

Case Study: Frontend and Backend Development for a Blog Platform

Andes, Warren [1], Lao, Vanness [2], Paluga, Edelyn Joy [3],
Tamayo, Lance [4], Tinamisan, Alexis [5]

1. INTRODUCTION

Web development is the process of developing web applications, including blogs, e-commerce sites, websites, and many more (Webb & Medleva, 2024). Both private and public access to these web apps is possible via the intranet and the Internet. A website's frontend and backend portions are the two primary parts of web development.

Frontend (Client-Side)

- Everything that users directly interact with in their browsers, such as the user interface, layout, and design, is referred to as frontend (client-side) work. Some of the tools utilized in front-end website development are HTML, CSS, JavaScript, React, and so on.

Backend (Server-Side)

- The Backend (Server-Side) includes the logic, database interactions, and codes that are hidden from the users. The backend makes the website's overall performance by having different functions like requesting and sending data to the users. Some tools that are used in the backend of a website include Laravel, Django, Express.js, etc.

However, during the process of web development, some issues and difficulties can arise due to various reasons. These

challenges that web developers may encounter include:

1. Data Inconsistencies
2. API Communication Issues
3. Authentication and Authorization
4. Performance Bottlenecks
5. Error Handling and Debugging

2. OBJECTIVES

This case study aims to overcome the challenges that both the frontend and backend team encounter with their project which is developing a dynamic blog platform using HTML, CSS, JavaScript for the frontend, and PHP Laravel for the backend.

Frontend Team:

1. Identify three specific issues that developers might face with the API responses.
2. Propose two debugging strategies to resolve these issues.
3. Describe how the frontend team communicated these issues to the Backend Team.

Backend Team:

1. Outline three best practices that should be followed when designing the API endpoints in Laravel.
2. Explain the implementation of error handling in the API to assist the Frontend Team in troubleshooting.

3. Describe one testing methodology to be used to ensure that the API is functioning correctly before the Frontend Team integrates it.

3. METHODOLOGY

3.1 Frontend

3 Specific Issues Developers Might Face with the API Responses

[1] Improper API Documentation

In many projects, API documentation is not always organized or well-written, resulting in miscommunications and difficulties in figuring out how everything works (Bilavendran, 2024). API documentation needs to be rigorously and well-written because it tells the developers where to connect endpoints, the kind of information to be sent or the data format, as well as the proper usage of it such as adding passwords or special keys.

[2] Unstable and Underdeveloped APIs

According to Bilavendran (2024), some APIs can not function properly, especially when developers are just starting to develop a project. This happens because there are circumstances in which APIs depend on other systems or services to function. If those systems are not available when needed, API might not function properly.

[3] Exceptional Handling Problems

Developers might not set up everything smoothly when dealing with different status codes. This circumstance

may lead to imperfections or malfunctions causing some parts of the program to not work as expected (Bilavendran, 2024).

2 Debugging Strategies to Resolve the Issues

[1] Effective API Documentation

API Documentation acts as a user manual since it contains detailed information regarding API. Having a well-written API Documentation is important so that developers can easily access clear instructions and examples to help them effectively use the API. This also promotes consistency which makes maintaining, debugging, and updating applications much easier (Anwar, 2024).

[2] API Testing and Proper Exception Handling

API Testing focuses on the verification of quality and performance of endpoints to provide a seamless digital experience (Postman, 2024). With this, developers can monitor, and detect issues early and then fix them before reaching production. Proper Exception handling is important since it ensures the program's continuance by handling errors. It also makes the program robust and resistant to unexpected conditions. Using specific exception types and meaningful error messages are some practices that can be utilized to have proper exception handling (Tutorialspoint, 2024). By implementing API Testing and Proper Exception Handling, developers can ensure that the system is well-equipped to handle errors.

Communicating with the Backend Team

First, arrange a meeting with the whole backend team. During the meeting, we will talk about the current project status and any challenges the team may be facing. Open communication and collaboration will be encouraged to effectively address any issues. The first part would be the discussion of issues such as improper API documentation, unstable and underdeveloped APIs, and exceptional handling problems. We would explain these by providing the description and impact of the said issues in a detailed and calm manner, ensuring that they fully comprehend them. Next, we will offer our proposed solutions (effective API documentation, API testing, and proper exception handling) to address these problems. Afterward, we would give them ample time to propose their suggestions and ideas before finalizing the action plan. Doing this will ensure that everyone is heard and fully understands the steps that need to be taken to improve the backend system.

3.2 Backend

3 Best Practices for API Design in Laravel

[1] Versioning

Versioning is key to maintaining and evolving APIs without breaking functionality. In Laravel, two common strategies are:

URL Versioning:

- This approach includes the version number in the API URL (e.g., /v1). For example:



```
1 Route::middleware('auth:sanctum')->prefix('v1')->group(function(){});
```

It allows distinct API versions with clear separation and easy management, though it can lead to longer URLs and more routing configuration.

Folder Versioning:

- Organize files into versioned folders like Controllers/V1. This keeps versions distinct and ensures updates don't affect earlier versions, maintaining backward compatibility.

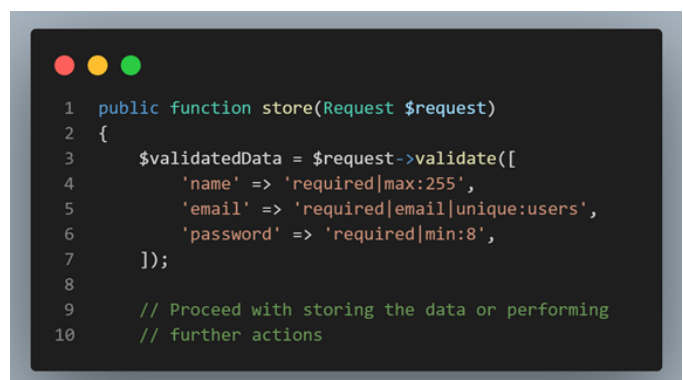
[2] Data Validation

Laravel offers two main methods for validating incoming data:

Validating in the Controller:

- Validation rules are applied directly in the controller method handling the request, using `$request->validate()`. This is simple and effective for quick validation.

