contains sections for Collections, Environments, Flows, APIs, and History. The main workspace shows a collection named "Monet API" with a sub-collection "Authentication". A specific POST request for "Login" is selected. The request details show a POST method to "https://monet.test/api/login". The "Body" tab is active, displaying form-data fields: "email" (alejadomarjocic+123@gmail.com) and "password" (alejado@123). The response section shows a 200 OK status with a response time of 468 ms and a body size of 587 B. The response content is a JSON object:

```
1 {
2     "message": "Login Sucessfully, Please verify your account with
3         OTP!",
4     "result": {
5         "user": {
6             "id": "4a0b7dae-1c72-426a-b445-62796364276f",
7             "name": "Marjovic Alejado",
8             "email": "alejadomarjocic+123@gmail.com",
9             "email_verified_at": null,
10            "created_at": "2025-03-05T00:32:38.000000Z",
11            "updated_at": "2025-03-05T00:32:38.000000Z"
12        },
13        "token": "28|1V8unIVk0j1341AwVAvzMWJyph7bI25c77BbDPQ028e95e82"
14    }
}
```

The right panel displays a "Code snippet" for Dart - http, showing the equivalent code for making the API call:

```
1 var headers = [
2   'Accept': 'application/json'
3 ];
4 var request = http.MultipartRequest
5   ('POST', Uri.parse('https://monet.test/
6   api/Login'));
7 request.fields.addAll({
8   'email': 'alejadomarjocic+123@gmail.com',
9   'password': 'alejado@123'
10 });
11 request.headers.addAll(headers);
12 http.StreamedResponse response = await
13   request.send();
14 if (response.statusCode == 200) {
15   print(await response.stream.bytesToString()
16   ());
17 } else {
18   print(response.reasonPhrase);
19 }
20
```

The bottom navigation bar includes icons for Online, Find and replace, Console, Postbot, Runner, Start Proxy, Cookies, Vault, Trash, and a weather widget showing 29°C Partly sunny at 8:38 AM on 3/5/2025.

## Request OTP

The screenshot shows the Postman application interface. On the left, the sidebar displays 'My Workspace' with collections like 'Monet API', 'Authentication', and various API endpoints such as 'POST Register', 'POST Login', 'POST Otp', etc. The main area shows a 'POST Otp' request with the URL <https://monet.test/api/otp>. The 'Auth' tab is selected, showing 'Bearer Token' as the type. A note warns about sensitive data and recommends using variables. The 'Body' tab shows a JSON response with the message "OTP Sent Successfully!". To the right, a 'Code snippet' panel shows Dart code for making the same request using http.MultipartRequest. The bottom navigation bar includes icons for Postbot, Runner, Start Proxy, Cookies, Vault, Trash, and a search bar.

```
1 var headers = [
2   'Accept': 'application/json',
3   'Authorization': '.....',
4 ];
5 var request = http.MultipartRequest(
6   'POST',
7   Uri.parse('https://monet.test/api/otp')
8 );
9 http.StreamedResponse response = await
10 request.send();
11 if (response.statusCode == 200) {
12   print(await response.stream.bytesToString());
13 }
14 else {
15   print(response.reasonPhrase);
16 }
17
```

8:40:46 0.00 KB/S 76%



MONET 8:39 AM  
to alejadomarjoc... ▾



MONET

Hi Marjovic Alejado,

Your 6-digit code is:

**529322**

Use this code to complete the verification process in the app.

Do not share this code. MONET representatives will never reach out to you to verify this code over the phone or your email app.

**The code is valid for 10 minutes.**



Reply all



## Verify email using OTP

The screenshot shows the Postman application interface. The left sidebar displays the 'My Workspace' section with various collections, environments, flows, APIs, and history. The 'Monet API' collection is expanded, showing several POST requests: 'Register', 'Login', 'Otp', 'Reset OTP', 'Reset Pass', 'Register', 'Otp', and 'Verify Account'. The 'Verify Account' request is currently selected.

The main workspace shows the 'HTTP Monet API / Authentication / Verify Account' endpoint. The method is set to 'POST' and the URL is 'https://monet.test/api/verify'. The 'Body' tab is active, showing a single form-data entry: 'otp' with the value '529322'. The 'Headers' tab lists 'Content-Type: application/json' and 'Accept: application/json'. The 'Auth' tab shows a token '.....'.

