Technical Design Document

# Version Control

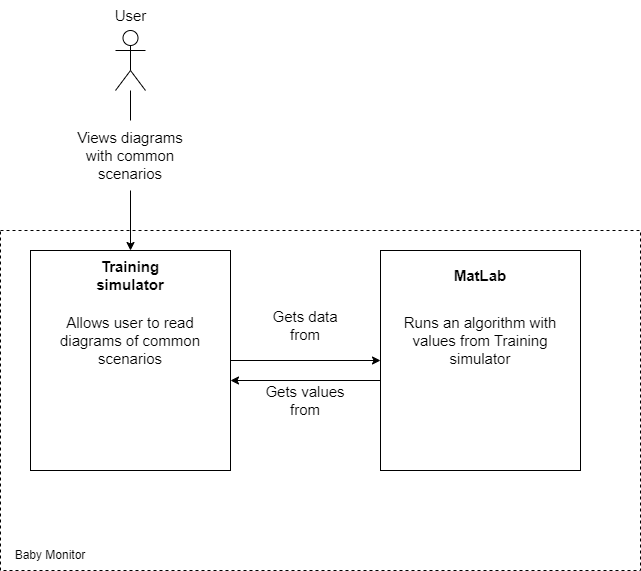
|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Sprint | Changes |
| 0.1 | 27-02-2023 | 0 | Initial Document |
| 0.2 | 02-03-2023 | 0 | C4 Models Added |
| 1.0 | 23-03-2023 | 1 | Motivation of choices |
| 1.1 | 03-04-2023 | 2 | Description on C1 & C2; Update and create C2 & C3 diagrams |

# Architecture Diagrams

## C4 Models

### System Context Diagram

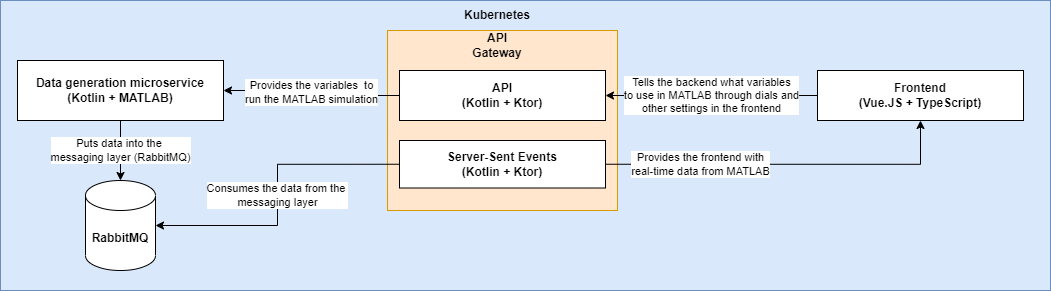
Level 1



The end user will interact with the Baby Monitor simulator using the UI. This is part of the Training Simulator which will send any new values to the external MATLAB component. This component will use the new values to calculate the graphs. The graph data is sent back to the Training Simulator, which will then display the graphs for the user to further interact with.

## Container Diagram

Level 2



### Data generation microservice

Continuously generates data using MATLAB to feed RabbitMQ which in turn feeds Server-Sent Events (SSE) which feeds the graphs in the frontend.

### RabbitMQ

Provides a messaging layer between the data generation microservice and the frontend through SSE. RabbitMQ is used in conjunction with a MATLAB microservice which will continuously generate data from the MATLAB model and put it into RabbitMQ, the server-sent events part can then consume these data points and relay them to the frontend. Another advantage of this approach is that data can be replayed from various points to the client as well as throughput can be tracked.

### Server-Sent-Events

The SSE module within the API provides the layer between the frontend and RabbitMQ (and thus the data generation service) which is responsible for feeding the graphs in the frontend.

### API

Provides the interaction layer between the frontend and various other services within the ecosystem. The API is responsible for feeding the variables used for the generation of data to the data generation microservice as well as providing the SSE layer between the frontend and the messaging system.

