**Asignatura: Taller de Diseño y desarrollo de Soluciones**

**Sección: D-IEI-N4-P1-C1**

**Nombre del docente:** Armin Vladimir Brun Ruth

**Nombre de los integrantes del grupo: Nicolas Ivan Vicencio Ordenes**

**Fecha de entrega 22/09/2022**

Guía Informe de Proyecto

Unidad 1: Arquitectura de la solución, almacenamiento y plan de pruebas

**Contenido**

[I. Introducción 3](#_Toc114655723)

[II. Objetivo 3](#_Toc114655724)

[III. Patrón de arquitectura 4](#_Toc114655725)

[IV. Estándares de programación segura. 5](#_Toc114655726)

[V. Servicios de backend y Almacenamiento de datos cloud 6](#_Toc114655727)

[VI. Plan de pruebas 8](#_Toc114655728)

[VII. Conclusiones 9](#_Toc114655729)

[VIII. Referencias bibliográficas 10](#_Toc114655730)

1. Introducción

Este documento contempla la definición de patrones de diseño para la aplicación organiApp, Veremos el patrón flux y sus componentes. Luego describiremos los estándares de programación segura incluyendo roles, perfiles y la seguridad de la aplicación.

Luego haremos una comparativa entre diversos servicios de backend con base de datos y detallaremos sus funcionalidades, similitudes y diferencias.

Finalmente Describiremos nuestro plan de pruebas para el correcto desarrollo de la aplicación.

1. Objetivo

**Objetivo General:**

Definir que herramientas y metodologías se adaptan de mejor manera al desarrollo de la aplicación, con el fin de aumentar en seguridad y rendimiento.

**Objetivo Especifico:**

Para alcanzar el objetivo general se ha planteado completar los siguientes objetivos específicos

* Identificar que patrón de diseño es el más específico dados los requerimientos de los clientes
* Definir el proveedor de bases de datos en la nube para nuestra aplicación.
* Definir tipos y rangos de usuarios en base a sus responsabilidades/beneficios.
* Establecer estrategias preventivas de seguridad para prevenir ataques cibernéticos.
* Definir un plan de pruebas con sus criterios de aceptación, suspensión y reanudación.

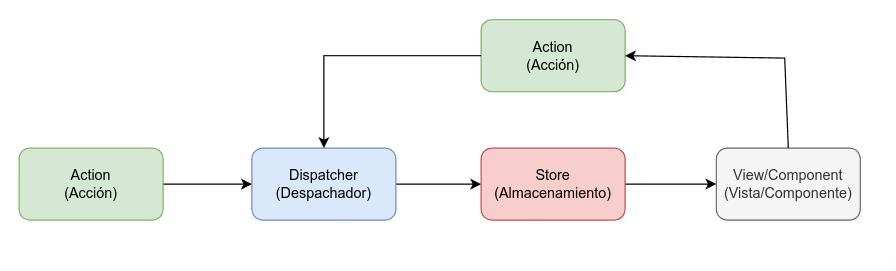
1. Patrón de arquitectura

Para este proyecto utilizaremos el patrón de arquitectura flux, que es creada por la empresa Meta (ex Facebook) para mitigar el cierto grado de complejidad que proviene de utilizar MVC o MVVM en aplicaciones web.

Flux es un patrón de arquitectura que propone que los datos confluyan en un solo sentido y que exista una única fuente de verdad, De este modo todo el flujo de la aplicación llega a un sitio donde se almacena el estado global y que se encarga de actualizar las vistas que están suscritas a los cambios que ocurran.

De esta forma flux se encarga de separar de las vistas, el estado global de la aplicación.

Diagrama de patrón de diseño flux



Las vistas o componentes son las partes que componen las interfaces. La store representa de algún modo una bodega donde se almacenan los datos y que estos datos solo pueden ser modificados a través de dispatchers que son mediadores entre la acción y las vistas. Por último, un Action es código javascript que encapsula una funcionalidad de hacer algo que puede o no, contener datos.

Cuando el usuario realiza alguna acción dentro de la aplicación, desencadena un evento que envía esta acción por medio de un dispatcher hacia la store, esta, actualiza el estado y notifica los cambios a las vistas que correspondan.

