



**UNIVERSIDAD NACIONAL AUTÓNOMA DE  
MÉXICO  
FACULTAD DE INGENIERÍA  
DIVISIÓN DE INGENIERÍA ELÉCTRICA**



Asignatura: **Cómputo Móvil**

**Trabajo final**

Semestre: 2024-2

Grupo: 3

Profesor: Ing. Marduk Perez de Lara Dominguez

Nombre de los integrantes:

**Monreal Aldrighetti Adrián  
Garay Jimenez Santiago Daniel  
Fragoso Sorcia Gilberto**

Fecha de entrega: 24/05/2024

## **-Reglas de negocio**

### **-Datos y Actualizaciones**

**Actualización de Datos:** Los datos sobre la disponibilidad y calidad del agua en cada municipio se actualizarán cada 24 horas.

**Fuente de Datos:** Los datos deben ser obtenidos de fuentes oficiales y verificadas, como el Sistema de Aguas de la Ciudad de México (SACMEX) y la Comisión Nacional del Agua (CONAGUA).

**Historial de Datos:** Mantener un historial de los datos de los últimos 12 meses para análisis y comparación.

### **-Acceso y Visualización**

**Acceso a Información:** Todos los usuarios deben tener acceso libre a la información básica sobre la disponibilidad y calidad del agua en su municipio.

**Mapa Interactivo:** Implementar un mapa interactivo donde los usuarios puedan seleccionar su municipio y ver información detallada sobre el estado del agua.

**Alertas:** Los usuarios pueden suscribirse a alertas personalizadas para recibir notificaciones sobre cambios significativos en la disponibilidad o calidad del agua en su municipio.

### **-Funcionalidades**

**Reporte de Problemas:** Permitir a los usuarios reportar problemas relacionados con el suministro de agua (como fugas o baja presión) directamente a través de la aplicación.

**Estadísticas y Gráficas:** Proveer estadísticas y gráficos que muestren tendencias y comparaciones en la disponibilidad y calidad del agua a lo largo del tiempo.

**Comparativa de Municipios:** Permitir la comparación entre diferentes municipios en términos de disponibilidad y calidad del agua.

### **-Usuarios y Privilegios**

**Tipos de Usuarios:** Diferenciar entre usuarios básicos, que pueden ver y reportar información, y usuarios administradores, que pueden gestionar y actualizar los datos.

**Acceso Restringido:** Datos sensibles y de administración solo deben ser accesibles para usuarios con permisos adecuados.

### **-Seguridad y Privacidad**

**Protección de Datos:** Garantizar que toda la información personal de los usuarios esté protegida y cumpla con las leyes de privacidad locales y nacionales.

Cifrado de Datos: Todos los datos transmitidos deben estar cifrados para proteger la integridad y privacidad de la información.

#### -Integración y Extensibilidad

API Pública: Proveer una API pública para que terceros puedan acceder a los datos y crear sus propias aplicaciones o servicios basados en la información de la situación del agua.

Interoperabilidad: La aplicación debe ser capaz de integrarse con otros sistemas de gestión de recursos hídricos y servicios municipales.

#### -Reportes y Análisis

Generación de Reportes: Permitir la generación de reportes detallados sobre la situación del agua que puedan ser descargados en formatos como PDF o Excel.

Análisis Predictivo: Implementar herramientas de análisis predictivo para anticipar problemas futuros basados en patrones históricos y datos actuales.

#### -Comunicación y Educación

Educación Ambiental: Incluir una sección educativa que informe a los usuarios sobre la importancia del ahorro de agua y consejos prácticos para reducir el consumo.

Foro Comunitario: Crear un espacio para que los usuarios puedan discutir y compartir experiencias y soluciones relacionadas con la gestión del agua en sus municipios.

Estas reglas de negocio ayudarán a estructurar la funcionalidad y operación de la aplicación, garantizando que sea útil, accesible y segura para todos los usuarios.

### **Viabilidad de la aplicación**

Nosotros creemos que esta aplicación sí es viable a poder desarrollarse ya que los datos con los que vamos a arrancar la aplicación tienen que ser datos que sean proporcionados por las mismas compañías que se encargan de recaudar estos datos para poder dar estadísticas a todo el país, solo que nosotros lo que haríamos sería que cada persona tenga esta información acerca del lugar donde vive en la palma de su mano para poder prevenirse de una sequía, aunque los datos pueden ser no muy exactos porque es muy complicado tener una certeza al 100% de un tema como lo es la sequía del agua, al menos poder prevenirse en caso de que llegara a pasar, creemos que es bastante viable la funcionalidad de poder acceder a los diferentes municipios de la ciudad y poder encontrar el tuyo para que sepas como se encuentra la situación actualmente y si hay riesgo de que te quedes sin agua y una aproximación de en cuanto tiempo podría pasar esto.

### **Alcance de la Aplicación**

El alcance de la aplicación para monitorear la situación del agua en diferentes municipios de la Ciudad de México debe centrarse en proporcionar información accesible y actualizada

sobre la disponibilidad y calidad del agua, además de incluir funcionalidades básicas para reportar problemas y recibir alertas.

El proyecto debe priorizar la usabilidad y la exactitud de los datos, con una interfaz sencilla y clara para los usuarios.

-Funcionalidades del Alcance

Mapa Interactivo: Visualización de la situación del agua en cada municipio.

Actualización de Datos: Información diaria sobre la disponibilidad y calidad del agua.

Alertas Personalizadas: Notificaciones sobre cambios significativos en el suministro de agua.

Reporte de Problemas: Funcionalidad para que los usuarios reporten problemas como fugas o baja presión.

Estadísticas Básicas: Gráficos y estadísticas simples para mostrar tendencias a lo largo del tiempo.

Acceso a Información Básica: Información libremente accesible sobre la situación del agua en cada municipio.

### **Producto Mínimo Viable (MVP)**

El MVP debe enfocarse en las funcionalidades esenciales que ofrecen valor inmediato a los usuarios y permiten una rápida implementación y evaluación del mercado. El objetivo es lanzar una versión básica pero funcional de la aplicación que pueda ser iterada y mejorada en función del feedback de los usuarios.

Funcionalidades del MVP

-Mapa Interactivo con Información Básica:

Visualización de los municipios de la Ciudad de México.

Indicadores de disponibilidad y calidad del agua por municipio (colores o íconos simples).

-Actualización Diaria de Datos:

Integración con fuentes de datos oficiales (SACMEX, CONAGUA) para actualizar la información diariamente.

-Alertas por Email o Notificaciones Push:

Sistema de alertas que notifique a los usuarios sobre cambios importantes en la disponibilidad o calidad del agua.

#### -Función de Reporte de Problemas:

Formulario sencillo para que los usuarios reporten problemas relacionados con el suministro de agua.

Envío de reportes a las autoridades competentes o a una base de datos interna.