Example:



```
1 public function store(Request $request)
2 {
3     $validatedData = $request->validate([
4         'name' => 'required|max:255',
5         'email' => 'required|email|unique:users',
6         'password' => 'required|min:8',
7     ]);
8
9     // Proceed with storing the data or performing
10    // further actions
```

Using Form Request Files:

- This approach organizes validation into custom classes that extend `FormRequest`, keeping validation logic separate from the controller. It is ideal for more complex or reusable validations.

Example:

```
1 class StoreUserRequest extends FormRequest {
2     public function rules() {
3         return [
4             'name' => 'required|max:255',
5             'email' => 'required|email|unique:users',
6             'password' => 'required|min:8',
7         ];
8     }
9 }
```

In both methods, if validation fails, Laravel handles error redirects automatically. Form requests are preferred for complex or reusable validation, while controller validation is quicker for simpler cases.

[3] Documentation

Provide thorough documentation for each API endpoint, including the expected request format, parameters, and possible responses.

Error Handling

To help the Frontend Team troubleshoot, it is important to prepare different error-handling methods in the Laravel API. We can use these different strategies and code examples to implement error handling.

[1] Standardized Error Responses

When the API encounters an error, it should respond consistently, which means that every error response should have a message or indication and the right HTTP status code.

Example:

```
{
  "error": {
    "message": "Resource not found",
    "status_code": 404
  }
}
```

[2] Custom Exception Handling

In Laravel, you can customize how your API handles specific errors by modifying the `'render'` method in `'App\Exceptions\Handler.php'`. When something is not found, we can return a specific message.

Example:

```
{
  "error": {
    "message": "This Route is not found",
    "status_code": 404
  }
}
```

[3] HTTP Status Codes

Use the correct HTTP status codes for different errors because these codes accurately help the Frontend Teams understand the context of the error.

Example:

- **404** for Not Found

- **401** for Unauthorized
- **422** for Validation Errors

Testing Methodology

When it comes to this testing, the main focus is on verifying the functionality and performance of the API. Basic assertions start from validating the response body, schema, and response codes.

However, other tests need to be performed, such as security testing and performance testing of APIs.

Here are some of the common types based on which test cases can be created:

- **Functional Testing:** Functional testing for APIs involves verifying that expected responses and data formats are returned given a specific request. This is done by sending requests to an API and validating that the correct responses are returned. This testing should be done to check that new and existing functionality behaves in the expected manner. For example, validating the status code and the response body.

Why API Functional Testing?

API functional testing is vital to validate if the API functions as expected and can handle different data types, requests, and responses. Using the test results, developers can easily identify and fix bugs, security vulnerabilities, or issues in the API.

4. CONCLUSION

Resolving challenges between the frontend and backend teams is crucial for the successful development of the dynamic

blog platform. In doing this, the frontend team encountered issues such as unclear API documentation, unstable responses, and handling errors, all of which created roadblocks in displaying content properly. To address these problems, improving API documentation, conducting thorough testing, and implementing proper error handling were proposed as solutions. By doing so, the frontend team can better understand how to interact with the API, reducing miscommunication and ensuring a smoother process for retrieving and displaying content.

On the other hand, the backend team plays an equally important role in resolving these issues. By following best practices like API versioning, data validation, and providing clear error messages, they can help the frontend team troubleshoot more effectively. Standardized error handling will also assist in better communication between the two teams, ensuring that when issues arise, they are dealt with consistently. Additionally, the backend team should conduct comprehensive testing of the API before integrating it with the frontend to catch and fix any issues early on.

Finally, collaboration between the frontend and backend teams is key to overcoming these challenges. By working together, improving communication, and implementing technical solutions, they can ensure the platform is both functional and user-friendly, resulting in a successful project. Just as Dalai Lama said, "The best way to resolve any problem in the human world is for all the sides to sit down and talk." With joint cooperation, any problem can be solved, whether it is technical or internal.

5. REFERENCES

Adeshina, A. (2023, July 24). Laravel API Error Exception Handling Methodologies. <https://dev.to/iamadeyemiadex/laravel-api-error-exception-handling-methodologies-26bb>

Anwar, Mariam. (2024, July 5). What is API Documentation and Why is it Important? Astera. <https://www.astera.com/type/blog/api-documentation/>

Bilavendran, P. (2024, October 1). Tackling Common API Testing Challenges. MuukTest. <https://muuktest.com/blog/challenges-in-api-testing>

Kwade, J. (2023, June 5). Building a RESTful API with Laravel: Best Practices and Implementation Tips. Retrieved from DEV Community website: <https://dev.to/rebelnii/building-a-restful-api-with-laravel-best-practices-and-implementation-tips-1lok>

Postman. (2024). API Testing. <https://www.postman.com/api-platform/api-testing/>

Testsigma Engineering Team (August 20, 2024) API Functional Testing – Why Is It And How to Test from <https://testsigma.com/blog/api-functional-testing/>

Tutorialspoint. (2024). Importance of Proper Exception Handling in Java. <https://www.tutorialspoint.com/importance-of-proper-exception-handling-in-java>

Webb, M., & Medleva, N. (2024, May 14). *What is Web Development? Definition, Processes & Types.* Techopedia. <https://www.techopedia.com/definition/23889/web-development>