Where is the Power?

Testing Policy

APP Testing Policy

Verification of Testing Policy

To ensure the effectiveness of our application testing policy, it is imperative that we rigorously verify whether the feature under development aligns seamlessly with the defined functional requirements. This verification process serves as the foundational step to ascertain that the feature is not only being worked upon but is also being implemented accurately in accordance with the specified functional criteria.

Diverse Testing Types

In our comprehensive testing strategy, usability testing takes precedence even before the initiation of the pull request submission. This early-stage usability evaluation is instrumental in identifying potential user experience issues and addressing them proactively. Additionally, it is essential to enforce a prerequisite of ensuring the successful completion of previous unit and integration tests before any code is deemed ready for review.

Post-Implementation Testing Collaboration

Following the implementation of a new feature, a pivotal phase involves reaching out to our dedicated testers. These testers play a critical role in crafting unit, integration, and end-to-end (E2E) tests that scrutinize the functionality and integration of the new feature within our application environment. Their involvement is vital in guaranteeing the robustness and reliability of the implemented feature.

Testing Environments and Deployment Protocols

Our testing environments are carefully structured to uphold the integrity of our application. Primary testing activities are primarily conducted on the developer's local machine, ensuring that the initial testing and debugging stages are carried out efficiently. Moreover, it is mandatory that all new features are integrated into the development branch before any release, fostering a systematic approach to feature integration. The subsequent deployment of a dedicated development site, emulating a production-like environment, is imperative. This dev site serves as a valuable testbed to verify the seamless functioning of usability testing. When it comes to deploying changes to the

production environment, it is incumbent upon us to validate that all code has undergone meticulous testing, with an emphasis on automation to ensure reliability and consistency.

Efficient Bug Reporting and Resolution

In our commitment to maintaining the quality and integrity of our application, we have established a robust system for handling bugs. Any instances of feature-related bugs or those reported by users through our email channels necessitate the creation of corresponding issue reports. These reports are meticulously tagged with 'Frontend' and 'bug' labels to streamline the tracking and resolution process. Importantly, it's crucial to acknowledge that every resolved bug must undergo retesting to confirm its successful resolution, thereby maintaining the overall health of our application.

Regular Compliance Checks

At regular intervals of every two weeks, we reinforce our commitment to compliance with project specifications by conducting system walkthroughs. These walkthroughs serve as a means to validate that the development efforts are aligned with the envisioned software and that the project remains on the intended course. This proactive approach ensures that deviations from the project's objectives can be promptly identified and addressed, ultimately resulting in a software product that consistently meets our clients' needs and expectations.

API Testing Policy

Endpoint Annotations For OpenAPI

All Post endpoints are required to be annotated using:
#[utoipa::path(post, tag = “Service tag", path = "/api/endpointName", request_body = structureRequest)]
The requestStructure must be annotated using the:
#[schema(example = json! { structureRequest {egparameter: [-90.0, 90.0],egparameter2: [90.0, -90.0], time: None } })]
This provides the necessary annotations for your endpoint to be available on SwaggerUI so that the frontend can test and use your endpoint before implementation.

Database Operations

Database operations are required to be wrapped and mocked. Try to reuse database operations as much as possible and when you write new database operations try to make them as generic and applicable as possible in order to avoid redundant mocking.
Database operations must be placed in a trait with the annotations:
#[automock] #[async_trait]
From there you can implement the trait and from there all functions that use database operations must take in the following:

async fn example(connection: Option<&Database>, db_functions: &dyn DBFunctionsTrait)

The methods for the Database trait must take in atleast the following:
async fn example<'a>(&self, query: Document, connection: Option<&'a Database>, options: Option<FindOptions>) -> Result<returnStructure, ApiError<'static>>;

Within tests, mocks are passed instead of a normal implementation of the Trait as is done in endpoints.

Tests

Tests that make use of other endpoints must be avoided, endpoint responses should rather be mocked and those endpoints should be tested individually.

In many cases an endpoint itself should not be tested and rather the functions the endpoint makes use of in its business logic should be tested due to mocking limitations of the database.

Any helper functions that are not covered in unit tests should be covered in unit tests. Integration test times must be fixed and suburbs must be able to be compared to the results produced by a similar service like ESP on the SwaggerUI endpoints.

CodeCov reports are deemed acceptable if the main business logic is all tested. The database functions should be tested manually as it is not possible to automate that testing. Constructors and structure initialization are not needed to be tested. Deserializers and serializers are not required to be tested.

The rust compiler will enforce the covering of edge cases as long as .unwrap() is not used. It is strongly discouraged to use .unwrap() unless you are unwrapping a database connection or something similar when you know that that state will be available when the rocket server has launched.