#### -Panel de Estadísticas Básicas:

Gráficos simples que muestren la tendencia de la disponibilidad y calidad del agua en los últimos días o semanas.

#### -Interfaz de Usuario Intuitiva:

Diseño sencillo y accesible para facilitar el uso por parte de cualquier ciudadano.

#### -Tecnologías y Herramientas

Frontend: React Native (para desarrollo de aplicaciones móviles multiplataforma)

Backend: Node.js con Express (para la lógica del servidor y manejo de API)

Base de Datos: MongoDB (para almacenar datos de usuarios, reportes y estadísticas)

Notificaciones Push: Firebase Cloud Messaging

Mapa Interactivo: Mapbox o Google Maps API

#### -Ejemplo de Flujo de Usuario en el MVP

Inicio de Sesión/Opción de Continuar como Invitado:

El usuario puede registrarse o usar la app como invitado.

Visualización del Mapa:

El usuario ve un mapa interactivo de la Ciudad de México con indicadores de agua en cada municipio.

Selección de Municipio:

Al tocar un municipio, el usuario ve información detallada sobre la disponibilidad y calidad del agua.

Suscripción a Alertas:

El usuario puede suscribirse a recibir alertas para su municipio.

## Reporte de Problemas:

El usuario puede llenar un formulario para reportar problemas de agua, que se envía automáticamente a las autoridades.

## Consulta de Estadísticas:

El usuario puede ver gráficos básicos sobre la situación del agua en su municipio.

## Objetivos a Largo Plazo

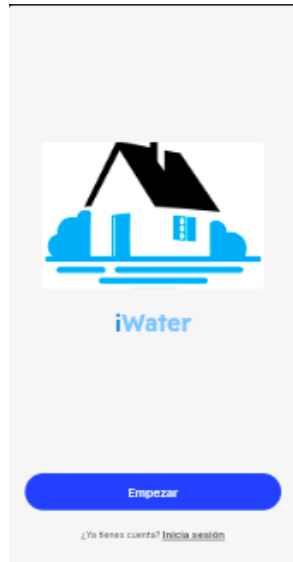
Integración con más fuentes de datos para mejorar la precisión de la información.

Implementación de análisis predictivo para anticipar problemas futuros.

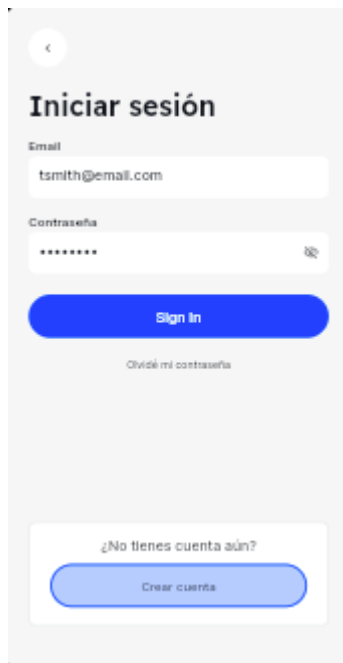
Ampliación de la funcionalidad de reporte para incluir seguimiento del estado de los reportes. Educación y concienciación sobre la gestión del agua mediante contenidos adicionales.

## Nombre final de la app y diseño

Finalmente nos decidimos por llamar la app iWater, ya que sentimos que es un nombre conciso y corto para describir lo que es, pensabamos si agregarle algo más al nombre pero finalmente nos decidimos por dejar este, ya que también nos pareció que un estilo minimalista tanto del nombre como del logo haría ver a la aplicación más limpia y más cómoda para los usuarios que la vayan a usar.

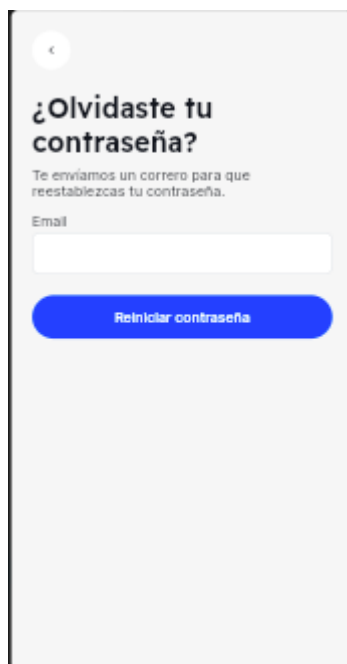


Como primera imagen tenemos el logo de la aplicación, así como el botón de empezar para dar inicio a la aplicación o, en su defecto, en la parte inferior del botón tenemos un mensaje que pregunta al usuario si ya tiene una cuenta, que pueda iniciar sesión.



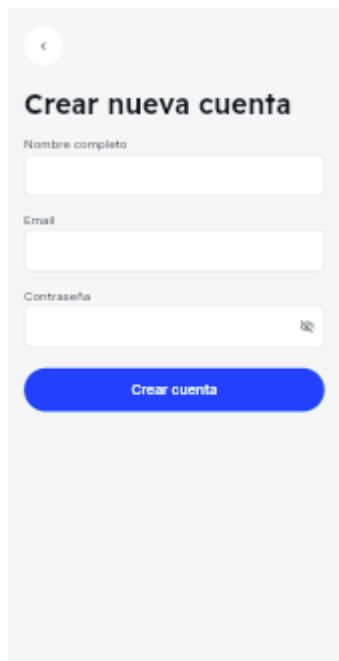
The image shows a mobile application login screen. At the top left is a back arrow icon. The title "Iniciar sesión" is centered. Below it are two input fields: "Email" with the text "tsmith@email.com" and "Contraseña" with masked characters "\*\*\*\*\*". To the right of the password field is an eye icon. A blue "Sign in" button is centered below the fields. Underneath the button is the text "Olvidé mi contraseña". At the bottom, there is a white box containing the text "¿No tienes cuenta aún?" and a blue "Crear cuenta" button.

Si le das al botón de iniciar sesión, se mostrará esta pantalla, en la cual puedes introducir tu correo electrónico y tu contraseña para poder iniciar sesión en la aplicación. Una vez introduces tu correo y contraseña, abajo del campo de contraseña se encuentra el botón “Sign in”, para dar inicio a la app, debajo de este botón se encuentra un pequeño mensaje que te pregunta si olvidaste tu contraseña.



The image shows a mobile application screen for forgetting a password. At the top left is a back arrow icon. The title "¿Olvidaste tu contraseña?" is centered. Below it is a subtitle: "Te enviamos un correo para que reestablezcas tu contraseña." Below this is an "Email" input field. At the bottom is a blue button labeled "Reiniciar contraseña".

Si le das al botón de de olvidaste tu contraseña, se abrirá esta página donde te pide que introduces el correo con el cual te registraste para mandarte instrucciones por correo para que puedas restablecer tu contraseña.



