UNMDP - Facultad de Ingeniería
Departamento de Informática

Alumnos:

- Aprea, Ivan <ivanaprea.cabj@gmail.com>
- Casas, Martín Ignacio <casasmartinignacio@gmail.com>

Director:

- Hinojal, Hernán

Co-director:

- Finochietto, José Mariano

Proyecto final para optar por al grado de Ingeniero en Informática

Mar del Plata, 12 de diciembre de 2022

UNMDP - Facultad de Ingeniería
Departamento de Informática

Alumnos:

- Aprea, Ivan <ivanaprea.cabj@gmail.com>
- Casas, Martín Ignacio <casasmartinignacio@gmail.com>

Director:

- Hinojal, Hernán

Co-director:

- Finochietto, José Mariano

Proyecto final para optar por al grado de Ingeniero en Informática

Mar del Plata, 12 de diciembre de 2022

Índice

Agradecimientos

Al cuerpo docente de la Facultad de Ingeniería de la Universidad Nacional de Mar del Plata por la formación y grandes oportunidades que nos han brindado a lo largo de la carrera. En especial a nuestro director Hernán Hinojal, por su buena predisposición a la hora solicitar su participación en el proyecto y ante cada duda o consulta que surgió a lo largo de él.

A los compañeros que conocimos a lo largo de la carrera y por sobre todo a los integrantes del subsistema nube (Mariquena Sofía Gros y Pablo Porzio).

A nuestros padres, hermanos y familiares por el apoyo y afecto dados.

A Ponce AgTech por la oportunidad brindada. En especial a Mariano Finochietto quien nos acercó la propuesta de este proyecto.

Resumen

El objetivo general del proyecto es brindar una solución informática a Ponce AgTech, que permita reemplazar el sistema automatizado de riego para kiwi que poseía y que se consideraba imposible de escalar y mejorar. A partir de esta premisa, se relevó y propuso una alternativa que cubre todas las funcionalidades disponibles hasta el momento y agrega otras como la actualización de estado de conectividad y presión en tiempo real. La solución fue debatida e iterada junto a la empresa con el fin de obtener un producto mínimo viable útil.

