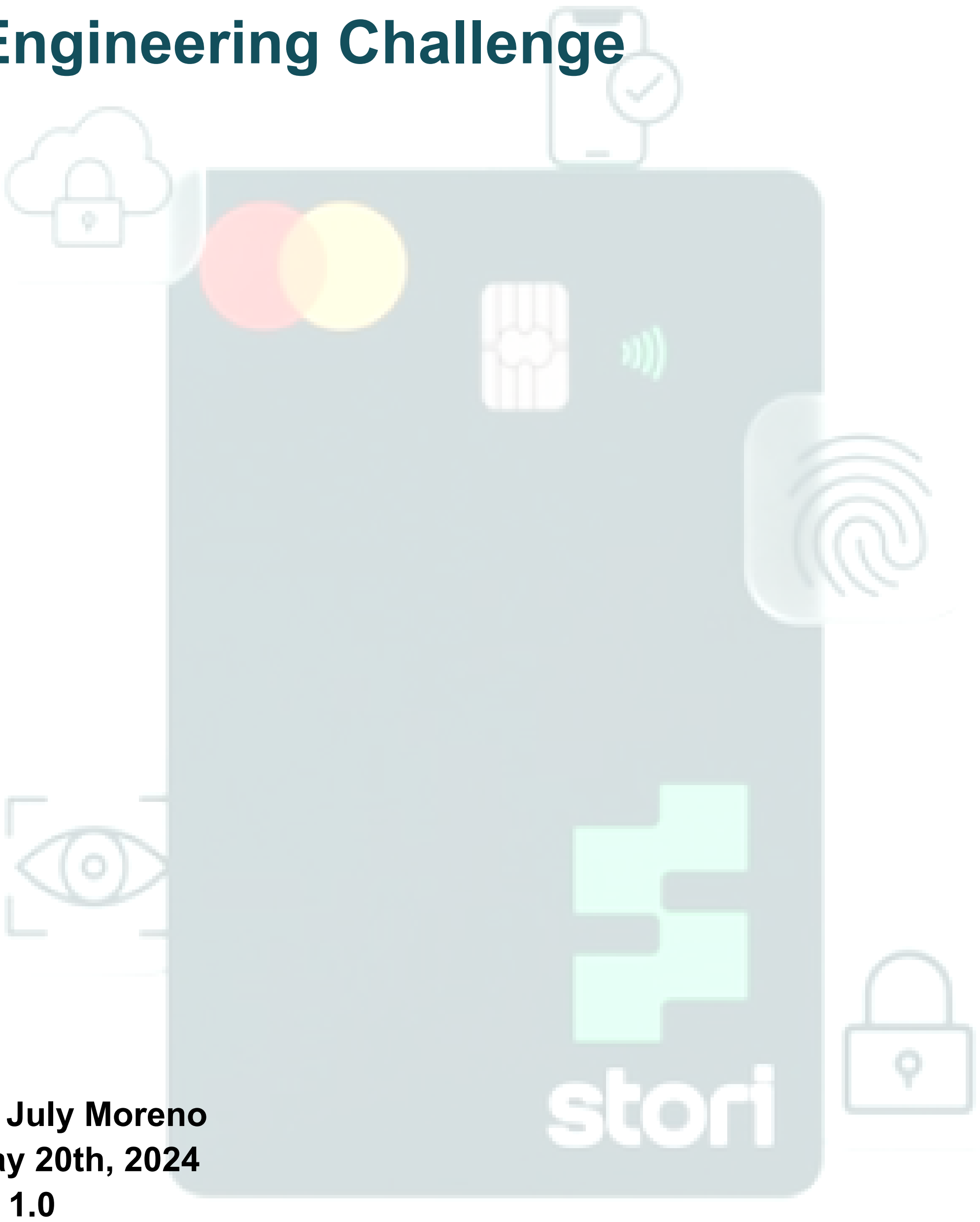
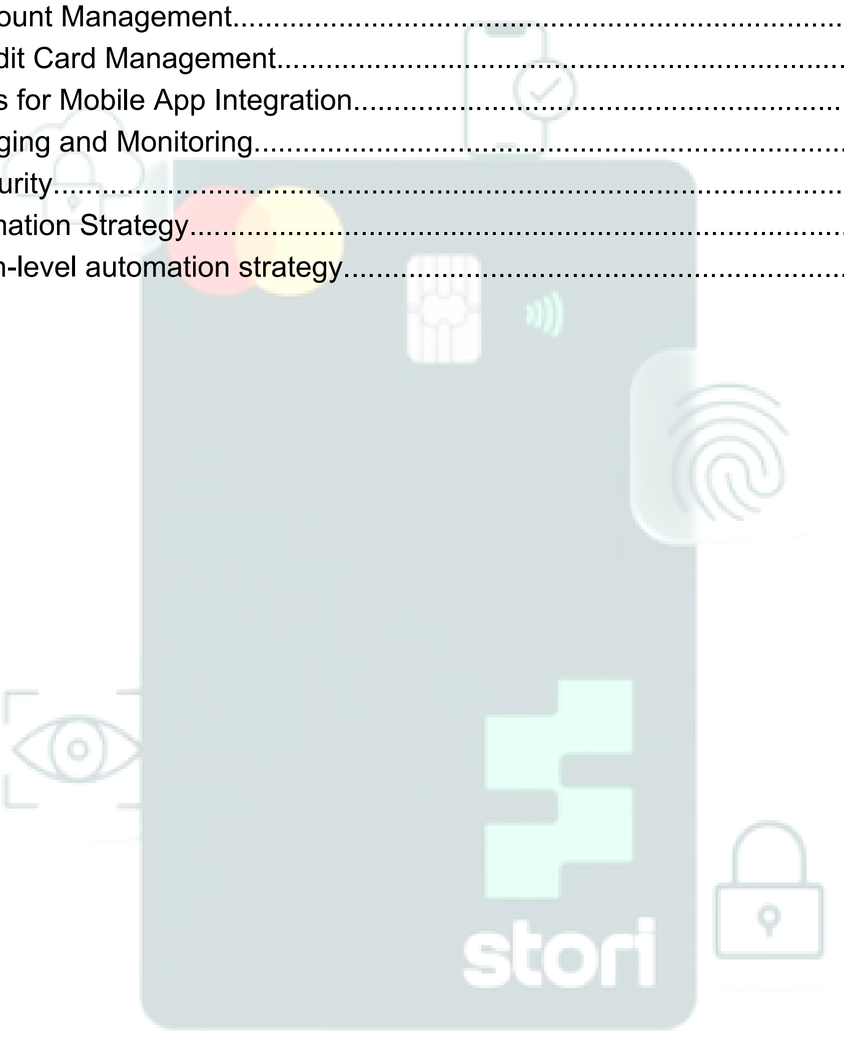
QA Engineering Challenge