**Crear nueva cuenta**

Nombre completo

Email

Contraseña

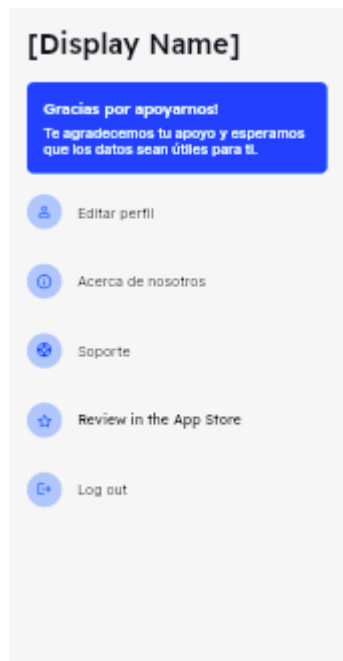
Crear cuenta

Si no tienes cuenta y no le diste a ninguna de las opciones anteriores para iniciar sesión, sigue la pantalla de crear una cuenta en la que introduces tu nombre completo, correo y contraseña, posteriormente a introducir estos datos, le das a crear cuenta.

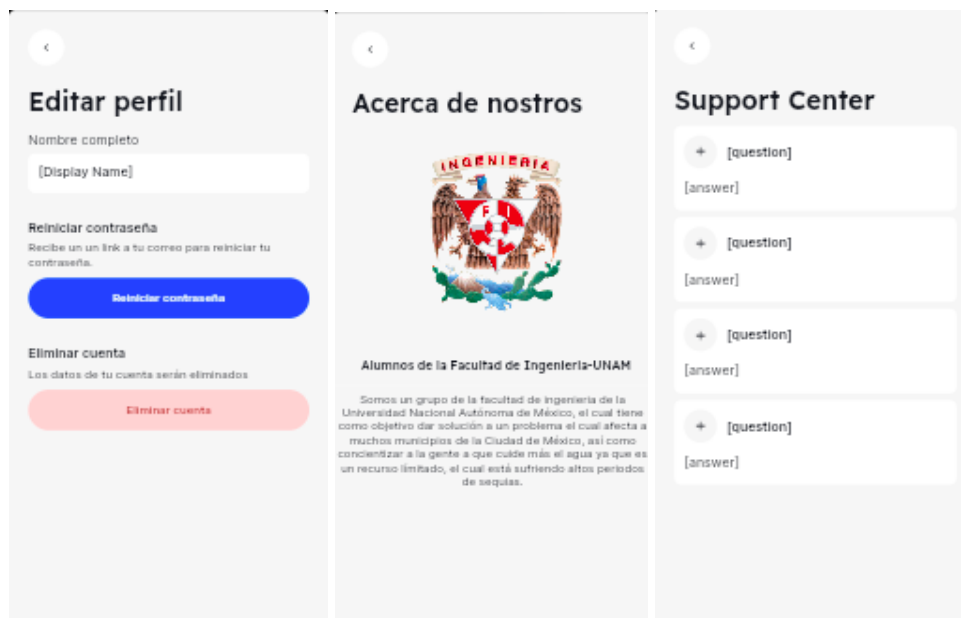


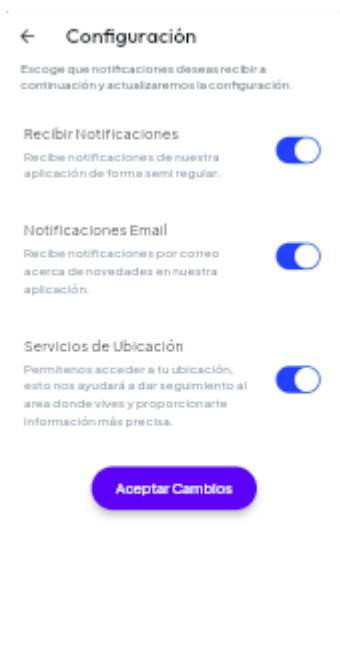
Una vez que hayas iniciado sesión o que hayas creado tu cuenta, te saldrá esta pantalla con un slideshow en el cual brindaremos información acerca del cuidado del agua, esta información podemos e iremos cambiando cada semana para que la gente tenga conciencia acerca del cuidado del agua y cómo puede ayudar a su comunidad a que no se desperdicie.



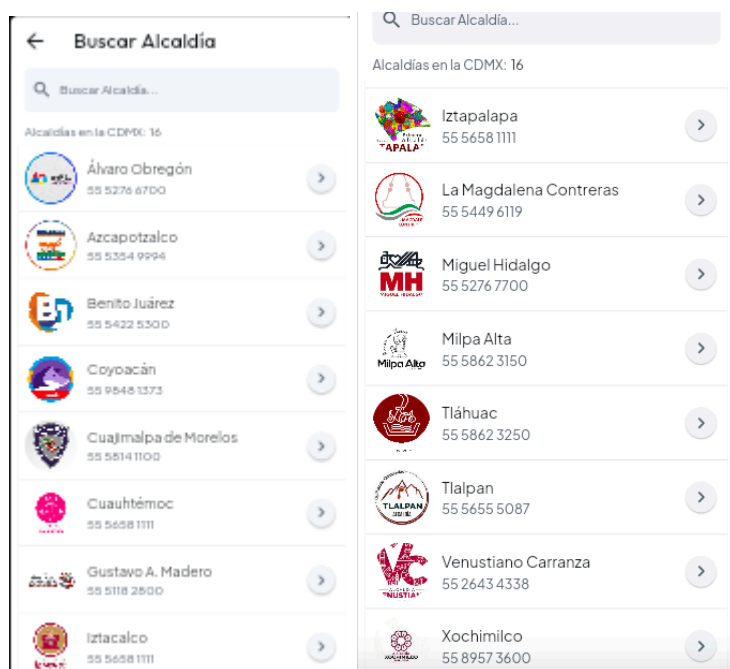


Si ingresas a tu perfil puedes acceder a estas opciones, ya sea para editar tu perfil, para saber acerca nosotros que estamos diseñando la aplicación, puedes acceder al soporte para reportar algún problema o para dar alguna sugerencia, que se tomará en cuenta para mejorar el rendimiento de la aplicación, también vemos que hay un botón para darle rating a la aplicación en la app store y finalmente un botón para salir de la cuenta.