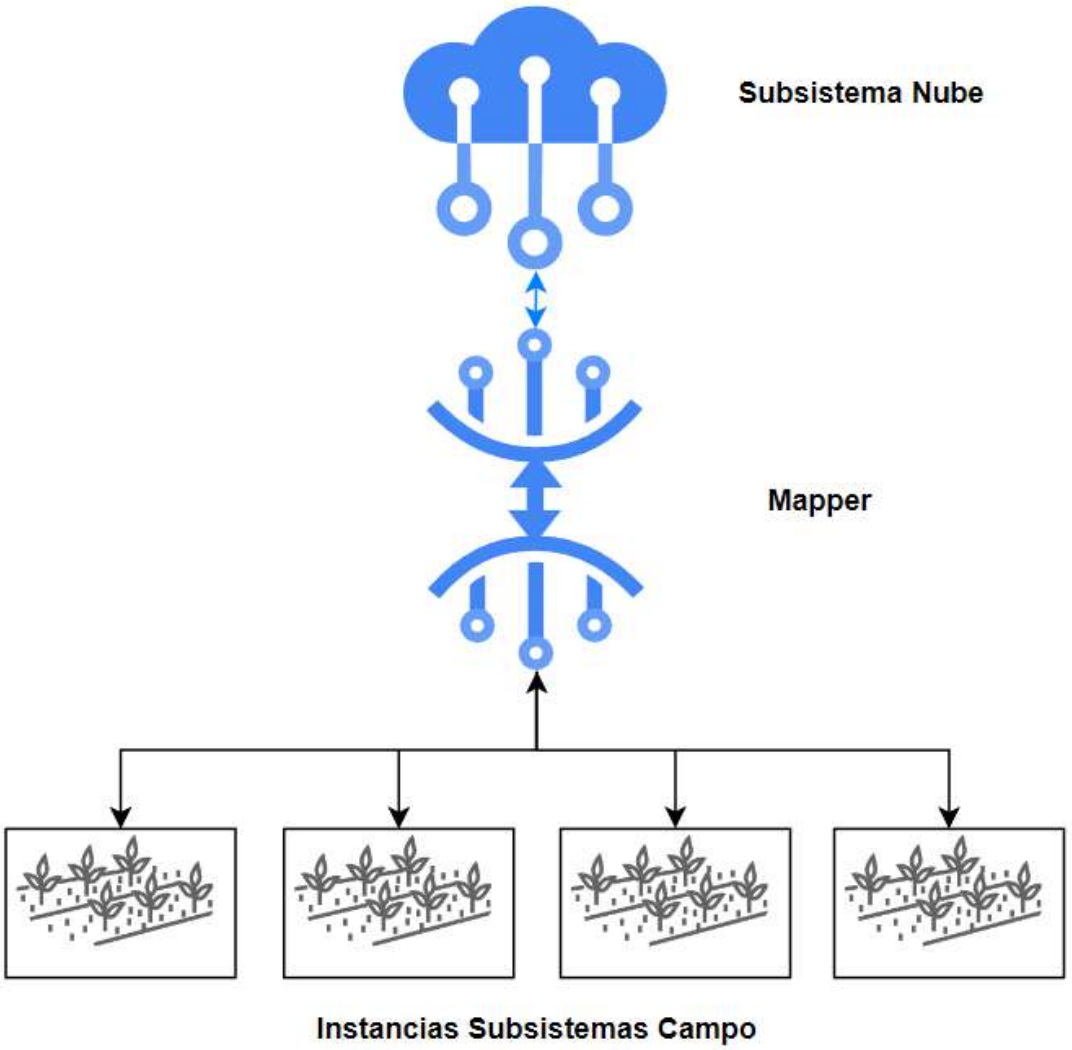

Figura 1 - Diagrama en resumen del sistema general

El producto desarrollado es un sistema de riego automatizado para kiwis, que cumple con atributos de calidad como escalabilidad y mantenibilidad, y ha sido desarrollado en conjunto por dos grupos, de forma de dividir al sistema completo en dos subsistemas con fronteras bien marcadas. Esto permitió realizar un proyecto de un mayor alcance global y fomentar el trabajo en equipo. Por un lado, existe el subsistema remoto, desarrollado por Mariquena Sofia Gros y Pablo Porzio, y está compuesto por las funciones en la nube y control remoto del sistema a través de internet. Por otro lado, se encuentra el subsistema campo (aquí descrito) integrado por Martin Ignacio Casas e Ivan Aprea, que es el encargado de la funcionalidad que administra los equipos en las plantaciones y de las posibles acciones *offline* del sistema.

El sistema en general busca automatizar los procesos de riego que son realizados de forma manual, como encender o apagar tanto válvulas como bombas, y permitir reducir tiempos muertos generados por el traslado de los encargados de riego hasta donde se encuentren los dispositivos que se desean manipular. Para esto, se siguió una metodología que permitió analizar, diseñar e implementar una solución que involucra conceptos del Internet de las Cosas (IoT) y comunicación entre sistemas de forma remota.

El proyecto reforzó gran cantidad de conceptos adquiridos a lo largo de la etapa estudiantil. Además, permitió aprender otros nuevos que resultan de gran interés para los participantes. Por lo que la experiencia generada a lo largo de él se considera un gran aporte al desarrollo profesional.

Capítulo 1: Introducción

La agricultura es una de las principales actividades económicas desarrolladas en la Argentina. Dentro de ellas se encuentra la producción de kiwi, donde un 80 % se concentra en la región sudeste de la provincia de Buenos Aires. Empresas como Ponce AgTech se interesan en este mercado y ofrecen servicios a los productores que buscan aumentar la productividad de los cultivos y minimizar costes y derroches de recursos.

El kiwi demanda de una serie de factores para crecer satisfactoriamente como lo son la humedad alta y un régimen de lluvias abundantes y frecuentes. Para ello, se utilizan sistemas de riego que conducen el agua a través de tuberías y la aplican a los cultivos de forma dosificada cada ciertos períodos. Esto requiere de personal en campo que se encargue de prender y apagar los equipos en horarios preestablecidos, decantando en una serie de problemáticas: los operadores no siempre están disponibles para realizar las tareas en el horario asignado, no se puede controlar que los planes de riego se están siguiendo correctamente y en caso de ocurrir un desperfecto, no se tiene información hasta verlo en el lugar.

Desde Ponce AgTech, diseñaron e implementaron un sistema de monitoreo y administración de riego utilizando dispositivos de Internet de las Cosas (IoT) aplicado a la metodología de riego por goteo. Debido al foco en otra línea de producto de la empresa, la solución propuesta se volvió inmantenible y solo dirigida a un productor en específico haciendo frente a sus requerimientos, pero sin posibilidad de comercializar la herramienta. Además, no permite la integración con las demás soluciones ofrecidas. Sin embargo, desde Ponce AgTech siempre se consideró a la industria del kiwi como una de gran potencial, sobre todo debido a la cercanía de la ubicación de la empresa (Mar del Plata) con el sector de mayor producción del país.

Es por esto que, en base a esta limitación que frena el avance de una línea de negocio de la empresa surge este proyecto. El objetivo principal es construir un

sistema que se pueda integrar con el resto de la gama de productos de Ponce AgTech, cubra las mismas funcionalidades existentes y que tenga como principales atributos la mantenibilidad y escalabilidad, haciendo foco en la producción de documentación para posibilitar el desarrollo futuro por parte de la empresa demandante.

Capítulo 2: Objetivos del proyecto

2.1 Objetivo principal

- Desarrollar un sistema de *Internet of Things* (IoT) de riego que le permita al productor, tanto de forma local como remota, monitorear y controlar bombas y válvulas, así como establecer programas de riego. De esta manera, le permitirá reducir los tiempos destinados a la supervisión, registrar los eventos en campo, minimizar errores, alertar sobre posibles desperfectos y tomar decisiones y ejecutarlas en el momento.

2.2 Objetivos secundarios

- Desarrollar competencias al gestionar un proyecto que atraviesa la mayoría de las etapas del ciclo de vida del software y es solicitado por un demandante real.
- Impactar positivamente en la región al no existir oferta de herramientas similares en el mercado local.
- Trabajar de manera conjunta con otro grupo en la realización del análisis inicial y manteniendo una comunicación constante a lo largo del proyecto para lograr una solución funcional y fortalecer el trabajo en equipo.
- Ampliar el conocimiento sobre las tecnologías aplicadas en el sistema a partir de su uso e investigación.

2.3 Alcance

Ambos proyectos tienen como objetivo general desarrollar una solución tecnológica de IoT que permita la administración de un sistema de riego de kiwis. La funcionalidad abarca tanto el control y monitoreo de válvulas y bombas como la gestión de programas de riego. Además, se busca desarrollar el producto de forma que sea escalable y se facilite la mantenibilidad, atributos que hoy están ausentes en el sistema en uso.

