AREA GLOBAL DOCUMENTATION

Felix Stavonhagen, Kentin Paille, Paul ThiebaultBenjamin Desprets

Contents

Introduction

Welcome to the technical documentation for the AREA project.

AREA, inspired by the functionality of IFTTT (If This Then That), empowers users to create intuitive automations triggered by events within various web services like GitHub, Slack, and Discord.

This document elucidates the code conventions, architectural design, technological choices, and other pivotal aspects that govern the development and maintenance of the AREA project.

With this document you will gain a better understanding of the project's file and folder structure, preferred syntax, and more, which collectively contribute to a robust and scalable solution capable of seamlessly integrating with an array of web services.

Code conventions

Adhering to established code conventions is a hallmark of professional software development, fostering readability, maintainability, and consistency across the codebase. The AREA project embraces a set of conventions tailored to align with industry standards and the peculiarities of the technologies employed.

Variable and File naming

In the realm of the AREA project, the naming convention predominantly follows the **camelCase** style for both variables and files. This convention promotes readability and is widely adopted in many programming languages including **JavaScript** and **TypeScript**, which constitute the backbone of the project.

Folder structure

The folder structure of the AREA project is logically organized to reflect the **architectural separation** of concerns. Key directories like **client**, **server**, and **db** encapsulate the respective components, ensuring a clean and intuitive organizational scheme.

Database naming

Contrastingly, the database adopts the **snake_case** naming convention, diverging from the camelCase used elsewhere. This divergence is justified as **snake_case** is a common convention in **PostgreSQL** which

the project employs for data management. It aids in ensuring a clear distinction between database entities and other elements of the code, besides aligning with the customary practices within the PostgreSQL community.

React components

In alignment with industry norms, the naming convention for React components within the project adheres to **PascalCase**. This convention not only conforms to the established React community standards but also facilitates easy identification of components within the codebase.

Preferred syntax

The syntax adopted in the project conforms to the widely accepted standards of the technologies employed. This includes adherence to the ES6+ features in JavaScript, and the utilization of TypeScript for static typing, which significantly enhances code reliability and maintainability.

The code conventions outlined herein form an integral part of the AREA project's commitment to quality, consistency, and professional software development practices. By following these conventions, the project ensures a cohesive and understandable codebase that is in sync with industry standards and best practices.

Architecture

The architectural design of the AREA project is delineated into a triad of primary components: the client, server, and database, each housed within its respective directory in the project tree. This division not only encapsulates the underlying structure but also exhibits a clear separation of concerns, fostering maintainability and scalability.