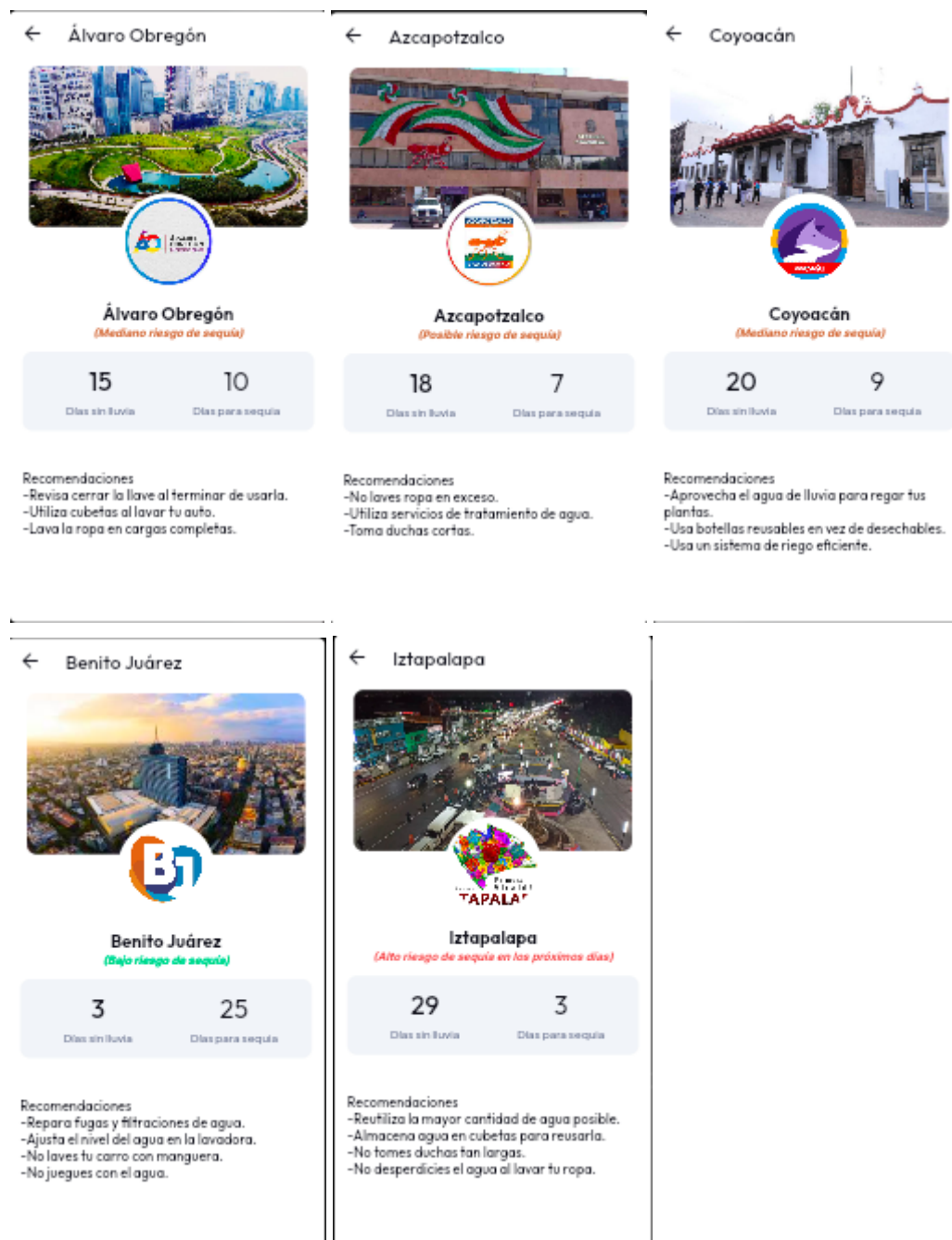




En la ventana de configuración puedes escoger si recibir notificaciones, recibir notificaciones a tu correo y la última opción es la más importante, ya que con los servicios de ubicación podemos dar datos más exactos acerca de dónde te encuentras y la información que tendrás como usuario será más precisa.



Después tenemos la pantalla en la cual se muestran todos los municipios de la CDMX, esta pantalla es scrollable para que puedas buscar tu municipio para obtener la información acerca de la sequía del agua, así como un número de contacto en caso de que quieras hacer algún llamado.



Finalmente tenemos las pantallas de las alcaldías, donde podemos ver el nombre de la alcaldía, una foto del lugar, el ícono de la alcaldía, y en la parte de abajo del nombre podemos ver un mensaje que muestra la situación actual en cuanto a sequía del agua, aquí podemos identificar si la alcaldía tiene problemas en cuanto al agua. En la parte de abajo de la pantalla se pueden encontrar recomendaciones basadas en la situación actual de la sequía de la alcaldía, a pesar de la situación de la sequía, es recomendable siempre cuidar el agua y no desperdiciarla.

**- De las funcionalidades viables detallar qué servicios, frameworks o kits de desarrollo se usarían para implementar y justificar por qué.**

Para el diseño de una aplicación enfocada en el cuidado del agua en la Ciudad de México, con funcionalidades que incluyan el monitoreo de consumo de agua y alertas de riesgo de escasez, es crucial seleccionar los servicios, frameworks y kits de desarrollo adecuados. A continuación, detallo las funcionalidades viables y las herramientas recomendadas para implementarlas, junto con sus justificaciones:

### **Funcionalidades Viables y Herramientas Recomendadas**

#### **1. Base de Datos de Consumo de Agua**

Servicio/Framework: PostgreSQL

PostgreSQL es una base de datos relacional muy robusta y escalable, adecuada para manejar grandes volúmenes de datos. Soporta consultas complejas y tiene una excelente integración con diversas herramientas de análisis de datos.

#### **2. Sistema de Alertas de Riesgo de Escasez**

Servicio/Framework: Firebase Cloud Messaging (FCM)

El FCM permite enviar notificaciones push a los usuarios en tiempo real. Es altamente eficiente para alertar a los usuarios sobre riesgos de escasez de agua de manera inmediata.

#### **3. Interfaz de Usuario (UI)**

Framework: React Native

React Native permite desarrollar aplicaciones móviles para iOS y Android usando una sola base de código. Es eficiente, tiene un rendimiento cercano al nativo y una gran comunidad de soporte.

#### **4. Monitoreo de Consumo de Agua en Tiempo Real**

Servicio/Framework: MQTT (Message Queuing Telemetry Transport)

MQTT es un protocolo de mensajería ligero y eficiente, ideal para la comunicación de datos en tiempo real. Es ampliamente utilizado en aplicaciones de IoT y permite un intercambio de información rápido y fiable entre dispositivos.

#### **5. Visualización de Datos y Análisis**

Framework: D3.js

D3.js es una biblioteca de JavaScript para producir visualizaciones de datos dinámicas e interactivas en el navegador. Es perfecta para mostrar gráficos, mapas de calor y otros elementos visuales que ayudan a los usuarios a comprender su consumo de agua.

#### **6.Backend y API**

Framework: Node.js con Express

Node.js con Express es una combinación potente para construir APIs RESTful. Es rápida, escalable y tiene una gran cantidad de módulos disponibles que pueden facilitar el desarrollo.