Particularmente, en el subsistema campo se deben llevar a cabo las acciones planteadas para el sistema en general, ya que es el que se encuentra en contacto con el microcontrolador elegido y traslada esas acciones a los elementos del campo. Tiene que brindar una aplicación web con una cantidad de funciones limitadas desde la casilla donde se aloja el sistema o la red local del campo, que no incluyen la creación ni modificación de programas de riego, pero sí el control y monitoreo de válvulas y bombas, así como la consulta en tiempo real de la presión de trabajo. Además, este subsistema debe facilitar una interfaz al remoto para realizar las acciones primitivas, que incluyen entre otras la alteración del estado de bombas, válvulas y programas de riego. Tiene que mantener un archivo con registro de todas las acciones que ocurran en el subsistema campo. Queda fuera del alcance de este proyecto la implementación de mecanismos de seguridad en la comunicación entre servidores, la inclusión de autenticación desde los *frontend* de campo, lo listado en los requerimientos deseados y la realización de un *testing* integral para llevar el producto a producción.

Los entregables generados son:

- Listado de requerimientos funcionales y no funcionales esperados.
- Listado de requerimientos funcionales y no funcionales deseados.
- Documentación:
    - Diagramas de arquitectura del sistema.
    - Diagrama de entidad-relación.
    - Casos de uso.

    - APIs Servidor Campo y Mapper (conexión con subsistema campo).
    - BPMN.
    - Protocolo de errores.
- Código fuente del sistema:
    - Aplicación web para que los usuarios puedan acceder desde la casilla de bombas o campo.
    - Servidor encargado de la gestión de los activos en campo.
    - Servidor intermediario encargado de la comunicación entre el servidor remoto y los servidores en cada campo.
    - Programa para el microcontrolador.

Capítulo 3: División del proyecto

Debido al gran alcance del sistema general a desarrollar, se decidió dividir al proyecto en dos: uno que se enfoque en la funcionalidad “en la nube” y otro en campo, que posea la integración con los dispositivos IoT. De esta forma, se procedió a trabajar en conjunto con los integrantes del otro grupo, lo que permitió construir un proyecto más enriquecedor. En el inicio del trabajo se realizaron constantes reuniones con el referente funcional de la empresa demandante, de forma que fue necesario que ambos grupos estuvieran presentes. Además, fue vital definir el alcance para poder tener una lista limitada y clara de tareas a ejecutar, y obtener un producto que al mismo tiempo tenga un valor elevado para Ponce AgTech. La mayor comunicación se dio especialmente en las etapas en las que se realizó el análisis de datos y requerimientos y se definió la arquitectura en los puntos de contacto de ambos subsistemas.

La oportunidad de realizar dos proyectos finales a partir de uno solo y la intervención de dos grupos en la etapa de análisis y diseño permitió definir un alcance más grande y poder brindar un producto con una mayor funcionalidad.

Capítulo 4: Aporte del proyecto

4.1 Beneficiarios

Ponce AgTech[(0)] (Caloian SAS) es la empresa demandante del proyecto. Constituida en 2019 en la ciudad de Mar del Plata, brinda servicios de monitoreo y detección de fallas para equipamiento de riego mediante sistemas basados en IoT y en la nube. Cuenta con clientes en la zona, como Balcarce y Sierra de los Padres; también en provincias, como Santa Fe, Cordoba y Rio Negro y en el extranjero. Los principales beneficios que brindan mediante sus sistemas son: reducir los costos de transporte, evitar fallas e imprevistos, registrar de forma precisa el riego y controlar los regadores.

En el caso particular de este proyecto, Ponce AgTech ve la oportunidad de impactar en el mercado del riego por microaspersión y goteo, al contar con clientes potenciales en la zona de Sierra de los Padres.

4.2 Impacto

Se pueden considerar distintos enfoques para analizar el impacto que tiene el proyecto final:

- Impacto sobre Ponce AgTech
    - Dado que la mayor parte de la producción de kiwis a nivel nacional se encuentra en la región del sudeste bonaerense, se cuenta con una gran cantidad de potenciales clientes del producto.
    - Alta competitividad en el mercado, caracterizado además por una competencia prácticamente nula, lo que permitiría obtener mayores ganancias.
- Impacto sobre la economía regional
    - Mayor productividad de los campos usuarios, debido a reducción de costos de transporte, monitoreo, supervisión y facilidad en la toma de decisiones.

    - Posibilidad de detección de desperfectos a tiempo y remotamente. Reducción de costos de mantenimiento y reparaciones. Aumento de las cantidades de kiwis producidas al disminuir los productos desechados a causa de un riego defectuoso.
- Impacto sobre el medio ambiente
    - Si bien el riego por goteo es el sistema más eficiente en cuanto al consumo de agua, al estar automatizado permite brindar a las plantas de kiwis los litros justos y necesarios de forma precalculada para un correcto crecimiento y productividad. Además, teniendo en cuenta la detección de desperfectos nombrada anteriormente, disminuye en gran cantidad el desperdicio que se podría generar por este motivo.

4.3 Análisis FODA

A continuación se presentan los resultados del análisis FODA realizado por el equipo sobre el proyecto.

- Fortalezas
    - Experiencia laboral previa de todos los integrantes del equipo de desarrollo sobre tecnologías de desarrollo web, como Javascript, Node.JS, Ionic, entre otras. Algunas de estas tecnologías son utilizadas por la empresa demandante para el desarrollo de sus sistemas.
    - Motivación del equipo de desarrollo en aprender y obtener experiencia sobre sistemas distribuidos, servicios en la nube e IoT.
    - Existencia de un prototipo de referencia provisto por Ponce AgTech, útil especialmente para el análisis de requerimientos.
- Oportunidades
    - Asistencia sobre el dominio, gestión de proyectos y disponibilidad de recursos (placas Arduino y Raspberry Pi) de Ponce AgTech.
    - Competencia en cuanto a producto prácticamente nula en la región.
    - Mercado potencial, especialmente en la región.