### Frontend

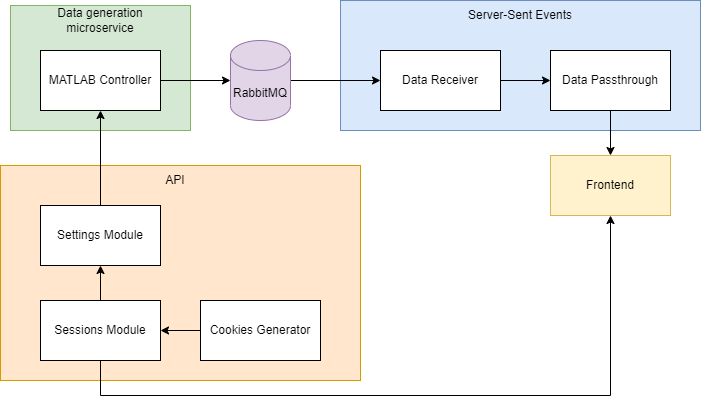
The frontend is a single one-page application meant to be used by the end-user, this frontend contains the graphs and access to the settings for the simulations. The frontend will be hosted in its own container and can be scaled up to deal with high demand through Kubernetes.

### Kubernetes

Kubernetes will be used in order to guarantee both availability and scalability of every component within our infrastructure. This will make sure that we can reliably scale up as demand fluctuates as well as guarantee that if any one component goes down it will be automatically restarted, and the workload can be picked up by another instance.

## Component Diagram

Level 3



### Frontend

The Frontend will be where the end user interacts with the application. Here they make any data inputs or setting changes that will be run through the required parts in the API. It then receives a response via the SSE component. The Frontend itself has the ability to import and export data into files, which is why this is not specified in any other parts.

### Cookies Generator

Generates a cookie ID (probably UUID or similar) to give to the client to make each client’s settings uniquely identifiable without the need for an account. Though this has the drawback of being limited to one browser on one device. We felt that it was unnecessary to make a whole account service for this type of application, since this would be too much effort for the end users, on top of the application needing to be generally usable by anybody in the work field at a moment’s notice.

### Sessions Module

The sessions module will be responsible for keeping track of all the different sessions that are currently active and storing their information and settings.

### Settings Module

The settings module will be responsible for keeping track of settings (based on the session) and making changes to said settings based on REST API requests.

### MATLAB Controller

Generates MATLAB data and puts it into RabbitMQ.

### Data Receiver

Reads the data from RabbitMQ and passes it to the passthrough.

### Data Passthrough

Passes MATLAB data to the relevant frontend client based on their session.

# Motivation of Choices

## Backend Technology Stack

Kotlin, Ktor and Gradle

* Server-side support with JVM technology:
  + Platform independence
  + Common API For runtime
  + Security
  + Moment Management
* Async support (coroutines)
* Interoperable with Java
  + All the benefits and stability of Java
  + All the new features and functionality of Kotlin
* Minimal Boilerplate
* Ktor natively supports Kotlin source code.
* Gradle is significantly faster than other build tools such as Maven.
* Null safety in Kotlin

Additionally, we chose to explore Kotlin with the Ktor framework to also learn new technologies since our group knowledge in these technologies is minimal.

## Frontend Technology Stack

Vue, TypeScript and Vite

### Vue

Vue is “*an* ***approachable****,* ***performant*** *and* ***versatile*** *framework for building* ***web user interfaces****”*. For our stakeholders, we must build a **web application** that exactly requires these factors. It needs to be an intuitive (**approachable**) for the clinicians, fast (**performant**) to display real-time data and needs to support being able to show graphs (**versatile**).

### TypeScript

To avoid common runtime errors, TypeScript is a strongly typed implementation of JavaScript, thus providing better tooling at any scale. Errors can also be caught earlier on in the editor itself and the type of inference functionality will be useful in any context. We therefore see no reason to use JavaScript compared to TypeScript for a shared group project.

### Vite

Vite is a front-end tooling system that includes many useful functionalities to improve the workflow and efficiency of our group.