#### **7.Autenticación de Usuarios**

Servicio: Auth0

Auth0 proporciona servicios de autenticación y autorización seguros, fáciles de integrar y con soporte para múltiples métodos de inicio de sesión (social, email, etc.). Facilita la gestión de usuarios sin comprometer la seguridad.

## 8. Alojamiento y Despliegue

AWS (Amazon Web Services) o Google Cloud Platform (GCP)

Ambas plataformas ofrecen servicios escalables, seguros y de alta disponibilidad para alojar aplicaciones web y móviles. Además, cuentan con una variedad de servicios adicionales que pueden complementar el desarrollo de la aplicación, como bases de datos gestionadas, servicios de almacenamiento y herramientas de análisis.

## ¿Cómo lo implementamos?

- Base de Datos: PostgreSQL para almacenar datos de consumo de agua y riesgo de escasez.
- Backend/API: Node.js con Express para manejar la lógica del servidor y las solicitudes API.
- Notificaciones: Firebase Cloud Messaging para enviar alertas a los usuarios.
- Frontend: React Native para el desarrollo de la aplicación móvil.
- Visualización de Datos: D3.js para gráficos y visualizaciones interactivas.
- Autenticación: Auth0 para gestionar el acceso de los usuarios.
- Alojamiento: AWS o GCP para desplegar tanto el frontend como el backend.

**- Si necesitan un servicio back end propio detallar sus características y funcionalidades, si es un API, los endpoints, las características de autenticación, la tecnología con la que se desarrollaría, etc.**

Si se necesita un servicio backend propio para la aplicación de cuidado del agua, este debe ser robusto, escalable y seguro. A continuación, se detallan las características y funcionalidades de dicho backend, incluyendo la estructura del API, los endpoints, las características de autenticación y la tecnología utilizada para su desarrollo.

## Características y Funcionalidades del Backend

### 1. Gestión de Usuarios

- Registro y autenticación de usuarios.
- Gestión de perfiles de usuario (datos personales, preferencias, etc.).

### 2. Monitoreo y Registro de Consumo de Agua

- Recepción y almacenamiento de datos de consumo de agua.
- Análisis de patrones de consumo.

### 3. Sistema de Alertas

- Envío de notificaciones sobre el riesgo de escasez de agua.
- Configuración de alertas personalizadas por los usuarios.

### 4. Visualización de Datos

- Provisión de datos para gráficos y visualizaciones en la aplicación.

## 5. Interacción en Tiempo Real

- Comunicación en tiempo real para actualizaciones instantáneas.

## Endpoints del API

### Autenticación

#### 1. POST /api/auth/register

- Descripción: Registrar un nuevo usuario.
- Parámetros: `email`, `password`, `name`.
- Respuesta: `201 Created` con el objeto de usuario o un mensaje de error.

#### 2. POST /api/auth/login

- Descripción: Iniciar sesión para un usuario existente.
- Parámetros: `email`, `password`.
- Respuesta: `200 OK` con token JWT o mensaje de error.

#### 3. POST /api/auth/logout

- Descripción: Cerrar sesión.
- Parámetros: Token JWT.
- Respuesta: `200 OK` con mensaje de éxito.

## Usuarios

#### 1. GET /api/users/profile

- Descripción: Obtener el perfil del usuario autenticado.
- Parámetros: Token JWT.
- Respuesta: `200 OK` con el objeto de perfil de usuario.

#### 2. PUT /api/users/profile

- Descripción: Actualizar el perfil del usuario.
- Parámetros: Token JWT, datos del perfil.
- Respuesta: `200 OK` con el perfil actualizado o mensaje de error.

## Consumo de Agua

#### 1. POST /api/water-consumption

- Descripción: Registrar datos de consumo de agua.
- Parámetros: Token JWT, datos de consumo (`user\_id`, `amount`, `timestamp`).
- Respuesta: `201 Created` con el registro de consumo o mensaje de error.

#### 2. GET /api/water-consumption

- Descripción: Obtener datos de consumo de agua para el usuario autenticado.
- Parámetros: Token JWT, filtros opcionales (`start\_date`, `end\_date`).
- Respuesta: `200 OK` con los datos de consumo.

## Alertas

#### 1. POST /api/alerts

- Descripción: Crear una alerta personalizada.
- Parámetros: Token JWT, datos de la alerta (`type`, `threshold`, `frequency`).

- Respuesta: `201 Created` con la alerta creada o mensaje de error.

## 2. GET /api/alerts

- Descripción: Obtener las alertas del usuario autenticado.
- Parámetros: Token JWT.
- Respuesta: `200 OK` con las alertas configuradas.

## Características de Autenticación

- JWT (JSON Web Tokens): Se utilizarán tokens JWT para autenticar las solicitudes. Estos tokens se generarán al iniciar sesión y se enviarán en el encabezado de autorización de las solicitudes posteriores.
- Bcrypt: Para el almacenamiento seguro de contraseñas.
- Roles y Permisos: Se pueden implementar roles de usuario (por ejemplo, usuario normal, administrador) para manejar diferentes niveles de acceso y permisos.

## Tecnología Utilizada

### 1. Node.js con Express

- Justificación: Node.js es altamente eficiente y adecuado para manejar múltiples conexiones concurrentes, lo cual es ideal para una aplicación que requiere actualización en tiempo real. Express facilita la creación de APIs RESTful con una estructura modular y clara.

### 2. PostgreSQL

- Justificación: PostgreSQL es una base de datos relacional robusta y escalable, capaz de manejar consultas complejas y grandes volúmenes de datos.

### 3. Socket.io

- Justificación: Para la funcionalidad en tiempo real, como la actualización instantánea de datos de consumo y alertas, Socket.io es una excelente opción para la comunicación bidireccional en tiempo real.

### 4. Redis

- Justificación: Redis puede ser utilizado para almacenamiento en caché y gestión de sesiones, mejorando el rendimiento de la aplicación y la experiencia del usuario.

### 5. Docker

- Justificación: Para la implementación y gestión de contenedores, Docker facilita el desarrollo y despliegue de la aplicación en diferentes entornos de manera consistente.

## Ejemplo de Arquitectura

- Frontend (React Native)
  - Se comunica con el backend a través de los endpoints de la API.
  - Maneja la interfaz de usuario y muestra datos en tiempo real utilizando Socket.io.
- Backend (Node.js con Express)

- Implementa los endpoints RESTful para gestionar usuarios, consumo de agua, y alertas.
- Utiliza JWT para la autenticación.
- Se comunica con PostgreSQL para operaciones de CRUD.
- Base de Datos (PostgreSQL)
  - Almacena datos de usuarios, consumo de agua, y configuraciones de alertas.
- Real-time Updates (Socket.io)
  - Proporciona actualizaciones en tiempo real sobre consumo de agua y alertas.
- Cache and Session Management (Redis)
  - Mejora el rendimiento mediante el almacenamiento en caché de datos frecuentes.