- Debilidades
    - Nula experiencia del equipo de desarrollo en sistemas y hardware para riego y sistemas embebidos.
    - Disponibilidad horaria acotada de todos los integrantes.
- Amenazas
    - Aparición de una tecnología de riego superior a la de goteo/microaspersión que vuelva al producto obsoleto.
    - Cierre de Ponce AgTech que anule la asistencia y recursos brindados para el proyecto.
    - Cambios de objetivos / estructura interna de Ponce AgTech, que vuelvan incompatible lo desarrollado con el resto de sus productos.

4.4 Análisis de riesgos

Se han considerado otros factores que pudieran hacer inviable o afectar seriamente al proyecto antes de su finalización, además de los contemplados en el análisis FODA. Se tiene en cuenta para cada riesgo potencial su probabilidad de ocurrencia (Po) e impacto sobre el proyecto (I), los cuales se califican con valores del 1 al 3, siendo 3 el de mayor valor.

Se calcula el peso como el producto entre la probabilidad de ocurrencia y el impacto. Si resulta mayor o igual a 6, se debe tener un plan de contingencia para aplacar su efecto ya que su impacto es lo suficientemente grande y su probabilidad de ocurrencia es elevada.

Riesgo	**Consecuencia**	**Po**	**I**	**Peso**
Cierre de Ponce AgTech	Cancelación del proyecto	1	3	3
Integrante abandona el proyecto	Pérdida de mano de obra y conocimiento	1	3	3
Desconocimiento sobre tecnologías, sistemas de riego	Posibles demoras, dudas sobre dónde se originan errores	1	2	2

Imposibilidad de coordinar reuniones con demandante	Retraso en validaciones, cambios en requerimientos inoportunos	2	3	**6**
Mala integración de componentes	Demora por retrabajo	1	3	3
Librerías de terceros desactualizadas durante desarrollo	Producto con errores, vulnerabilidades y sin novedades útiles de versiones reciente	3	2	**6**
Ausencia momentánea de integrante por razones personales	Pérdida de mano de obra y conocimiento	2	3	**6**
Pérdida de interés del demandante en el proyecto	Cancelación del proyecto	1	3	3
Rotura de microcontroladores	Imposibilidad de continuar desarrollo o pruebas	2	3	**6**
Abandono del proyecto de la totalidad del equipo del subsistema remoto	Imposibilidad de continuar con el desarrollo integral del sistema.	1	3	3

Figura 2 - Análisis de riesgos obtenido durante el desarrollo del trabajo final de la materia Taller de Programación 2

Planes de contingencia desarrollados:

- Imposibilidad de coordinar reuniones con el demandante: contar con más de un referente para resolver consultas y para obtener información específica del dominio. También, pactar de antemano un día y horario en la semana para resolver consultas si las hubiera.

- Librerías de terceros desactualizadas durante el desarrollo: actualizar librerías en otra rama del proyecto.

- Ausencia momentánea de integrante por razones personales: compensar las horas hombre de desarrollo perdidas aumentando las de los otros integrantes por el tiempo necesario.

- Rotura de microcontroladores: solicitar a Ponce AgTech microcontroladores nuevos, ya contemplados en el presupuesto del proyecto.

Al comenzar el proyecto aún existía el riesgo de una nueva cuarentena por COVID-19. Desde el equipo de campo se necesitaba del hardware para poder desarrollar y hacer pruebas. La falta de éste podría llevar a retrasos en el desarrollo, por lo que el riesgo fue eliminado al solicitar a Ponce prestado el equipo al inicio del proyecto.

Capítulo 5: Estimación inicial

De forma de poder realizar estimaciones sobre los tiempos del proyecto, inicialmente fue vital entender el dominio del problema y el alcance. Se trabajó en conjunto con el equipo del subsistema nube en el análisis para definir las tareas generales a ser desarrolladas y el tiempo demandado por cada una. Sin embargo, no se contaba con demasiada experiencia en estimaciones para proyectos de esta escala, por lo que se decidió estimar teniendo una mirada pesimista de la duración que abarcaría cada tarea. Además, los cálculos de tiempo surgían como un requisito para la escritura del protocolo del trabajo final. Esto facilitó la definición del alcance que luego se tradujo en requerimientos específicos.

Teniendo en cuenta que la colaboración y comunicación de los dos grupos debían ser prácticamente permanentes en las etapas iniciales, las estimaciones debían ser realizadas respetando los tiempos laborales de todos los integrantes, de forma que las tareas de diseño pudieran realizarse de la manera más paralela posible. Esta metodología fue necesaria para determinar interfaces y límites claros entre ambos subsistemas, para lograr un desacople y parcial independencia en el trabajo de cada uno a futuro.

La estimación inicial seguía una metodología de cascada. En ella se realiza cada etapa bien definida, y la próxima comienza cuando la anterior fue terminada en su totalidad. Como se puede ver en el diagrama, el orden fue: análisis, diseño, implementación y *testing*, y cierre del proyecto. Esta metodología aportó a realizar un amplio análisis e investigación en las etapas iniciales del trabajo.

Es importante aclarar que los módulos presentes en las estimaciones fueron ideados antes de desarrollar una arquitectura concreta, por lo tanto no tuvieron en cuenta las funcionalidades y el desarrollo que llevaría cada una en el subsistema correspondiente, sino que se pensó en el sistema general.