Diagram

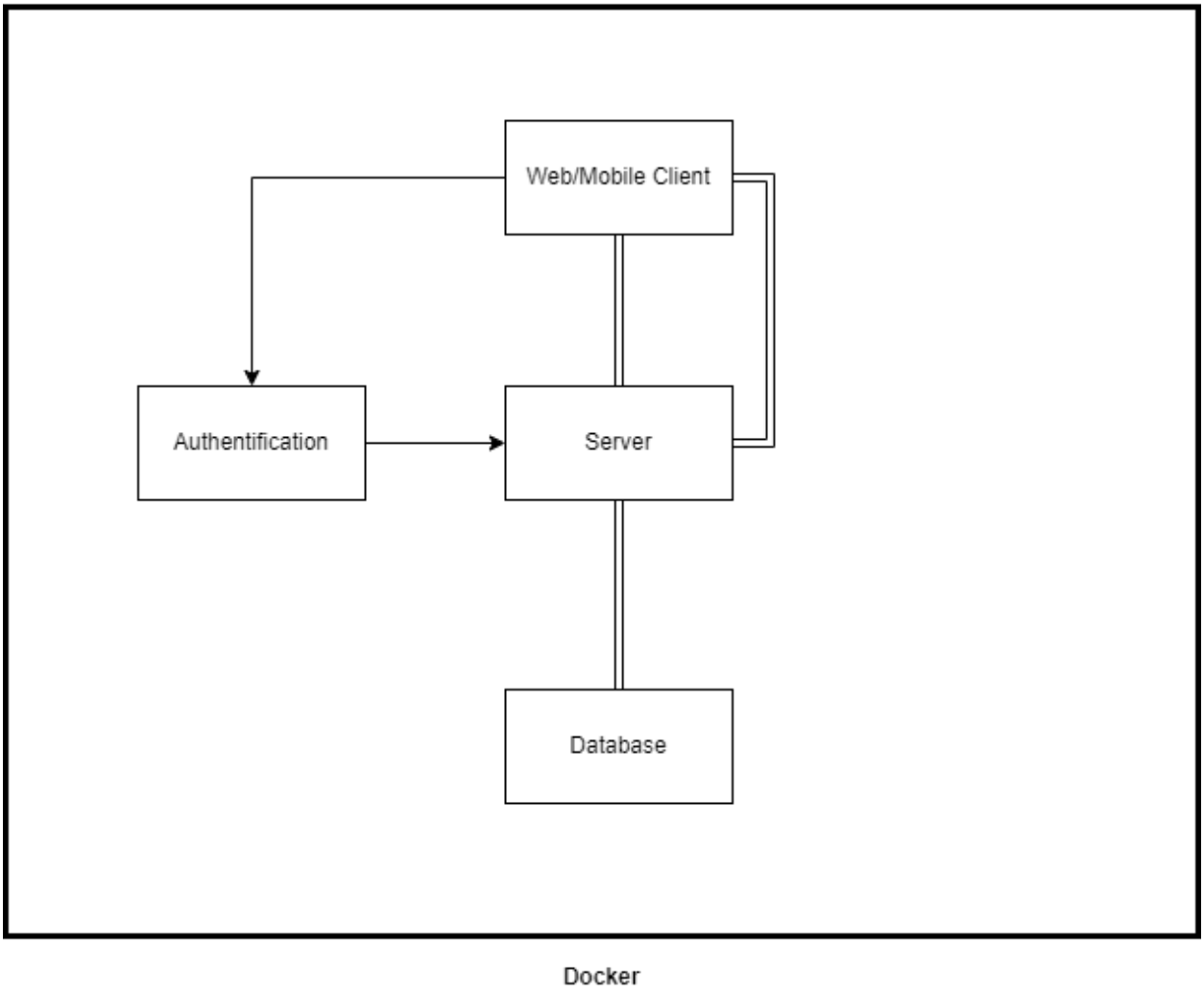

Figure 1: Architecture diagram

Client

The client component is bifurcated into mobile and web sub-directories, each tailored to deliver a seamless user experience across different platforms. The mobile directory is structured around React Native, facilitating the development of a cross-platform mobile application, while the web directory is engineered using React, ensuring a responsive and intuitive web interface.

Server

The **server** directory, this component is the linchpin that bridges the client and database components. It's architected around **NestJS**, a progressive **Node.js** framework, offering a robust foundation for building efficient, reliable, and scalable server-side applications. The directory structure within the **server** is meticulously organized, with sub-directories like **src** housing the core application logic, **templates** for email rendering, and **test** for end-to-end testing.

Database

The **db** directory encapsulates the database component of the AREA project. Utilizing PostgreSQL, encapsulated within a Docker container, this component provides a sturdy and reliable data storage solution.

The postgres_config.conf file within this directory serves as the configuration nexus, ensuring optimal database performance and security.

Dockerisation

The infusion of Docker within the AREA project epitomises the modern approach to ensuring consistency across development, testing, and production environments.

The docker-compose.yml file is the fulcrum around which the multi-container Docker applications pivot, orchestrating the seamless interaction between the client, server, and database components.

Technology Decision Reports

The AREA project leverages a selection of modern technologies, each chosen to address specific requirements and collectively contribute to the project's overall goals of providing a robust, scalable, and user-friendly automation solution.

React and React Native

For crafting intuitive and responsive user interfaces, the decision was made to employ React for the web client and React Native for the mobile client as well as the previous experience from the team members in these languages.

React's component-based architecture facilitates modular development, while React Native enables the creation of native mobile applications for both Android and iOS from a single codebase.

NestJS

On the server-side, NestJS was chosen for its alignment with modern JavaScript and TypeScript, providing a solid foundation for building efficient and scalable server-side applications. Its modular architecture, along with its support for Dependency Injection, makes it a suitable choice for the AREA project.

PostgreSQL

PostgreSQL was chosen as the database solution due to its proven reliability, data integrity, and extensibility. It offers a robust data storage and management solution that is crucial for the effective handling and retrieval of data within the AREA project.

Docker

Docker has been integrated to ensure consistency across different operational environments. It simplifies the deployment, scaling, and operations of the AREA application by encapsulating components within containers. The docker-compose.yml file orchestrates the interaction between these containers, enabling a streamlined infrastructure setup.

The technological choices within the AREA project were meticulously made, each after a thorough evaluation of its merits and alignment with the specific needs of the project. These decisions have collectively contributed to a cohesive and well-structured application conducive to effective automation and integration across various web services.

Client Directory (Mobile & Web)

The client directory is a cornerstone of the AREA project, housing the codebase for both the mobile and web interfaces. It is bifurcated into two sub-directories: mobile and web, each tailored to cater to the

respective platforms while retaining a cohesive structure that promotes ease of navigation and maintenance.

Mobile

The **mobile** sub-directory is structured around **React Native**, a framework that facilitates the development of cross-platform mobile applications. Here are some key files and their purposes:

App.tsx: This is the entry point of the mobile application, where the main App component is defined and rendered.

package.json: Holds metadata and defines dependencies, scripts, and other configurations necessary for the mobile application.

tsconfig.json: Specifies the root files and the compiler options required to compile the mobile application.

node_modules: A directory where all the dependencies specified in package.json are installed.