Esta estructura asegura que la aplicación sea eficiente, segura y capaz de manejar grandes volúmenes de datos y múltiples usuarios simultáneamente.

**- Detallar el flujo de datos que se espera tanto del usuario como de los servicios y cómo se comunicarán estos datos, la forma en la que persistirán, que pasa si no hay conexión o servicio de datos y cómo se dará seguridad a los mismos.**

### **Flujo de Datos y Comunicación entre Usuario y Servicios**

#### **1. Registro e Inicio de Sesión**

- Usuario: El usuario se registra proporcionando su email y contraseña. Luego, inicia sesión con estos mismos datos.
- Backend: Los datos se envían al endpoint `/api/auth/register` o `/api/auth/login` respectivamente.
- Persistencia: Los datos del usuario se almacenan en PostgreSQL.
- Seguridad: Las contraseñas se hashsean usando Bcrypt antes de almacenarse. Las respuestas incluyen un JWT que se usa para autenticar futuras solicitudes.

#### **2. Monitoreo y Registro de Consumo de Agua**

- Usuario: El usuario ingresa o envía automáticamente datos de consumo de agua a través de dispositivos IoT.
- Backend: Los datos se envían al endpoint `/api/water-consumption` con el JWT en el encabezado de autorización.
- Persistencia: Los datos de consumo se almacenan en PostgreSQL.
- Seguridad: Solo los usuarios autenticados pueden enviar datos. Las conexiones usan HTTPS para proteger la transmisión de datos.

#### **3. Visualización de Datos**

- Usuario: El usuario solicita datos históricos de consumo de agua.
- Backend: La solicitud se envía al endpoint `/api/water-consumption` con posibles parámetros de filtro (`start_date`, `end_date`).
- Persistencia: Los datos se recuperan de PostgreSQL y se envían al usuario.
- Seguridad: El JWT se verifica antes de enviar los datos. Las conexiones usan HTTPS.

#### **4. Sistema de Alertas**



- Usuario: El usuario configura alertas personalizadas sobre el consumo de agua.
- Backend: Los datos de alerta se envían al endpoint `/api/alerts`.
- Persistencia: Las alertas se almacenan en PostgreSQL.
- Seguridad: Solo los usuarios autenticados pueden crear y modificar alertas. Las conexiones usan HTTPS.

## 5. Actualización en Tiempo Real

- Usuario: El usuario recibe actualizaciones instantáneas sobre su consumo de agua y alertas.
- Backend: Las actualizaciones se envían a través de Socket.io.
- Persistencia: Las actualizaciones se basan en datos en tiempo real procesados en el backend.
- Seguridad: La conexión de Socket.io está asegurada mediante HTTPS y autenticada usando JWT.

## Persistencia de Datos y Manejo de Conexiones

### Persistencia de Datos

- Base de Datos Principal: PostgreSQL se utiliza para almacenar datos persistentes como información de usuarios, registros de consumo de agua y configuraciones de alertas.
- Caché: Redis se usa para almacenar datos temporales y mejorar el rendimiento, como sesiones de usuario y resultados de consultas frecuentes.

### Manejo de Conexiones

#### 1. Si hay Conexión

- Operación Normal: Los datos se envían al backend, se procesan y almacenan en PostgreSQL. Las respuestas se devuelven en tiempo real.

#### 2. Si no hay Conexión

- Datos de Consumo: Los datos de consumo pueden almacenarse localmente en el dispositivo del usuario utilizando almacenamiento local (por ejemplo, AsyncStorage en React Native).
- Sincronización Posterior: Cuando la conexión se restablece, los datos almacenados localmente se sincronizan con el backend mediante un servicio de sincronización en segundo plano.

## Seguridad de los Datos

### Transmisión de Datos

- HTTPS: Todas las comunicaciones entre el cliente y el servidor se realizan a través de HTTPS para asegurar la transmisión de datos.

### Autenticación y Autorización

- JWT: Los tokens JWT se utilizan para autenticar las solicitudes. Los tokens se generan al iniciar sesión y se envían en el encabezado de autorización de las solicitudes.
- Roles y Permisos: Se implementan roles de usuario para manejar diferentes niveles de acceso a los recursos.

### Almacenamiento de Datos

- Contraseñas: Las contraseñas de usuario se almacenan hashseadas usando Bcrypt.
- Datos Sensibles: Los datos sensibles se cifran antes de almacenarse en la base de datos.

### Protección contra Ataques

- CSRF y XSS: Se implementan medidas de protección contra ataques CSRF (Cross-Site Request Forgery) y XSS (Cross-Site Scripting).
- Rate Limiting: Se implementa rate limiting para prevenir ataques de fuerza bruta y denegación de servicio.

## Ejemplo de Flujo de Datos en un Escenario

### 1. Registro de Usuario

- El usuario envía una solicitud POST a `/api/auth/register` con su email y contraseña.
  - El backend valida la solicitud, hashsea la contraseña y almacena los datos en PostgreSQL.
- El backend responde con un token JWT.

### 2. Envío de Datos de Consumo

- El usuario envía una solicitud POST a `/api/water-consumption` con datos de consumo y el JWT.
- El backend valida el JWT, procesa los datos y los almacena en PostgreSQL.
- El backend responde con una confirmación de éxito.

### 3. Recepción de Alertas

- El backend analiza los datos de consumo y, si se cumple una condición de alerta, envía una notificación push a través de Firebase Cloud Messaging.
- El usuario recibe la alerta en tiempo real en su dispositivo móvil.

### 4. Visualización de Datos Históricos

- El usuario envía una solicitud GET a `/api/water-consumption` con el JWT y posibles filtros de fecha.
- El backend valida el JWT, recupera los datos de PostgreSQL y los envía al usuario.
- El usuario ve los datos históricos en la interfaz de la aplicación.

Este flujo de datos asegura que la aplicación sea eficiente, segura y capaz de manejar situaciones de conectividad variable, garantizando la integridad y seguridad de los datos del usuario en todo momento.

**- Si es necesario para su app, detallar que tipo de permisos o adecuaciones tienen que considerar para poder publicar en tienda de apps o liberar su aplicación al mercado.**

Para publicar una aplicación de cuidado del agua en las tiendas de aplicaciones (Google Play Store y Apple App Store), es crucial cumplir con una serie de requisitos, permisos y adecuaciones específicos. A continuación, se detallan los aspectos clave que se deben considerar:

## Permisos Necesarios

### 1. Acceso a Internet

- Descripción: Permite a la aplicación conectarse a la red para enviar y recibir datos.
- Android: ``android.permission.INTERNET``
- iOS: No requiere permiso explícito, pero se debe especificar en las capacidades de la aplicación.