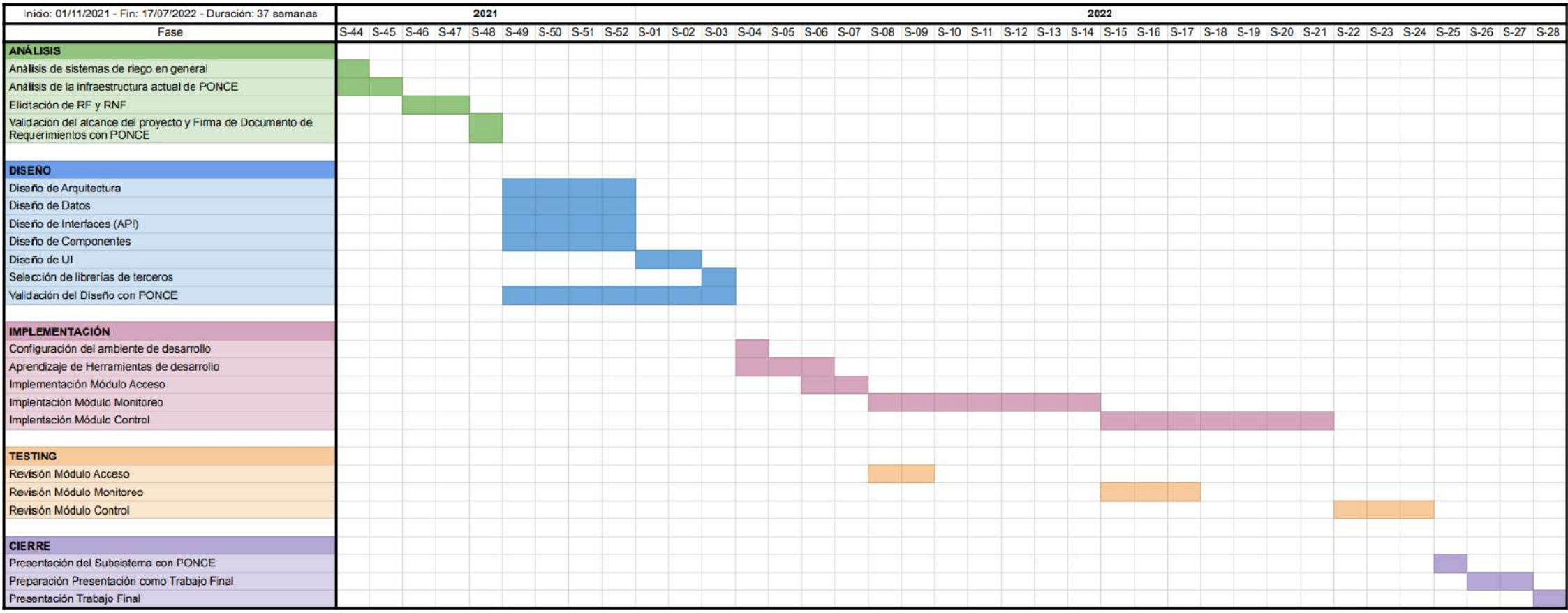

Figura 3 - Diagrama de Gantt presentado con el protocolo

Capítulo 6: Metodologías

6.1 Metodologías de trabajo

Al comenzar con el proyecto se definió la metodología de trabajo a utilizar. Debido a que la información que se tenía acerca del dominio del problema era poca y en busca de minimizar el retrabajo, se optó por iniciar con una metodología en cascada. En este enfoque, las etapas se llevan a cabo de forma completa para permitir continuar por la siguiente. Este tipo de trabajo mantiene sus desventajas, ya que minimiza la retroalimentación que se recibe del cliente y puede llevar al proyecto a no cumplir con sus expectativas. Es por esto que se decidió solo utilizarla en la etapa inicial para dedicar el 100 % del tiempo al análisis y aprovechar la disponibilidad del referente de la empresa demandante.

Para la continuación del proyecto se definió la utilización de la metodología *Scrum*(1) en busca de presentar valor en cortos períodos de tiempo y realizar una mejora continua del producto. Esta forma de trabajo tiene como base reconocer que el equipo no tiene todo el conocimiento en el inicio del proyecto y evolucionará a través de la experiencia. Para hacer uso de esta metodología se debe dividir el tiempo en períodos constantes, para este caso se decidió realizar una división de dos semanas. A cada una de estas se les da el nombre de *sprint*. Como inicio del ciclo se realiza una reunión de planeamiento donde el equipo decide qué tareas va a realizar durante el *sprint* y quién lo hará. Se intenta hacer uso de todo el tiempo pero sin excederlo. *Scrum* utiliza tres herramientas principales: un *backlog* del producto, donde se pueden encontrar todas las tareas por hacer que forman parte del proyecto y que a su vez es dinámico, por lo que puede crecer a lo largo del desarrollo; un *backlog* de *sprint*, dentro del que se encuentran todas las actividades que tendrán lugar y un incremento que hace referencia a lo que se entregará al finalizar. Todas las tareas presentan estados y se separan en: uno inicial “por hacer”, intermedios “en curso” y “en revisión” y uno final “finalizada”. La etapa “en curso” es aquella en la que el encargado realiza el desarrollo de la tarea y cuando

se mueve al estado “en revisión”, un referente del equipo valida el trabajo y en caso de estar terminada le coloca el estado “finalizada”.

El punto más importante por el cual se eligió esta forma de trabajo frente a continuar con la metodología cascada fueron sus ventajas y el conocimiento previo por los miembros involucrados en el proyecto general. *Scrum* fue utilizado en actividades de algunas cátedras a lo largo de la carrera y además, ambos equipos la utilizan en su vida profesional debido a que su uso está estandarizado. Como es de esperar con este enfoque, todas las tareas eran validadas con el referente de la empresa demandante quien definía si fue correctamente logrado el alcance o debía realizarse algún cambio.

6.2 Metodologías de análisis

La mayor parte del análisis de requerimientos del proyecto fue llevada a cabo durante el inicio. Para obtener información sobre el dominio del problema se recurrió principalmente al referente funcional de Ponce AgTech y al co-director. En un primer momento se tuvo una exposición detallada de parte del demandante en la que se explicó cómo funcionaba el sistema utilizado hasta el momento y la distribución que presentaba. Luego, se tuvo una reunión en un campo con un encargado de riego de una empresa cliente de Ponce AgTech, el cual ayudó a brindar una mirada más amplia sobre la funcionalidad esperada y deseada por los usuarios finales y también dio a conocer los problemas presentes en el sistema utilizado hasta ese momento.

