ACTIVITY #1
CASE STUDY

GROUP 1 MEMBERS :

ALDAS, DOMINIC SYD V.
AMPUSTA, JANINE CHRYSTAL B.
CANDELARIO, MACKY D.
REY, KATRINA A.
MIRADOR, SHEENA ALAINE L.

CASE STUDY ABOUT FRONT AND BACK DEVELOPMENT

You are part of a project team developing a dynamic blog platform using HTML, CSS, JavaScript for the frontend, and PHP Laravel for the backend. The frontend team encounters issues retrieving blog posts from the Laravel API, resulting in unexpected responses and difficulties displaying the content.

FRONTEND TEAM

Identify three specific issues you might face with the API responses.

1. Issues in Configuration of Environment

Applying a great deal of the foundational concept of the framework needs to be configured properly with the help of a .env file that stores the environment settings for different stages of application development. The file also contains details considered confidential as well as the configurations that are prerogative of the design inside which the application is currently standing (local, staging, production and so on and so forth).

A typical example would be where these system variables are not set properly or even worse they are left intact and unchanged for the production environment. This can result in a series of problems such as failure to connect to the database or the application working in an undesired manner beyond its intended use scope or worse security issues.

2. Compatibility and Extensions of the Server

Server compatibility is often an issue specifically related to the versions of PHP and associated extensions. Every framework, including Laravel, has minimum requirements that the server should satisfy; otherwise, the application might not run.

3. Problems with Database Migration

One of the advantages of utilizing the Laravel framework is the ease of managing the database schema changes with migrations. Deploying new versions of an application using migrations can be problematic at times when the production database does not correspond to the assumed state or when the migrations are not designed to be idempotent.

FRONTEND TEAM

Propose two debugging strategies you would use to resolve these issues.

1. Important Factors in the Setup of the Environment

Strategy A: Examine and Confirm .env File Information

Action: Investigate the .env settings for environment variables that are either missing or wrongly configured. Make sure the required settings such as the database connection, app key and any other major settings are properly configured for the respective environments whether local, staging or production.

Tools: Also, clear and cache the configuration by using php artisan config:cache so that the application embraces the new setting as required.

Strategy B: Practice Configuration Checks According To The Environment

Action: Write or leverage existing Laravel commands to implement environment validation. Such a script can be used to accept and check 'critical' variables and values, raising a warning when it is missing or when it exists set to values that are considered unsafe.

Tools: Capture found within thresholds differences in the environment validation process using the logging functionality embedded in the Laravel platform.

2. Server Compatibility and Additions

Strategy A: Verify Server Specifications

Action: Perform an audit of the server PHP version and other plugins against the requirements of Laravel framework. This is done by the reference to the official Laravel documentation regarding the effective version of the framework being used.

Tools: Issue the command php -m to display the feasible programs, issues the command php -v to make known the php version in use. Alter the server configuration or add the extensions if they are unavailable.

FRONTEND TEAM

Strategy B: Prefer Working on a Local Machine Action:
Action: Create an isolated development space (for example by Docker, Homestead or Valet) where the structure of the production server will be copied and any extensions and interoperability will be tested prior to implementation.
Tools: Employ tools such as Laravel Sail or Docker Compose to reproduce the production environment on a local machine with ease.

3. Challenges Encountered during Data Migration

Strategy A: Examine Migration Artifacts and Its Timeline
Action: Examine the migration scripts to determine that they conform to the principle of idempotency as they pertain to the application process (i.e., can be executed several times without any problems arising). Inspect the migrations table of the database to view the present status of migrations and highlight conflicts, if any, in relation to other tables.
Tools: Use php artisan migrate:status to check all migrations and see which one has been executed and which one has not (if any).

Strategy B: Take Precautions in Performing Migrations in Staging Environment First.
Action: For production purposes, run the migrations only after a backup of the database has been performed and the migrations have been tried out in an environment which is as close to production as possible. This helps avoid potential problems from affecting operational data.
Tools: Enable database backup solutions such as mysqldump in MySQL together with Laravel's migration rollback capabilities, to preview changes and even backtrack the changes if necessary.

FRONTEND TEAM

Describe how you would communicate these issues to the Backend Team, including the information you would provide.

1. Misconfigurations of the Deployment Environment

Findings:
The team has reported the possible absence or incorrect definition of some variables, particularly in the .env file. This can cause severe problems, including but not limited to the inability to connect to a database and exposure to security risks, most especially in the production environment.

Recommended Actions:
Investigate and Verify Contents of .env File:
The team suggests that the contents and configurations in the .env file which affect the database connection, application keys and other critical factors for other environments (local, staging and production) should be closely evaluated.
Upon completion of the evaluation, the team recommends executing the command php artisan config:cache to ensure that the relevant updates have been made to the application.

Conduct Regular Verifications of Set-ups:
The team suggests that a simple script should be created, or already available Laravel commands should be used for environment variables checking on a regular basis. Missing or insecure values, in this case – parameters in the configuration file should trigger notifications to the team.
In the process of testing, the discrepancies encountered can also be logged using Laravel.

FRONTEND TEAM

Describe how you would communicate these issues to the Backend Team, including the information you would provide.