On the right side, there is a 'Code snippet' panel for Dart - http. It contains the following code:

```
1 var headers = {  
2   'Accept': 'application/json',  
3   'Authorization': '.....'  
4 };  
5 var request = http.MultipartRequest  
  ('POST', Uri.parse('https://monet.test/  
    api/verify'));  
6 request.fields.addAll({  
7   'otp': '529322'  
8 });  
9  
10 request.headers.addAll(headers);  
11  
12 http.StreamedResponse response = await  
  request.send();  
13  
14 if (response.statusCode == 200) {  
15   print(await response.stream.bytesToString  
    ());  
16 }  
17 else {  
18   print(response.reasonPhrase);  
19 }
```

The bottom section shows the response details. The status is '200 OK' with a response time of '228 ms' and a size of '518 B'. The 'Body' tab displays the JSON response:

```
1 {  
2   "message": "Account Verified Successfully!",  
3   "result": {  
4     "user": {  
5       "id": "4a0b7dae-1c72-426a-b445-62796364276f",  
6       "name": "Marjovic Alejado",  
7       "email": "alejadomarjocic123@gmail.com",  
8       "email_verified_at": "2025-03-05T00:44:15.000000Z",  
9       "created_at": "2025-03-05T00:32:38.000000Z",  
10      "updated_at": "2025-03-05T00:44:15.000000Z"  
11    }  
12  }  
13 }
```

The bottom navigation bar includes icons for Online, Find and replace, Console, Postbot, Runner, Start Proxy, Cookies, Vault, Trash, and a search bar. The system tray shows the date and time as '3/5/2025 8:44 AM'.

## Request OTP for change password

The screenshot shows the Postman application interface. On the left, the sidebar lists collections, environments, flows, and history. The main area displays a POST request for 'Reset OTP' under the 'Monet API / Authentication' collection. The request URL is `https://monet.test/api/reset/otp`. The 'Body' tab shows a form-data key 'email' with the value 'alejadomarjocic+123@gmail.com'. The response section shows a 200 OK status with a JSON body containing the message 'OTP Sent Successfully!'. To the right, a code snippet panel shows Dart code for making the same API call using http.MultipartRequest.

```
1 var headers = [
2   'Accept': 'application/json',
3   'Authorization': '*****'
4 ];
5 var request = http.MultipartRequest(
6   ('POST', Uri.parse('https://monet.test/
7     api/reset/otp'));
8 request.fields.addAll({
9   'email': 'alejadomarjocic+123@gmail.com'
10 });
11
12 http.StreamedResponse response = await
13   request.send();
14
15 if (response.statusCode == 200) {
16   print(await response.stream.bytesToString()
17 ());
18 } else {
19   print(response.reasonPhrase);
20 }
```

8:48:09 🔍 🎙 • ⚡ 0.05 KB/S WiFi VOD LTEB 75%



MONET 8:46 AM  
to alejadomarjoc... ▾



MONET

Hi Marjovic Alejado,

Your 6-digit code is:

**502247**

Use this code to reset your password in the app.

Do not share this code. MONET representatives will never reach out to you to verify this code over the phone or your email app.

**The code is valid for 10 minutes.**

Thanks



Reply all



## Reset Password with OTP

The screenshot shows the Postman application interface for testing an API endpoint. The top navigation bar includes 'Home', 'Workspaces', 'API Network', a search bar, and various status indicators like 'Invite', 'Upgrade', and environment dropdowns.

The left sidebar contains sections for 'Collections' (with 'Monet API / Authentication / Reset Password with OTP'), 'Environments' (with 'https://monet.test/api/reset/password'), 'Flows', 'APIs', and 'History'.

The main workspace displays a POST request for 'Reset Password with OTP'. The URL is set to `https://monet.test/api/reset/password`. The 'Body' tab is selected, showing a 'form-data' payload with four fields:

Key	Value	Description
email	alejadomarjocic+123@gmail.com	
otp	502247	
password	alejado@123	
password_confirmation	alejado@123	

The 'Code snippet' panel on the right shows Dart code for making the same API call using the http package:

```
1 var headers = {
2   'Accept': 'application/json',
3   'Authorization': '.....'
4 };
5 var request = http.MultipartRequest('POST', Uri.parse('https://
monet.test/api/reset/password'));
6 request.fields.addAll({
7   'email': 'alejadomarjocic+123@gmail.com',
8   'otp': '502247',
9   'password': 'alejado@123',
10  'password_confirmation': 'alejado@123'
11 });
12
13 request.headers.addAll(headers);
14
15 http.StreamedResponse response = await request.send();
16
17 if (response.statusCode == 200) {
18   print(await response.stream.bytesToString());
19 }
20 else {
21   print(response.reasonPhrase);
22 }
23
```

The 'Body' section at the bottom shows the response details: a 200 OK status with a response time of 437 ms, a body size of 361 B, and a 'Save Response' button. The response content is displayed as JSON:

```
[{"message": "Password Reset Successfully!"}]
```

The bottom navigation bar includes icons for 'Online', 'Find and replace', 'Console', 'Postbot', 'Runner', 'Start Proxy', 'Cookies', 'Vault', 'Trash', and system status indicators for weather (29°C Partly sunny), time (8:52 AM), and date (3/5/2025).