Los encuentros con el referente funcional de Ponce AgTech y el co-director se realizaron mayormente mediante reuniones virtuales, las cuales eran organizadas todas las semanas en un día concreto en el que todos contaran con disponibilidad. Cada llamada que se realizó permitió obtener una lista de requerimientos más detallada, ya que se iteraba semanalmente sobre una lista inicial, se refinaban las descripciones de cada uno y se agregaban y eliminaban otros. Finalmente, se obtuvo una lista que, tanto los integrantes, el referente funcional de Ponce AgTech, el director y el co-director consideraron con un alcance válido, de tal forma que permitió concebir al proyecto como dos trabajos finales de carrera y servirle al demandante como una herramienta eficaz y que sea fácilmente mejorable luego de entregada. Además, en el documento de requerimientos funcionales y no funcionales obtenido, se agregó un apartado de requerimientos deseados, los cuales se consideraron como no necesarios dentro del alcance del proyecto, pero podrían realizarse eventualmente para agregarle más valor y serían entregados posteriormente para la continuidad del desarrollo del sistema.

6.3 Metodologías de diseño

Luego de la etapa de análisis y con los requerimientos planteados, comenzó la realización del diseño del sistema. Durante ella, se construyó un boceto de la arquitectura con el objetivo de cumplir con lo solicitado y se utilizó solo el conocimiento y criterio de los integrantes. El resultado en esta etapa se obtuvo luego de una serie de encuentros en los que a partir de cada requerimiento se desprendían nuevas problemáticas a resolver sobre lo planteado anteriormente. El diagrama de arquitectura logrado fue la pieza clave para continuar con el desarrollo del proyecto ya que permitió obtener una visión global del sistema y mostrar cómo se desarrollaba el flujo de cada funcionalidad. En todo momento del diseño se hizo foco en generar un producto escalable y que, en el caso del subsistema campo, no sea completamente dependiente del otro.

Con la definición de las entidades también se dió la de las tecnologías a utilizar. Previo a la entrega del protocolo se tuvo una reunión con un referente técnico de la empresa demandante con la idea de poder utilizar las mismas que ellos aplican en sus proyectos y de esta forma puedan continuarlo en el futuro.

Se presentó la propuesta al director y co-director, de donde surgieron recomendaciones y cambios a realizar en pos de mejorar el sistema. En una de las reuniones, desde Ponce AgTech se presentó la posibilidad de utilizar el sistema por comunicación satelital y esto se tradujo en un cambio en el diseño y la aparición de la nueva entidad Mapper que funcionaría como puente entre los dos subsistemas. El diseño se continuó iterando a lo largo del desarrollo del proyecto.

En busca de lograr independencia de trabajo entre los equipos, se diseñó un modelo de datos para que no fuera necesario realizar encuentros para definir detalles sobre comunicación entre sistemas antes del desarrollo de cada funcionalidad. Este modelo fue validado por el referente técnico de Ponce AgTech quien brindó un gran aporte sobre los datos que debían ser utilizados y la forma en que la empresa los usa.

6.4 Metodologías de desarrollos y pruebas

Al contar con un diseño claro, la tarea de implementación resultó una práctica ya conocida dada la experiencia laboral de los integrantes del equipo. En todos los proyectos de *software* donde se había trabajado antes o simultáneamente a este, se utilizó siempre la plataforma Jira, aplicación web de Atlassian, para la asignación y control de tareas, Github para alojar los repositorios de código y Gitflow[2] como flujo de trabajo con Git.

Gitflow consiste en tener en cada repositorio dos ramas fundamentales, una principal (llamada *master*) o de producción (llamada *main*), y una de desarrollo (llamada *develop*). Para agregar nuevas funcionalidades puntuales, se crea una rama nueva partiendo de la de desarrollo, usualmente con el nombre y código de la tarea a implementar. Una vez terminado el trabajo se crea una solicitud de incorporación de cambios, llamada *pull request,* desde esa rama hacia la de desarrollo. Además, en determinados momentos se hace la misma solicitud desde la rama de desarrollo hacia la de producción, usualmente para presentar una lista de funcionalidades nuevas a los interesados del proyecto.

Para realizar un *pull request*, el desarrollador debía completar una plantilla definida al inicio de la implementación, utilizada para registrar de forma más organizada qué tipo de cambios se introducian al fusionar esa rama, si se introducen nuevas librerías o no, si se incluían tests para la funcionalidad desarrollada, imagenes si aplicaba, entre otras. También, se debía incluir un *reviewer* para que analizara el código escrito e informara si encontraba errores o partes de código que afectaban funciones ya estables, de modo que el creador del *pull request* corrigiera esos errores y actualizara su rama.

Adicionalmente, para lograr un estilo de código unificado se utilizaron reglas de *linting* personalizadas, que definen ciertas características de estilo de escritura como la longitud máxima de una línea, buenas prácticas para definición de funciones, cantidad de líneas vacías toleradas, entre otras. Para la realización de

este proyecto se utilizó ESLint, una herramienta que señala en el código aquellos errores de *linting* en base a una serie de reglas definidas. Y en cuanto a la configuración, se optó por tomar la recomendada por la herramienta, utilizada en una gran cantidad de proyectos. ESLint fue integrada con Git para no permitir subir código que contenga errores de *linting*.

Es importante mencionar también, que se utilizó Postman como la plataforma para almacenar datos de las solicitudes REST de la API y ejemplos para facilitar su uso. Distintas peticiones fueron agregadas al espacio de trabajo de la plataforma a medida que eran desarrolladas, para facilitar la implementación futura del *frontend*, donde se utilizarían.