2. Server Compatibility and Extensions

Current Issues:
The team must take precautions regarding the server in regard to serving Laravel especially in the use of the correct PHP versions and also, the required extensions. Otherwise, some errors could be encountered when running the program.

Proposed Solutions:

Tasks to Do Orientation to the Server Specifications:
We advise the inspection of the existing PHP version and the current installed extensions. The team should check the official Laravel documentation in order to ensure the full accomplishment of the set targets.
The team can execute php -m to view the currently installed PHP modules, therefore the team also needs to run php -v to check which version of PHP is currently installed. Any versions that differ from one another should be resolved immediately.

Make Use Of Local Development Environments:
It is recommended to set up local environments using Docker or Laravel Sail that are the same with the production environment. This enables testing the compatibility and the extensions without touching the live application.
Sharing on the creation of docker for local development is encouraged.

FRONTEND TEAM

Describe how you would communicate these issues to the Backend Team, including the information you would provide.

3. Difficulties in Database Migrations

Understood Threats:
The team has flagged a few threats with respect to carrying out database migrations, particularly when they are not idempotent or the state of the production database is not as expected.

Suggested Methods:

Assess Migration Artifacts:
A thorough check of migration files is important to report their potential adoptions without any issues. The checking of the migrations table will aid in detecting any discrepancies.
The team can carry out php artisan migrate:status to find out which migrations have already been executed and which ones are still outstanding.

Test Migrations in Staging First:
Before running migrations in production, the team should take a database backup and verify everything in a staging environment. It would do well in nipping possible issues in the bud.

Also, there is a need to have a backup plan in place which could use mysqldump and also learn how to embed rolling back of the migrations in Laravel.

BACKEND TEAM

Outline three best practices you should follow when designing the API endpoints in Laravel.

1. Follow RESTful standards

It should also use the proper HTTP methods. The verbs used most are GET to fetch the data, POST to add, PUT/PATCH to modify, and DELETE to delete.

Use clear and simple URLs: Use your endpoints to indicate what their purpose is, such as all users, like `/api/users`, or a particular order, like `/api/orders/{id}`.

2. Secure API

Even make use of authentication: Use token-based authentication with tools like Laravel Passport or Sanctum for a safe API.

It includes access control, which ensures that only authorized users can view or modify certain data, especially where sensitive activities are involved.

3. Properly validate and handle errors

Validate your data: Only accept the incoming requests so that they can be processed.

Clear error messages: Use the proper status codes and send obviously readable error messages in a normal format which is generally JSON.

BACKEND TEAM

Explain how you would implement error handling in your API to assist the Frontend Team in troubleshooting.

To assist the Frontend Team in troubleshooting, we would implement error handling in our API with the following methods:

-> Structured, Consistent Error Responses:
Each error response would follow a uniform structure across the API, enabling easy parsing and making errors predictable for frontend handling.

-> Standardized Error Codes:
Utilizing standardized error codes helps categorize issues, allowing frontend developers to quickly recognize and troubleshoot errors. These codes simplify debugging and enhance compatibility across different systems.

-> Continuous Monitoring and Improvement:
Regularly reviewing error logs and user feedback will allow us to spot patterns and identify improvement areas. This iterative approach to error handling ensures the API's reliability is continuously refined.

-> Server-Side Error Logging:
Using Laravel's logging system (such as Log::error()), we capture essential details of server-side issues, like user data and timestamps, without exposing sensitive information to the frontend. This aids backend diagnostics while keeping user data secure.

-> Documenting Error Codes and Responses:
We maintain comprehensive documentation of all error codes, descriptions, and example responses. This resource enables frontend developers to troubleshoot effectively and test various error scenarios.

BACKEND TEAM

Describe one testing methodology you would use to ensure that your API is functioning correctly before the Frontend Team integrates it.

Manual API testing can be done to guarantee the functionality and performance of the API. Manual testing involves human judgment and insight in the testing process, making it more adaptable to changing requirements as it allows the testers to quickly adjust the test cases if necessary. This method requires testers to directly test the API by manually sending requests to the API and validating its responses using manual testing tools such as Postman and SoapUI. To do this, the tester must set up the tool first. Once set up, prepare the test plan by determining which conditions need to be tested and the criteria that must be met. Next, an environment must be created to test the API from the beginning up to end. It is also important for the tester to identify and record any errors, vulnerabilities, or behaviors that occurred during the testing process. Lastly, the testing progress must be tracked to ensure its coverage and status.

REFERENCES

Specific issues and debugging solutions:
What are the most common problems in Laravel and its solutions? - Quora
Resolving The Common Laravel Deployment Issues
Laravel Error Handling: Effective Strategies for Debugging

Communicating between Frontend and backend:
Efficiently communicating between frontend and backend sites without exposing backend - Stack Overflow

Building a RESTful API with Laravel: Best Practices and Implementation Tips:
Building a RESTful API with Laravel: Best Practices and Implementation Tips. - DEV Community

API Error Handling: Techniques and Best Practices:
API Error Handling: Techniques and Best Practices | by Rory Murphy

Best Practices for API Error Handling
Best Practices for API Error Handling | Postman Blog

Outline three best practices you should follow when designing the API endpoints in Laravel.
Authentication - Laravel 11.x - The PHP Framework For Web Artisans

API Manual Testing
Automated API Testing Vs Manual Testing.