Non-Functional Requirements

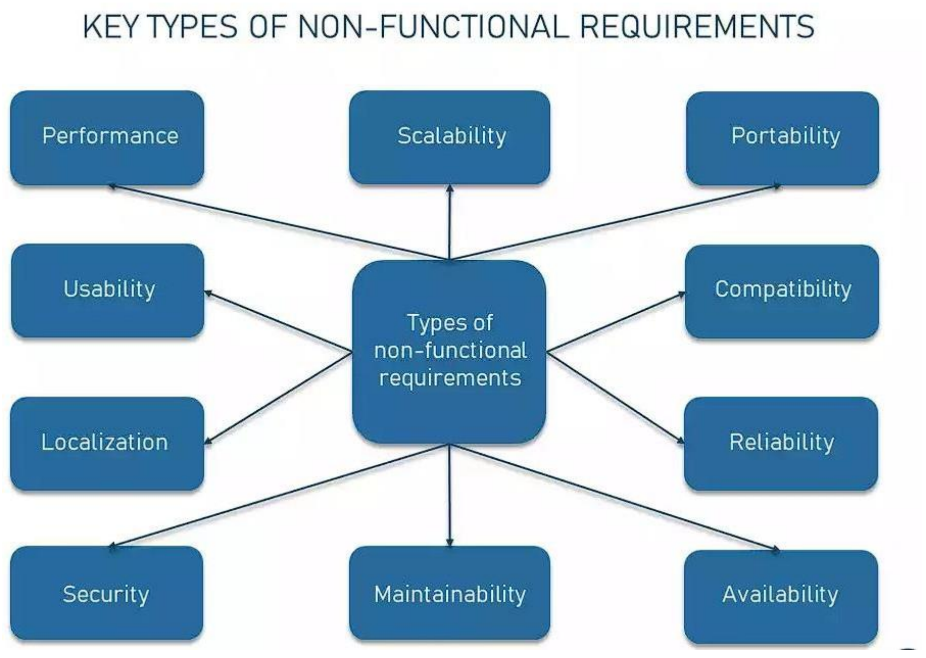
This document was created in the scope of defining the Non-Functional Requirements for our project. As our solution is not live and it does not have any real customers, some of these requirements will be made hypothetically. This will be made with the help of Flutter's Documentation and other sources.

We will continue by describing the conditions under which a solution must remain effective. Out project has the goal of building an application that can help restaurant staff mainstream the process of reservations and orders.

Compatibility of our application will be one of our main focuses of our project, because we will be using the Flutter framework, which is compatible with a variety of different platforms.

Every restaurant will have its own instance of the application and its own local database and if the owner so desires, there is the option of realizing an integration between the restaurant's site and the application to facilitate the creation of reservations or orders even further. Because the reservations are not equally distributed during the day, we will need to make sure our application works well during the most crowded timeframe, which would probably be during the night.

We will follow and expand on the next NFR model:



Next, we will define the system attributes:

Performance and scalability

As we mentioned previously, the reservations are not equally distributed during the day, we will need to make sure out application works well during the most crowded timeframe, which would probably be during the night. So, our application needs to be scalable enough to manage large volumes of data.

For the testing of the performance, we will have the following scenarios in mind:

- testing the speed of creating and updating a reservation when there are a few reservations in the database
- testing the speed of creating and updating a reservation when there are a large number of reservations in the database
- testing the speed of creating and updating an order when there are a few orders in the database
- testing the speed of creating and updating an order when there are a large number of orders in the database
- testing the speed of creating and updating a product when there are a few products in the database
- testing the speed of creating and updating a product when there are a large number of products in the database

Performance wise it needs to be able to do any of those inputted commands(tests) in less than one second.

Portability and compatibility

Compatibility our application will be one of our main focuses of our project, because we will be using the Flutter framework, which is compatible with a variety of different platforms.

Flutter defines three tiers of support for the platforms on which it runs:

- Google-tested platforms are automatically tested at every commit by continuous integration testing.
- Best-effort platforms, supported through community testing, are platforms we believe we support through coding practices and ad-hoc testing, but rely on the community for testing.
- Unsupported platforms, which are platforms that might work, but that the development team doesn’t directly test or support.

You can find here the full list of the Supported platforms. As you can see, Flutter offers compatibility with almost every platform. It doesn't conflict with other applications or processes within these environments.

For the testing of the compatibility, we will have the following scenarios in mind, we will test unsupported platforms against Google-tested platforms:

- different versions of Android (for Android SDK 15 and below it should not work, and for SDK 19 and above should work)
- different versions of iOS (for iOS 10 and below and arm7v 32-bit iOS it should not work, and for iOS 14 and above should work)
- different windows (for Windows Vista and below and any 32-bit platform it should not work and for Windows 10 it should work)

Regarding the portability part, the users will not be required to download anything besides the application itself.

From here on, the discussion will be hypothetical, as we cannot measure or test the following requirements because the nature of our app.

Reliability, maintainability, availability

Flutter's reliability and efficiency have already been time-tested by such giants as Google Ads with millions of downloads and daily users, based on this guide, which presents the benefits and drawbacks of Flutter.

Security

There are different methods through which we can secure our Flutter application: code obfuscation, secure API keys, Flutter jailbreak detection, ensure secure network connections, use only necessary permissions, secure user data, local authentication, secure developer identity, secure CI infrastructure and of course an account should be locked after 3 failed attempts to login, based on the username. More about these can be found/read here.

Usability

The application should be easy to use and intuitive for a customer to use.

Localization

Out application might be deployed to users who speak another language then we'll need to internationalize it. That means we could write the app in a way that makes it possible to localize values like text and layouts for each language or locale that the app supports. Flutter provides widgets and classes that help with internationalization and the Flutter libraries themselves are internationalized. Flutter localization uses ARB (Application Resource Bundle) files to house its translations by default. These are files written in JSON syntax, based on the Flutter's documentation.