## **Challenges and Solutions Applied**

Developing a **Money Expense Tracker System** using **Laravel** for the backend, **Flutter** for the UI, and **GitHub** for version control can be a rewarding but challenging project.

Below are some common challenges you might face and solutions to fix them:

### **1. Backend Development Challenges (Laravel)**

#### **Challenges:**

- **Complex Database Relationships:** Managing connections between tables (like users, expenses, and categories) can be hard.
- **API Development:** Making RESTful APIs that are safe, can grow, and are easy to understand.
- **Validation and Error Handling:** Checking user inputs properly and showing errors in a nice way.
- **Authentication and Authorization:** Setting up secure logins and controlling what users can do.
- **Performance Optimization:** Handling lots of data and making database searches fast.

#### **Solutions:**

- Use Laravel's Eloquent ORM to set up and manage relationships (likehasMany and belongsTo).
- Use Laravel's API resources and controllers to organize your APIs. Test them with tools like Postman.
- Use Laravel's built-in rules to check inputs and make custom error messages.
- Use Laravel Passport or Sanctum for secure logins and JWT tokens.
- Make things faster with Laravel's query builder, caching (like Redis), and database indexing.

### **2. Frontend Development Challenges (Flutter)**

## **Challenges:**

- **State Management:** Keeping track of the app's data (like expenses and user settings) efficiently.
- **UI/UX Design:** Building an interface that's easy to use and works on all devices.
- **API Integration:** Connecting the Flutter app to the Laravel backend and handling responses.
- **Cross-Platform Compatibility:** Making sure the app works well on Android and iOS.
- **Data Synchronization:** Storing data offline and updating the backend when online.

## **Solutions:**

- Use tools like Provider, Riverpod, or Bloc to manage the app's data.
- Use Flutter's widget library and tools like FlutterFlow to design the interface. Follow Material Design rules.
- Use the http or dio package to call APIs and manage responses. Add loading and error screens.
- Test the app on Android and iOS devices or emulators. Add special code if needed.
- Use Hive or SQLite to save data offline and sync it with the backend when online.

## **3. Version Control Challenges (GitHub)**

### **Challenges:**

- **Branch Management:** Handling different code versions for features, fixes, and releases.
- **Merge Conflicts:** Fixing problems when combining code changes.
- **Collaboration:** Working with a team and following good practices.
- **CI/CD Integration:** Setting up automatic testing and deployment.

### **Solutions:**

- Use GitFlow or a similar plan. Name branches clearly (like feature/add-expense or bugfix/login-issue).
- Pull updates often and fix conflicts early. Use GitHub's tools to help.
- Use GitHub Issues and Pull Requests to track work and review code. Set coding rules for everyone.
- Set up GitHub Actions to test and deploy code automatically.

## 4. Integration Challenges

### Challenges:

- **Backend-Frontend Communication:** Making sure the backend and frontend work together smoothly.
- **Data Consistency:** Keeping data the same in the app and on the server.
- **Security:** Protecting private info (like passwords and expenses) during transfer and storage.

### Solutions:

- Use JSON for API responses and keep the data structure the same on both sides.
- Handle errors and check data on both ends. Use timestamps or versions for updates.
- Use HTTPS for APIs, encrypt sensitive data, and add secure login methods.

## 5. Testing and Debugging

### Challenges:

- **Backend Testing:** Making sure the Laravel backend has no bugs and works with many users.
- **Frontend Testing:** Testing the Flutter app for ease of use, speed, and device support.
- **End-to-End Testing:** Checking the whole system to ensure everything works together.

## **Solutions:**

- Use PHPUnit to test Laravel's units and features. Test APIs with Postman.
- Use Flutter's testing tools for unit, widget, and integration tests.
- Do manual and automatic tests to check the full system.

## **6. Deployment Challenges**

### **Challenges:**

- **Backend Deployment:** Putting the Laravel app on a server (like AWS, DigitalOcean, or Heroku).
- **Frontend Deployment:** Publishing the Flutter app to Google Play Store and Apple App Store.
- **Environment Configuration:** Managing different setups (like development, staging, and production).

### **Solutions:**

- Use tools like Laravel Forge or Envoyer to deploy easily. Set up .env files for each environment.
- Follow Flutter's steps to build and release the app on Android and iOS.
- Use environment variables and separate config files for each setup.

## **7. Documentation and Maintenance**

### **Challenges:**

- **Code Documentation:** Writing clear code with good explanations.
- **User Documentation:** Making guides and tutorials for app users.
- **Maintenance:** Updating the app to fix bugs and add features.

### **Solutions:**

- Use PHPDoc for Laravel and DartDoc for Flutter. Add helpful comments and README files.
- Create a user manual or in-app tutorials with Flutter.

- Plan regular updates and track issues with GitHub Issues.