Author: July Moreno
Date: May 20th, 2024
Version: 1.0
Code: QA-01

https://github.com/jmorenoj/storiCard

TABLE OF CONTENT



1. Leadership and communication

Scenario:
You have been hired as a QA Engineer for a startup company that is developing a new financial application. The company is in the early stages of growth, and the development team is still finding its footing. As the QA Engineer, you will be responsible for establishing quality assurance processes and collaborating with stakeholders to ensure the application meets their requirements.

1.1. Leadership and Startup Mindset

Describe your approach to building a quality-focused culture within a startup environment.

A quality-focused culture is a key factor when thinking about successful endeavors, plans, projects, and consequently, companies. No matter how big or old a company is, building quality must be a shared mindset. Referring to software projects companies and teams can easily think of appeal to the most recognized and accepted frameworks and guides (such as Scrum with the agile manifesto and ISTQB with the wide open testing types and techniques), but for anyone involved in a software project is essential to realize each company/project has its characteristics that needs to be analyzed and treated according to its nature.

A startup company and its products tend to evolve quickly because there is an important component of innovation and an increasing need for growth, so agility and adaptability are crucial when considering building a quality-focused culture.

At the same time, startups have a strong entrepreneurial spirit not just for founders, but people involved in these companies are characterized by flexibility, determination, resilience, creativity, and willingness to take risks. So, more than ever taking these characteristics in advantage to start building a quality culture in companies is essential because, beyond brands, successful startups are full of motivated people.

It is important to include quality as part of the company's DNA. The startup culture often promotes collaboration, innovation-driven thinking, open communication channels among team members, etc., but quality needs to be an additional factor that supports how the company wants to disrupt existing industries and solve the problems that differentiate it from traditional competitors.

Quality culture is a win-win, from a management perspective is widely needed because of the cost-effectiveness, time-saving, user experience improvement, and maintaining reputation, but also from an operational point of view, there is a solid foundation for future development.

From the very beginning "Building the thing right" and "Building the right thing" must go hand in hand, and this needs to be supported by a shared mindset of win-win where people in the startup environment beyond their role (founder, manager, developer, QA, BI, etc.) is always thinking on a customer-centric approach instead of a processes-centric approach.