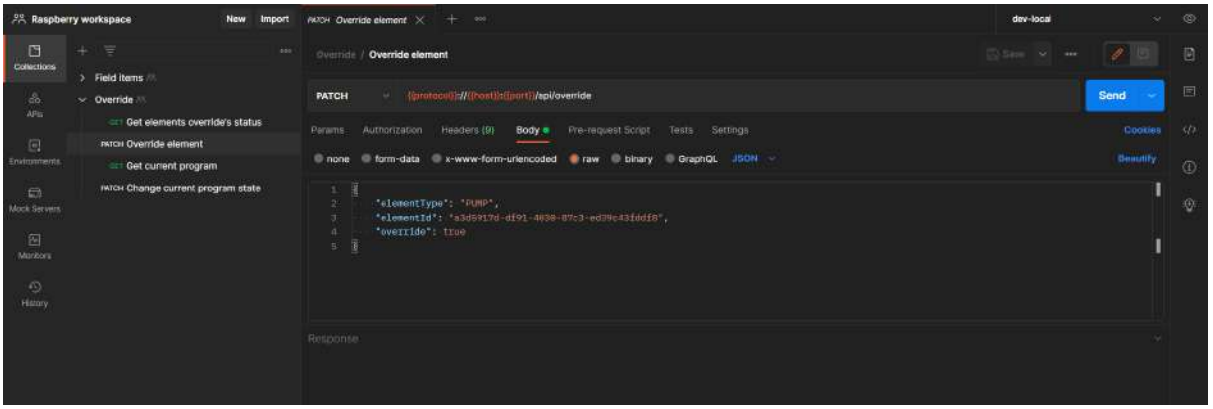

Figura 4 - Ejemplo de una solicitud al *backend* campo

Con respecto al *testing*, se ha realizado cierto número mediante la herramienta Jest, que permite definir rápidamente un entorno de pruebas y datos simulados. Los *tests* son ejecutados tanto por el desarrollador cuando lo desea como por Git antes de poder enviar los cambios al repositorio. Esto es para mantener la estabilidad y correcto funcionamiento del código ya sincronizado con las ramas principales. Sin embargo, dado que el proyecto consiste en desarrollar un producto mínimo viable, no se ha tenido en cuenta que los *tests* tengan una cobertura total sobre el producto. Por este motivo, se ha hecho hincapié en el *testing* automático para las funcionalidades primitivas del *backend* del subsistema, como las solicitudes de Hypertext Transfer Protocol (HTTP) para obtener o alterar el estado de bombas y válvulas.

Debido a que no se encontró una herramienta que propicie el *testing* de la comunicación del Mapper con el subsistema campo de forma automática utilizando gRPC, se realizaron pruebas manuales simulando las peticiones.

En cuanto al *frontend*, no se han realizado pruebas en él ya que la validación de la información que se envía se realiza con mayor énfasis en el *backend*, más allá de algunas validaciones de tipos simples presentes en el *frontend*.

Al no encontrar una forma eficiente de realizar *testing* en el microcontrolador se recurrió a realizar pruebas de forma manual. Se simuló el encendido de los dispositivos como válvulas y bombas con leds conectados a los puertos *gpio* del microcontrolador. La realización de estas pruebas llevó una gran cantidad de tiempo, sobre todo en aquellas que involucraban programas y se ejecutaban en un horario en específico.

Capítulo 7: Análisis del problema

7.1 Dominio del problema

Hoy en día para el cultivo de kiwi no se abastecen las necesidades de agua mediante lluvias ya que éstas no son distribuidas uniformemente a lo largo del año ni suficientes. Esto conlleva a realizar un aporte de aguas de riego y para esto se utiliza el sistema de microaspersión, que consiste en una aplicación periódica de pequeños volúmenes de agua a los cultivos mediante una red de tuberías y dispositivos emisores.

La temperatura óptima para el cultivo de kiwis es alrededor de 25-30ºC y una humedad relativa suficientemente alta que oscila sobre el 60%[(3)]. Se recomienda un aporte de agua en fechas de actividad vegetativa con una cantidad aproximada de 1000 mm^2/m cada mes.

Actualmente en Argentina, la producción de kiwi se estima entre 8000 y 9000 toneladas anuales, sin poder cubrir la demanda local. Según la presidente del Honorable Concejo Deliberante de General Pueyrredon, Marina Sánchez Herrero “En la región del partido de General Pueyrredón y sus aledaños se produce el 80 % del kiwi nacional que se consume en el país”[(4)].