Web

The web sub-directory, on the other hand, is engineered around React, ensuring a responsive and intuitive web interface. Below are some key files and their purposes:

pages: This directory contains the various pages of the web application, each represented by a React component.

components: Holds reusable components that can be shared across different pages of the web application.

package.json: Like the mobile sub-directory, it holds metadata, dependencies, scripts, and other configurations necessary for the web application.

tsconfig.json: Specifies the root files and the compiler options required to compile the web application.

node_modules: A directory where all the dependencies specified in package.json are installed.

The organisation of the client directory into mobile and web sub-directories enables a clear separation of concerns, ensuring that the codebase remains organised and maintainable. The delineation of key files

within these sub-directories provides a solid foundation, ensuring that developers have a clear understanding of the structure and the configurations that drive the client-side of the AREA project.

Server

The server directory encapsulates the **server-side logic** of the AREA project, embodying a structured and organized codebase that facilitates efficient development and maintenance.

The directory is meticulously structured into sub-directories and files, each serving a distinct purpose in the server-side operations of the project.

Sub-Directories

Src

The src sub-directory is the crucible of the server-side application logic. It houses the core modules, controllers, services, and other vital elements that drive the functionality of the AREA project.

Templates

The templates sub-directory contains template files used for rendering emails. It plays a pivotal role in ensuring that the email notifications sent from the AREA project are well-formatted and user-friendly.

Test

The test sub-directory is dedicated to housing end-to-end testing files, ensuring that the server-side logic operates as expected under various scenarios and conditions.

Key files

Dockerfile

The **Dockerfile** within the server directory defines the Docker image for the server component of the AREA project. It specifies the base image, dependencies, and the commands necessary to build and run the server component within a Docker container.

This file is instrumental in ensuring a consistent and reproducible environment for the server component across different stages of development.

nest-cli.json

The nest-cli.json file is a configuration file for the NestJS CLI. It defines the structure and settings for the NestJS application, aiding in the automation of mundane tasks, and ensuring a standardized setup for the server component.

The server directory's organized structure and the discerning placement of key files and sub-directories are emblematic of the thoughtfulness invested in architecting a scalable, maintainable, and efficient server-side setup. Each file and sub-directory are meticulously curated to contribute towards a robust server-side framework capable of handling the diverse functionalities required by the AREA project.

Database

The db directory is integral to the AREA project, serving as the nexus for data storage and management. It is structured to ensure an organized, secure, and efficient handling of data, which is pivotal for the seamless operation of the AREA project.

Key files

Dockerfile

The dockerfile within the db directory delineates the instructions for building a Docker image for the PostgreSQL database. It specifies the base image, environment variables, and other configurations necessary for setting up and running the PostgreSQL database within a Docker container. This

encapsulation facilitates a consistent and reproducible database environment across different stages of development and deployment.

postgres_config.conf

The postgres_config.conf file is a configuration file for the PostgreSQL database. It contains settings related to the database's performance, security, and other operational parameters. The configurations within this file are instrumental in ensuring that the database operates efficiently and securely.

The organization of the db directory, coupled with the key configuration files it houses, lays a solid foundation for robust data management within the AREA project. The thoughtful structuring and the meticulous configuration settings encapsulated within this directory contribute to a dependable and efficient database setup, ensuring that the data integral to the AREA project's operation is handled with precision and security.

Technology transition

The evolution of a project often necessitates a re-evaluation of the initial technological choices to better align with the emerging requirements and challenges. The AREA project witnessed such transitions, pivoting from certain technologies to others that offered a more streamlined, reliable, or efficient solution to the tasks at hand.

Transition to Next.js from React Native for Web

The initial choice of React Native for Web for the web client was driven by the allure of code reuse between mobile and web platforms. However, as the development progressed, it became apparent that React Native for Web introduced complications that hindered the project's momentum.

The decision was made to transition to Next.js, a React-based framework that not only alleviated the encountered issues but also provided a more streamlined development experience, better SEO optimisation, and server-side rendering capabilities.

This transition has significantly contributed to accelerating the development pace and enhancing the overall user experience on the web platform.

Transition to Custom Authentication System from KeyCloak (OAuth)

Initially, KeyCloak was employed to manage OAuth authentication within the AREA project. While it provided a robust and feature-rich solution, the need for a more tailored authentication system emerged as the project evolved.

The decision was thus made to pivot to a custom-built authentication system that is currently a Work-In-Progress (WIP). This new system is being designed to align closely with the specific authentication and authorization requirements of the AREA project, offering a more integrated and tailored approach to managing user identities and access controls.