**Test Plan**

**Introduction**

The purpose of this document is to provide the information and framework required to plan and develop the activities of the test process of Digital Escrow service by WaloPay.

In this time, I don't have any documentation of the work already done, neither they documented the progress of the project, so the nature of this Test Plan is to test the Web Page based on the analysis of the critical features of the Project and the designed Test Cases to assure there is no issue.

Note: Some of the times or data presented in this document are mere assumptions since complete information is not available.

**Product Analysis**

This digital escrow service is a Web Application, developed on HTML, CSS and JavaScript, with the assumption that there is a connection with a Rest API.

The page provides the service of an intermediary between the seller and the buyer, so the target user is going to use this page to buy or sell with unknown people in a safe way.

The focus of the business is act as intermediary in the payments between customers charging a fee for services and in this way buy and sell safely and avoiding scams.

The users can register and account, log in, make transactions, consult transactions, deposit money and receives payment.

**Scope**

Manual Exploratory Test Cases for both UI and Interaction of the following components are going to be performed:

* Register Account.
* Transaction execution.

With a schedule of 2 weeks to deliver the project, the 1st week of testing is for the Manual Exploratory Test Cases, and the 2nd week for the Automated UI Tests. Bugs and Issues are going to be reported and tested as they came across the scheduled time.

**Resource Allocation**

* Quality Assurance Software Engineer
* Automation Engineer

Tech Stack

* JavaScript
* Test Café
* Selenium
* Postman
* SQL
* Excel
* Word

**Environment**

It is not known what kind of environment where used for the development and deployment of the project, so it’s an assumption these are the environments used, meanwhile is assumed the current delivered project handled over to QA by the developers is in a staging or production environment.

**Development Environment**

* Frontend Local Environment.
* Backend is hosted in an AWS EC2 Micro Instance.
* Database is hosted in an AWS RDS.
* Storage is hosted in an AWS S3.

**Testing Environment**

* Frontend AWS Environment.
* Backend is hosted in an AWS EC2 Small Instance.
* Database is hosted in an AWS RDS.
* Storage is hosted in an AWS S3.

**Staging/UAT Environment**

* Frontend AWS Environment.
* Backend is hosted in an AWS EC2 Small Instance.
* Database is hosted in an AWS RDS.
* Storage is hosted in an AWS S3.

**Production Environment**

* Frontend AWS Environment.
* Backend is hosted in an AWS EC2 Medium (Auto scalable) Instance.
* Database is hosted in an AWS RDS.
* Storage is hosted in an AWS S3.

**Tools**

* Word - Documentation.
* Excel O - Test Cases.
* Visual Studio Community - Source Code Editor for Software Development.
* C# - Programming Language.
* Selenium - Automation Testing Tool.

**Defect Management**

In case of finding any Bug or Issue, the following format is going to be used to create a Ticket:

* What the requirements said it’s going to happen.
* What actually happened.
* Steps for replication.
* Environment.
* Browser.
* Severity.
* Time.
* Person who found the Bug or Issue.

Any ticket is going to be reported to the Project Manager (PM), so between the PM and the QA can deliberate to which member of the development team the ticket is going to be sent for bug or issue fixing.

When a fix is sent back for testing for any ticket, the same steps for replication are going to be followed to reassure the bug or issue was actually fixed. If there are no more bugs or issues, the ticket will be closed, if not, the ticket is going to be send back to the respective developer to keep working on the fix.

In this case, the Tickets are going to be sent and managed via e-mail, as it is assumed there is not a project management tool in use for this project.

**Risk Mitigation Strategy**

Due to the fact that there is no documentation or information of the modules or what the developers progress was in the development time or if the project was already tested or not, to assure the project can be delivered on time. The biggest risk is to face any critical issue that can break the expected functionality of the project.

The priority is to focus on the most critical components and assure none of them fail, have any Issue or bug or any behavior that can break the fundamental experience.

**Deliverables**

* The core components and functionalities of the Web Application works as intended.
* There are no Bugs or Issues directly related to the core components and functionalities.
* There are no Bugs or Issues directly related to the UI and UX.

**Definition of Done**

At least, 80% of the Test Cases should have a “Pass” status and none of the remaining Test Cases with a “Not Pass” status should have a “Critical” priority.

There should be not a single blocker that can break the intended functionality of the Web App:

• The user cannot register an account or Log In into his/her account.

• The user cannot create a new transaction

• The user cannot consult his transaction

All parties involved should be notified and informed about the progress and completion of this Test Plan.