1. Estándares de programación segura.

**Perfiles y roles**

Para asegurar una integridad en los datos del sistema, se implementará un sistema de perfiles que contará con el perfil de usuario y otro perfil de administrador.

El perfil de administrador tendrá acceso a todas las funcionalidades administrativas de la aplicación web, con la posibilidad de realizar informes de la aplicación, además contará con la posibilidad de mandar mensajes a usuarios y roles.

Para los roles tendremos: el rol de usuario básico y usuario suscrito.

El usuario básico solo podrá utilizar el 30% de la aplicación y contará con las funcionalidades básicas de tareas, eventos y calendario. El usuario básico podrá convertirse en usuario avanzado si paga la suscripción mensual a la aplicación.

El usuario suscrito obtendrá acceso a todas las funcionalidades extra, además de acceder a la barra lateral personalizable.

**Seguridad**

Para mantener saludable a integridad de la base de datos, se implementarán verificaciones de email valido, contraseña de al menos 6 caracteres, para la creación de nuevas cuentas.

Para todos los inputs de la página se tomarán las medidas correspondientes para evitar ataques de SQL Injection y Cross-Site Scripting tales como:

* Definiendo tipos estáticos a los datos entrantes.
* Filtrar caracteres especiales en las consultas.
* Delimitar tiempos máximos en las sesiones activas.
* Delimitar la cantidad de caracteres en la contraseña a 6.
* Cada mes realizar una copia de seguridad de la base de datos y realizarlo en los periodos de poca actividad en la aplicación, para prevenir errores en los datos.

1. Servicios de backend y Almacenamiento de datos cloud

**Firebase**

Firebase es una plataforma digital desarrollada por Google, que se utiliza para facilitar el desarrollo web y móvil, brindándonos características fundamentales como: Analítica, Desarrollo, Crecimiento y Monetización.

Una de las grandes ventajas de Firebase es que cuenta con una base de datos que puede ser actualizada en tiempo real, altamente configurarle y mantenible. Además, ofrece una intuitiva y eficaz forma de generar usuarios con roles y autorización. Además, Firebase ofrece un servicio de hosting gratuito para desplegar fácilmente nuestra aplicación´.

Este servicio utiliza una base de datos NoSQL llamada Firebase Realtime Database (FRB) alojada en la nube que nos permitirá almacenar y sincronizar datos entre los usuarios activos en tiempo real.

|  |  |
| --- | --- |
| **VENTAJAS** | **DESVENTAJAS** |
| Almacenamiento en la nube. | Limitaciones gratuitas (100 conexiones y 1GB almacenamiento) |
| Escalamiento automático. | Datos alojados en Firebase. |
| API multiplataforma. | Formato JSON, difícil migración. |

**Supabase**

Supabase se describe como una alternativa de código abierto a Firebase. Está diseñada para ayudar a los desarrolladores a poner en marcha su backend en cuestión de minutos. El mantra de Supabase es "Construir en un fin de semana, escalar a millones".

Supabase utiliza base de datos SQL con postgres y ofrece autenticacion, creacion de api REST, funciones, almacenamiento de archivos, y panel de control.

|  |  |
| --- | --- |
| **VENTAJAS** | **DESVENTAJAS** |
| Base de datos Postgres | Postgres al ser SQL puede ser un tanto repetitivo. |
| Almacenamiento de archivos. |  |
| API generado automáticamente. |  |

**Comparativa**

**Similitudes**

Tanto Firebase como Supabase se basan en la idea de aportar una experiencia superior a los desarrolladores de bases de datos. Con ambas plataformas puedes crear un nuevo proyecto directamente desde el navegador sin necesidad de descargar ninguna herramienta o software adicional en tu máquina. Ambas plataformas vienen con una útil interfaz de usuario para la depuración de los datos en tiempo real, lo que es especialmente útil para las iteraciones rápidas en el desarrollo.

**Diferencias**

Firebase y Supabase difieren en varios aspectos. El principal es que Firebase es un almacén de documentos, mientras que Supabase se basa en PostgreSQL, un sistema de gestión de bases de datos relacionales basado en SQL.