Summarizing, when building a quality-focused culture we need to ignite people's passion for having a shared vision about the product and team goals, because quality is part of an everyone collaborative effort, instead of the last step of the chain in which there is an isolated area called "Quality".

1.2. Communication and Stakeholder Management

Outline your strategy for effectively communicating with technical and non-technical stakeholders

As companies are centered around people, it is crucial to understand how to communicate effectively with them. Communication should be clear and straightforward regardless of the technical level. In my experience, communication with stakeholders is most effective when:

1. Understanding the background, knowledge, and interests of the stakeholders to identify how communication can fit their needs
2. Using simple language. Sometimes technical terms can confuse non-technical stakeholders, so if non-technical people are clear on communication, you're all set! We need to identify when technological language is appropriate and at what level.
3. Provide context. Even if people have a general idea about the topic being discussed/communicated, providing an overview is always helpful for better engagement while communicating.
4. Using visual aids to simplify complex concepts. (Depending on the case)
5. Illustrating stakeholders with real-life examples. Sometimes associating a concept with something we previously know empowers stakeholders to communicate ideas.
6. Asking for questions and feedback. This is helpful to know how stakeholders are involved in the communication but also to identify some concerns.
7. Defining communication channels according to the nature of the stakeholders and the information. We need to learn that some

stakeholders may prefer a chat message instead of an email, but also some information can have better engagement in a presentation instead of a document.

8. Having common interests in a project, we need to maintain regular communication with stakeholders throughout the different phases to know about updates, progress, achievements, milestones, challenges, etc.
9. Exist transparently with information even when things are not going well. Honest and timely communication can help change the course of the project.
10. Identifying and respecting the confidentiality of the information that is being communicated. Everything has a time and a place

Is important to keep in mind that every stakeholder has their background and expertise, so each point of view is valuable and should not be discarded without being communicated.

1.3. Quality Assurance Strategies

Discuss your approach to creating a comprehensive quality assurance plan for the application.

Creating a quality assurance plan requires collaboration among the stakeholders involved in the software development life cycle. This means that the quality assurance plan should not be set in stone, as the application, company, and market evolve, the documentation and process should be updated based on feedback received from execution and changes in the technology landscape so it stays relevant.

Before creating a quality assurance plan it is important to understand the application, its purpose, and its users. including the services it provides, the regulations it must comply with, and the security measures it must have.

When this is clear, it's time to define what quality means for the application. The next steps can help to structure the quality assurance plan:

1. **Define goals and objectives:** Define in terms of functionality, performance, security, usability, and compliance with regulations what quality is and if those are measurable and achievable.
2. **Identify stakeholders:** Identify who is going to be involved in the development and operation process.
3. **Understand requirements:** To understand what needs to be tested we precise clear understanding of the functional and non-functional requirements.

4. **Describe Test Strategy:** Document the types of testing to be performed during the different stages of development, the tools to be used, and the resources required.
5. **Create Test Plan:** Create test plans (as detailed as needed) for each type of testing that should take place throughout the SDLC including the test objectives, the test data to be used, the expected results, and the passed/failed criteria.
    a. **Establish the data management approach:** Determine how to get, store, and use the data required for the different levels of testing ensuring privacy.
    b. **Define roles and responsibilities**: According to the team members involved in the quality role, specify the responsibilities among the quality assurance activities.
6. **Define test cases:** Create test cases (as detailed as needed) that cover all the functionalities in the application.
7. **Set up a defect tracking process:** Establish the process for tracking the defects found when executing the testing, including the definition of the severity and urgency levels.
8. **Automation:** Identify automation opportunities to increase efficiency.
9. **Document QA procedures:** Document the procedures related to the QA process including how to perform and the tools used for each type of test.
10. **Monitoring and reporting:** Compare testing activities' progress against the target defined in the quality assurance plan to provide reports and metrics about the testing process and product quality.
11. **Establish continuous improvement process:** Review the results of the tests and use them to improve the application, the development process, and the QA strategy over time.
12. **Training and Knowledge Transfer:** All stakeholders need to be trained on QA processes to understand the role and responsibilities including the QA tools.

2. Technical Challenge

Scenario:

The startup company has developed a financial application with the following features:

- User login and registration
- Displaying the user's account balance
- Blocking and unblocking a user's credit card

Challenge:

As the QA Engineer, you are responsible for creating a comprehensive test plan to ensure the quality of the application.

2.1. Test Scenarios

Develop a set of test scenarios to cover the login and registration functionality.

