# Software architecture document

## ****Introduction****

This document provides a comprehensive architectural overview of the system, using several different views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made to implement the software solution. It also to help developers more easily grasp the software’s modules and components, without digging into the code, while also being incredibly useful as it gives an oversight of the entire system and exposes any potential pitfalls the developers might experience.

### ****System context - C1****

Diagram

Description automatically generated

From the system context diagram, it can be seen that the software is going to make use of an external API for getting the weather forecast. This means that the developers can use the already implemented solution, since implementing an entire weather API and forecast is out of scope for this project.

### ****Containers and technology - C2****

Diagram

Description automatically generated

From the diagram it can be seen that the software system is composed from two main containers:

* iOS application using SwiftUI
* API of Tomorrow.io

#### ****Separation of concerns****

With the two main containers divided in this way it is ensured that the different divisions of the software are properly separated. The iOS application takes care of all the UI and presentation logic required for using the application. The weather API takes care of the forecast, since it is an incredibly complex and an entire profession by itself, using different sensors and methods to predict the weather.

#### ****Why SwiftUI****

SwiftUI is the most modern way to design applications for the whole Apple ecosystem. The other option is UIKit, which definitely has its own advantages like storyboards, a drag-and-drop solution to building user interfaces. For this reason, it does not require a strong programming background. However, building a user interface programmatically (without storyboards) in UIKit is significantly more difficult compared to SwiftUI. Not only that, but UIKit is known as an imperative framework, which simply means you are stating how to do something. In contrast SwiftUI, is a declarative framework, meaning you are declaring something you want to happen. It is also very well documented in Apple’s docs, which is the reason we chose to build our application with SwiftUI

#### ****Why Tomorrow.io****

#### ****Tomorrow.io is a very intuitive and user-friendly API. They have a lot of solutions like cutting-edge, real-time weather data, hyperlocal data up to 14 days in the future for any location on the globe, integrations for aviation, transportation, construction, and mining. For this project we are using their weather APIIt is simple to use for and implement for our small-scale project, providing a free tier with up to 25 requests per hour, 1 user and 1 location, which is more than enough to provide a proof of concept for this project.**** It features has great functionality, documentation, customer support and HTTPS. It is also by a magnitude of large-scale companies, businesses and governments, making it secure and properly maintained.

### Components - C3

Graphical user interface, diagram, application

Description automatically generated

From the component diagram it can be seen in more detail how the iOS client app is structured. The user interacts with the UI, which is written in SwiftUI. The views holds information about their state and other objects. The logic and communication layer are stored in the repositories. By following Apple’s code principles, they are injected into the application as an environment object that can be used in the views. Since this application is dependent on the phone’s location, that logic is separated into its own component that deals with requesting and sharing the device’s coordinates. This component can be injected in any class that requires its functionality. Requests to the Tomorrow.io API are done through the repositories, which parse the JSON and return domain objects that can be used in the application for presentation or logical operations.