### 2. Almacenamiento Local

- Descripción: Permite a la aplicación leer y escribir datos en el almacenamiento del dispositivo.
- Android: ``android.permission.WRITE_EXTERNAL_STORAGE`` y ``android.permission.READ_EXTERNAL_STORAGE``
- iOS: No requiere permiso explícito, pero debe utilizarse el almacenamiento local adecuado (por ejemplo, Core Data o UserDefaults).

### 3. Notificaciones Push

- Descripción: Permite a la aplicación enviar notificaciones al dispositivo del usuario.
- Android: Utiliza Firebase Cloud Messaging (FCM).
- iOS: Utiliza Apple Push Notification Service (APNs). Se requiere permiso del usuario.

### 4. Ubicación (Opcional)

- Descripción: Permite a la aplicación acceder a la ubicación del dispositivo para proporcionar servicios basados en ubicación (por ejemplo, alertas específicas de áreas con escasez de agua).
- Android: ``android.permission.ACCESS_FINE_LOCATION`` y ``android.permission.ACCESS_COARSE_LOCATION``
- iOS: ``NSLocationWhenInUseUsageDescription`` en el archivo ``Info.plist``.

## Adecuaciones Necesarias

### 1. Política de Privacidad

- Descripción: Una política de privacidad clara y accesible que explique cómo se recopilan, utilizan, y protegen los datos del usuario.
- Requisito: Debe estar disponible desde la tienda de aplicaciones y dentro de la aplicación misma.

### 2. Cumplimiento de GDPR y CCPA

- Descripción: Asegurar el cumplimiento de regulaciones de privacidad como el Reglamento General de Protección de Datos (GDPR) en la UE y la Ley de Privacidad del Consumidor de California (CCPA).
- Requisito: Informar a los usuarios sobre sus derechos y proporcionar mecanismos para que puedan ejercerlos (por ejemplo, solicitar la eliminación de sus datos).

### 3. Revisión de Seguridad

- Descripción: Realizar auditorías de seguridad y pruebas de penetración para identificar y mitigar posibles vulnerabilidades.
- Requisito: Implementar medidas de seguridad adecuadas para proteger los datos del usuario.

#### 4. Optimización de Desempeño

- Descripción: Optimizar la aplicación para garantizar un rendimiento fluido y eficiente en una variedad de dispositivos.
- Requisito: Minimizar el consumo de batería y recursos del dispositivo.

### **Publicación en Google Play Store**

#### 1. Cuenta de Desarrollador

- Requisito: Crear y configurar una cuenta de desarrollador en Google Play Console.

#### 2. Cumplimiento de Políticas

- Descripción: Asegurarse de que la aplicación cumpla con las políticas de Google Play, incluyendo contenido, privacidad, y seguridad.
- Requisito: Revisar y seguir las directrices de Google Play Developer Policy Center.

#### 3. App Bundle o APK

- Descripción: Compilar y firmar la aplicación en formato Android App Bundle (AAB) o APK.
- Requisito: Subir el archivo a Google Play Console junto con los detalles de la aplicación (descripción, capturas de pantalla, etc.).

#### 4. Revisión y Aprobación

- Descripción: Google revisará la aplicación antes de su publicación.
- Requisito: Cumplir con los estándares de revisión de Google.

### **Publicación en Apple App Store**

#### 1. Cuenta de Desarrollador

- Requisito: Crear y configurar una cuenta de desarrollador en el Apple Developer Program.

#### 2. Cumplimiento de Directrices

- Descripción: Asegurarse de que la aplicación cumpla con las directrices de revisión de la App Store.
- Requisito: Revisar y seguir las directrices de App Store Review Guidelines.

#### 3. Certificados y Perfiles

- Descripción: Configurar certificados de desarrollo y distribución, así como perfiles de aprovisionamiento.
- Requisito: Utilizar Xcode para gestionar los certificados y perfiles.

#### 4. Compilación y Firma

- Descripción: Compilar y firmar la aplicación utilizando Xcode.
- Requisito: Crear un archivo IPA (iOS App) y subirlo a App Store Connect.

#### 5. Revisión y Aprobación

- Descripción: Apple revisará la aplicación antes de su publicación.
- Requisito: Cumplir con los estándares de revisión de Apple.

### **Consideraciones Finales**

- Documentación: Incluir documentación clara y detallada para los usuarios sobre cómo usar la aplicación y sus características.
- Soporte y Mantenimiento: Planificar la provisión de soporte técnico y actualizaciones regulares para la aplicación.
- Pruebas Beta: Considerar el lanzamiento de una versión beta para recolectar feedback y realizar ajustes antes del lanzamiento final.

Cumplir con estos requisitos y adecuaciones no solo facilitará la publicación de la aplicación en las tiendas de aplicaciones, sino que también garantizará una experiencia segura y satisfactoria para los usuarios.

### **- Inventario de qué lenguajes de programación y servicios utilizarán**

Para desarrollar y desplegar la aplicación de cuidado del agua en la Ciudad de México, se utilizarán varios lenguajes de programación y servicios. A continuación, se presenta un inventario detallado de las tecnologías y servicios que se emplearán en el proyecto:

#### **Lenguajes de Programación**

##### **1. JavaScript/TypeScript**

- Uso: Desarrollo del frontend (React Native) y backend (Node.js con Express).
- Ventajas: Flexibilidad, gran ecosistema de librerías y frameworks, y facilidad para desarrollar tanto el frontend como el backend con el mismo lenguaje.

##### **2. SQL**

- Uso: Consultas y manipulación de datos en PostgreSQL.
- Ventajas: Lenguaje estándar para bases de datos relacionales, optimizado para realizar consultas complejas.

##### **3. Python (Opcional)**

- Uso: Análisis de datos, scripts de backend o tareas de administración.
- Ventajas: Facilidad de uso, vasta cantidad de bibliotecas y soporte para tareas de análisis y manipulación de datos.

#### **Servicios y Frameworks**

##### **1. Frontend**

- React Native
  - Uso: Desarrollo de la aplicación móvil para iOS y Android.
  - Ventajas: Desarrollo multiplataforma con una única base de código, rendimiento cercano al nativo, y una gran comunidad de soporte.

##### **2. Backend**

- Node.js con Express
  - Uso: Implementación del servidor y los endpoints de la API RESTful.
  - Ventajas: Rendimiento eficiente para aplicaciones en tiempo real, estructura modular, y gran ecosistema de paquetes npm.