📕 storiCard_testScenarios_julymoreno

2.2. Test Cases

For each of the test scenarios, write detailed test cases, including test steps, expected results, and pass/fail criteria.

📘 storiCard_testCases_julymoreno

2.3. Backend Functionality Testing

Identify the key backend functionalities that support the application's features.

We can split the application feature into three main entities. Those entities would require database operations under the CRUD functions (Create, Read, Update, and Delete)

1. User Management

The backend should provide APIs for registration, validate the input data, and store credentials

- **User registration:** This functionality allows new users to register in the app. Includes storing user information in the database
- **User login:** This functionality allows existing users authentication. Include checking the provided credentials against the stored information in the database.

- **User profile management:** This functionality allows users to view and update their profile information.
- **Authorization:** This functionality aims to ensure the user can see just the information that is authorized.

2. Account Management

The backend should provide APIs for fetching and displaying user account details

- **Display the account balance:** This functionality allows retrieval and displays the current account balance of the user from the database.

3. Credit Card Management

The backend should provide APIs to block or unblock a user's credit card and notify users about the changes:

- **Block credit card:** This functionality allows blocking a user's credit card. It involves reading the credit card's current status and updating it in the database.
- **Unblock credit card:** This functionality allows unblocking a user's credit card. It involves reading the credit card's current status and updating it in the database.

Also, some additional backend functionalities need to be designed to support the features of the mobile application:

4. APIs for Mobile App Integration

The backend should expose RESTful APIs that the mobile app can consume to perform functionalities like user authentication, account management, and credit card management.

5. Logging and Monitoring

As part of the backend functionalities, there is a need to implement logging for important events and errors, also for auditing and troubleshooting. Additionally, it is important to include monitoring tools at the backend level to track system performance and ensure uptime.

6. Security

The backend should store sensitive information such as user passwords and credit card details, so secure storage needs to be part of the functionalities. Also, the backend supports the application to manage the user session and prevent attacks.

2.4. Automation Strategy

Propose a strategy for automating the test scenarios, including the tools and frameworks you would use.

Going forward with the automation strategy for the app, I suggest defining a strategy based on the following steps. However, the choice of tools and frameworks may vary depending on the programming languages the team is comfortable with, the mobile platforms we are targeting (iOS, Android, or both), and the specific architecture of the app:

1. **Test Scenarios Identification:** From the test scenarios library, identify for each functionality positive and negative test cases that could be objective as part of the automation.
2. **Select Automation Tools:** According to the business needs, budget, and technical knowledge, select the automation tools that fit with the company. I would suggest:
   - UI testing:
     - Mobile app: Appium
     - Web app: Robot framework / Cypress
   - API testing:
     - Postman
3. **Select Automation Frameworks:** According to the tools and languages that support the application we need to select a framework like TestNG or JUnit for Java projects or PyTest for Python-based projects.
4. **Training and knowledge transfer:** When selecting test cases to be automated, and the tools and frameworks it is important to train all the people involved in the automation process to make the scripting process effective.
5. **Writing Test Scripts:** When writing test scripts, think of techniques like data-driven testing (DDT) to cover various input combinations and scenarios.
6. **Integration with a Continuous Integration (CI) pipeline:** Integrate the test automation suite with CI tools (such as Jenkins) to be able to run the test automatically whenever needed.
7. **Reporting and Analysis:** Generate testing reports and analyze the results to identify failures, performance issues, and improvement opportunities.
8. **Maintenance and Updates:** Update the test scripts according to the UI and functionality changes, but maintain a repository of versions.

1. High-level automation strategy

Tools/Frameworks	● Appium ● Robot Framework ● TestNG/JUnit ● Postman/RestAssured ● Java/Python

Feature	User Login and Registration
Strategy	Create automated test scripts to validate both positive and negative scenarios. ● Positive scenarios include successful login/registration with valid credentials. ● Negative scenarios include attempts to log in with invalid credentials, register with an already registered email, or register with invalid data.

Feature	Displaying the User's Account Balance
Strategy	● Automate the verification of the account balance display after successful login. ● The balance should be cross-verified with the database or through API testing to ensure the correct balance is displayed.

Feature	Blocking and Unblocking a User's Credit Card
Strategy	● Automate the process of blocking and unblocking the credit card from the user interface. ● Verify the card's status in the database or through API testing after each operation. ● Automate the scenario where a user tries to transact with a blocked card. The transaction should fail. ● Automate the scenario where a user tries to transact with an unblocked card. The transaction should work.