**Precios**

Firebase cobra por las lecturas, escrituras y eliminaciones, lo que puede dar lugar a cierta imprevisibilidad, especialmente en las primeras etapas de un proyecto, cuando su aplicación está en pleno desarrollo. Supabase cobra en función de la cantidad de datos almacenados, con espacio para un número ilimitado de solicitudes de API y un número ilimitado de usuarios Auth.

**Rendimiento**

Según los benchmark del repositorio de suparbase el rendimiento de suparbase puede superar 4 veces el de firebase.

Tomando en cuenta la comparativa anterior. Para nuestro proyecto utilizaremos el BaaS open source Supabase.

1. Plan de pruebas

**Desarrollo TDD:**

Para asegurar la calidad del código en el proceso de desarrollo, se implementará el proceso de TTD (Test-Driven Development) que consiste, en pocas palabras, en desarrollar sobre teste previamente implementados y no al revés, es decir, primero se desarrollan los test unitarios y luego se codifica en base a el éxito de esos test.

**Pruebas de caja Blanca:**

Dentro de las pruebas de caja blanca, tendremos todas las pruebas unitarias de los componentes de la web.

Para desarrollar las pruebas unitarias de código utilizaremos el framework de testeo Jestjs, debido a que es ligero, fácil de configurar y ofrece una api de uso amigable.

**Pruebas de caja Negra:**

Para las pruebas de caja negra usaremos la librería de testing Playwright para pruebas end to end, que simulara el uso de un usuario en nuestra página web.

Con esto nos aseguraremos de que cada elemento de la interfaz y del DOM funcionan correctamente, tanto para el administrador, para los usuarios básicos y para los usuarios de pago.

**Plan de Pruebas:**

Para que el plan de pruebas sea aceptado:

* Completar el 100% de pruebas unitarias hechas a componentes, Es necesario que sea el 100% ya que, si un componente no funciona correctamente, La aplicación se rompe
* Completar correctamente una prueba e2e a nivel de usuario básico.
* Completar correctamente una prueba e2e a nivel de usuario suscrito.
* Completar correctamente una prueba e2e a nivel de administrador.
* Completar exitosamente las pruebas de seguridad de SQL Injection
* Completar exitosamente las pruebas de seguridad de Cross-Site Scripting.

Criterios de suspensión

Para que el plan de pruebas sea suspendido:

* Ocurra alguna falla que pueda vulnerar los datos de los clientes.
* Ocurra alguna falla que pueda vulnerar la página web.
* Una falla en la base de datos.

Criterios de reanudación

* Verificación exhaustiva de que todos los errores anteriores sean solucionados.

1. Conclusiones

En este documento definimos el patrón de diseño que utilizaremos para desarrollar la página web ‘organiApp’, El patrón elegido es Flux desarrollado y mantenido por Meta.

Luego vimos los estándares de programación segura, definiendo los perfiles y roles que utilizaran la aplicación. Por otro lado, vimos los componentes más importantes en el ámbito de la seguridad de la web.

Continuamos revisando las distintas opciones de servicios backend, comparamos y definimos cual utilizaremos para el proyecto, quedándonos con supabase.

Finalmente desarrollamos un plan de pruebas para el aplicativo, donde revisamos Test Driven Development, pruebas de caja blanca, pruebas de caja negra y el plan de pruebas y los requisitos para ser aceptados, suspendidos y reanudados.

1. Referencias bibliográficas

Meta. Application architecture for building user interfaces. Recuperado de <https://facebook.github.io/flux/>

Gowtham M, Pramod H B, (2021) "Semantic Query-Featured Ensemble Learning Model for SQL-Injection Attack Detection in IoT-Ecosystems", IEEE Transactions on Reliability,

J. Fonseca, N. Seixas, M. Vieira and H. Madeira, (2013) "Analysis of Field Data on Web Security Vulnerabilities," in IEEE Transactions on Dependable and Secure Computing.

Sánchez J, González J (2015). “Pruebas de Software. Fundamentos y Técnicas”, E.T.S.I. y Sistemas de Telecomunicación (UPM)