##### **3. Base de Datos**

- PostgreSQL
  - Uso: Almacenamiento y gestión de datos de usuarios, consumo de agua, y configuraciones de alertas.
  - Ventajas: Robusta, escalable, soporte para consultas SQL complejas y transacciones.
- 4. Comunicación en Tiempo Real
  - Socket.io
    - Uso: Actualizaciones en tiempo real de datos de consumo y alertas.
    - Ventajas: Comunicación bidireccional eficiente y fácil de integrar con Node.js.
- 5. Notificaciones Push
  - Firebase Cloud Messaging (FCM)
    - Uso: Envío de notificaciones push a dispositivos Android y iOS.
    - Ventajas: Servicio gratuito y fiable para notificaciones push, fácil de integrar con las plataformas móviles.
- 6. Autenticación y Autorización
  - Auth0
    - Uso: Gestión de autenticación de usuarios y autorización.
    - Ventajas: Seguridad robusta, fácil de integrar, y soporte para múltiples métodos de autenticación (social, email, etc.).
- 7. Almacenamiento en Caché y Sesiones
  - Redis
    - Uso: Almacenamiento en caché de datos y gestión de sesiones.
    - Ventajas: Velocidad de acceso a datos, soporte para estructuras de datos avanzadas, y mejora del rendimiento.
- 8. Visualización de Datos
  - D3.js
    - Uso: Creación de gráficos y visualizaciones interactivas en la aplicación.
    - Ventajas: Flexibilidad para crear visualizaciones personalizadas y dinámicas.

## **Servicios de Alojamiento y Despliegue**

1. AWS (Amazon Web Services)
  - Uso: Despliegue y alojamiento de la aplicación.
    - Servicios: EC2 para servidores, RDS para la base de datos PostgreSQL, S3 para almacenamiento, y CloudFront para distribución de contenido.
  - Ventajas: Alta disponibilidad, escalabilidad, y amplia gama de servicios complementarios.
2. Google Cloud Platform (GCP)
  - Uso: Alternativa para despliegue y alojamiento.
    - Servicios: Compute Engine para servidores, Cloud SQL para PostgreSQL, Cloud Storage para almacenamiento, y Firebase para notificaciones.
  - Ventajas: Integración con otros servicios de Google, escalabilidad, y confiabilidad.

## **Consideraciones Adicionales**

- Git y GitHub/GitLab: Para control de versiones y colaboración en el desarrollo.
- Docker: Para contenedorización y despliegue de la aplicación en distintos entornos.
- CI/CD (Integración Continua/Despliegue Continuo): Jenkins, GitHub Actions, o GitLab CI para automatizar pruebas y despliegue.

## Resumen

- Frontend: React Native (JavaScript/TypeScript)
- Backend: Node.js con Express (JavaScript/TypeScript)
- Base de Datos: PostgreSQL (SQL)
- Comunicación en Tiempo Real: Socket.io (JavaScript)
- Notificaciones Push: Firebase Cloud Messaging (FCM)
- Autenticación: Auth0
- Almacenamiento en Caché: Redis
- Visualización de Datos: D3.js (JavaScript)
- Alojamiento: AWS o GCP
- Control de Versiones: Git y GitHub/GitLab
- Contenedorización: Docker
- CI/CD: Jenkins, GitHub Actions, o GitLab CI

Esta combinación de tecnologías y servicios asegura un desarrollo ágil, un despliegue eficiente, y una operación segura y escalable de la aplicación.

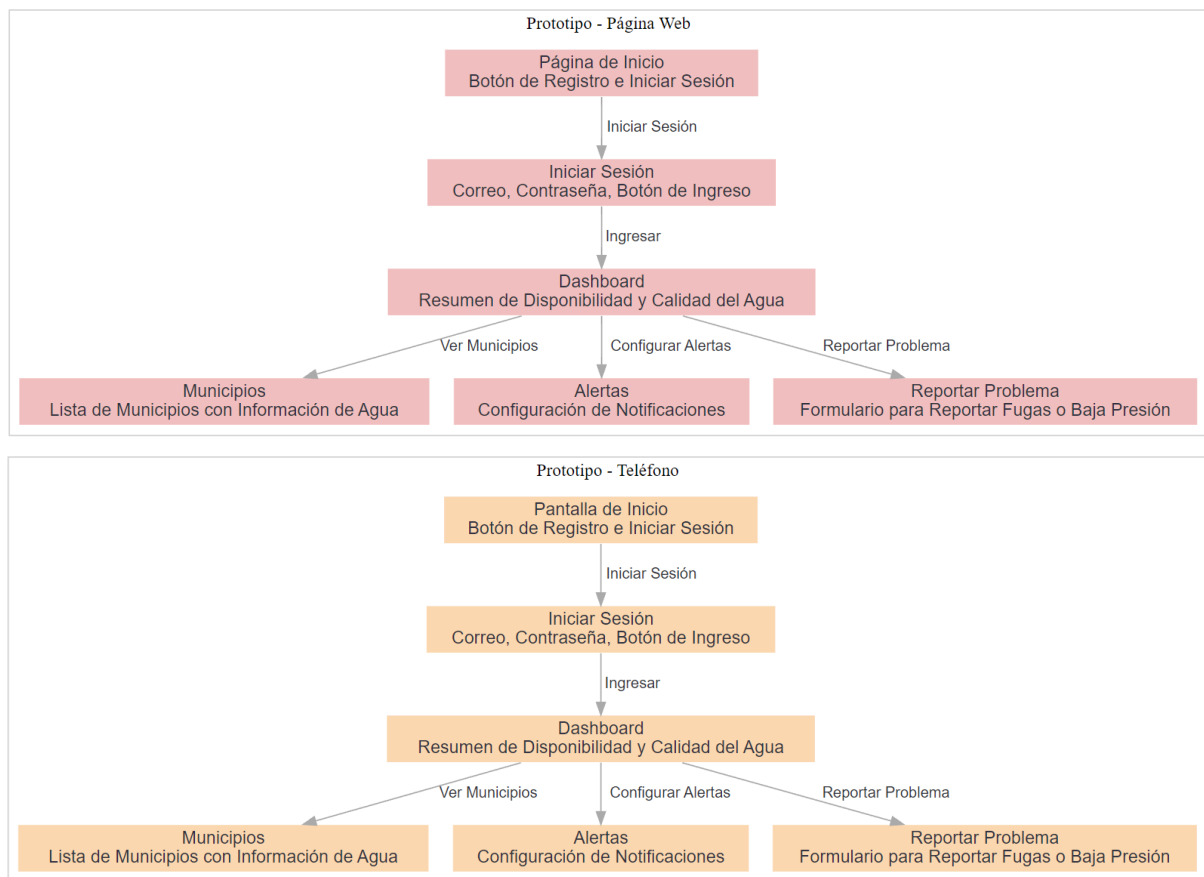
## Wireframe de la aplicación:



## Diagrama de funcionalidades







## Referencias:

<https://www.ngenespanol.com/ecologia/cual-es-la-situacion-actual-del-agua-en-mexico/>

<https://www.ngenespanol.com/ecologia/cual-es-la-situacion-actual-del-agua-en-mexico/>

<https://smn.conagua.gob.mx/es/climatologia/pronostico-climatico/precipitacion-form>

<https://blogs.iadb.org/agua/es/israel-como-la-innovacion-en-agua-y-saneamiento-puede-lograr-objetivos-nacionales-estrategicos/>