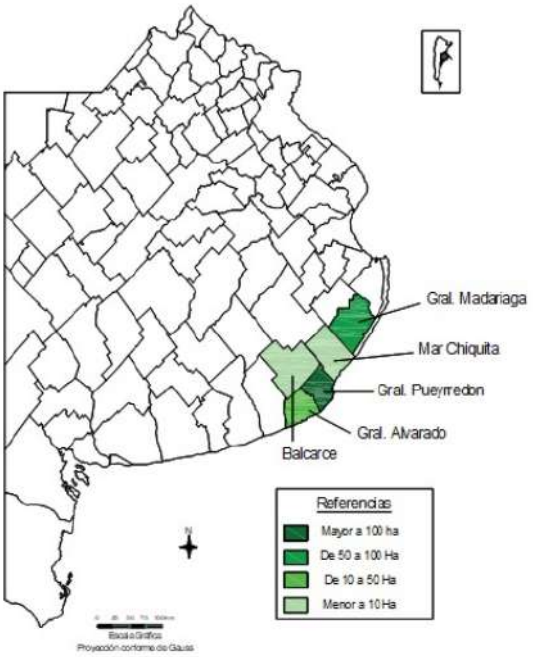

Figura 5 - Hectáreas de kiwi cultivadas por partido en la pcia. de Buenos Aires

7.2 Problema a resolver

Actualmente, es normal que las plantaciones de kiwi sean regadas manualmente por los operarios, pero esta actividad presenta puntos débiles. En primer lugar, los encargados de riego deben prender y apagar las bombas y válvulas del sistema con distintos tiempos entre sí para que no ocurran desperfectos técnicos. A su vez, los horarios de riego pueden resultar incómodos para el personal que se encuentra en el campo o forzar a que haya una persona trabajando solo para realizar la tarea de encendido y apagado. Además, el traslado a las casillas quita tiempo que podría ser empleado en otras tareas. En segundo lugar, como se explicó, el kiwi necesita de ciertas condiciones para crecer saludablemente y si no son cumplidas podrían llevar a la pérdida de los cultivos y recursos energéticos que se traduce en pérdidas monetarias.

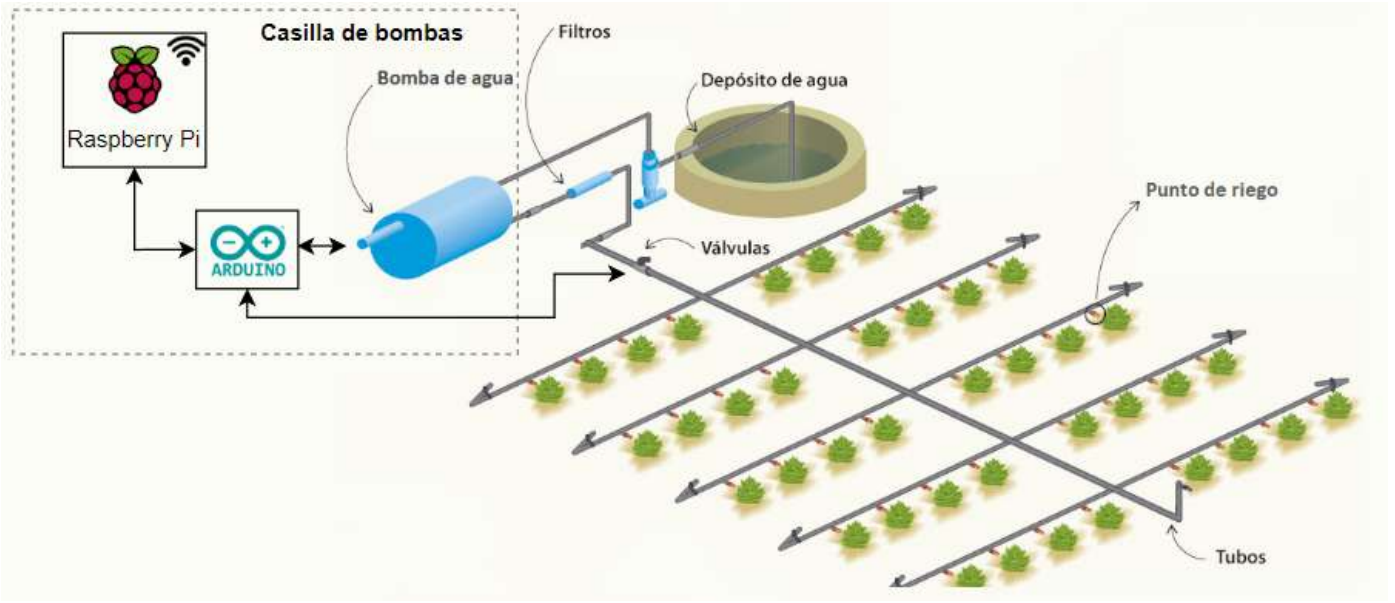

Figura 6 - Distribución de los elementos en campo

Una distribución similar a la de la Figura 6 es la seguida por Ponce AgTech en las plantaciones de kiwi y otros de sus productos.

Desde este subsistema se hace foco en cuatro entidades principales:

- La bomba.
- La válvula.
- El campo.
- El programa.

7.2.1 Bomba y válvula

Ambas herramientas conforman elementos claves para la funcionalidad del sistema. Las bombas son las encargadas de la distribución del agua a través del sistema de tuberías mientras que las válvulas son las encargadas de abrir o cerrar el paso de agua a los cultivos. Ambos elementos pueden ser encendidos y apagados desde la casilla en campo o a través del sistema remoto. Las bombas cuentan con diferentes presiones de trabajo, que varían de acuerdo a las distancias que el agua debe recorrer, además de contar con valores mínimos y máximos. Existen válvulas de

alivio que no son controladas por el sistema y se activan en el caso de un exceso de presión para aliviar la carga y evitar daños. Estos elementos se encuentran dentro de la casilla de bombas para evitar su deterioro por parte del clima.

7.2.2 Campo

Lugar donde se desarrolla la actividad y está colocado el sistema. Desde el punto de vista de este subsistema, el campo es único y se encuentra en donde están posicionados los componentes de control.

7.2.3 Programa

Dentro de la organización dedicada al cultivo se definen programas de riego en busca de mantener los cultivos saludables. Cada uno consta de una serie de bombas y válvulas que se prenden en un momento dado y se apagan luego de transcurrir un período de tiempo. Los programas que existen pueden ser múltiples, pero para el alcance de este proyecto solo será posible mantener solo uno como activo en cada instancia del subsistema campo.

7.3 Flujo de trabajo

En el siguiente diagrama de Modelo y Notación de Procesos de Negocio se muestra cómo es la interacción con el sistema por parte de los usuarios tanto regadores como encargados del campo. Contiene la secuencia de acciones y decisiones posibles a tomar a la hora de crear y en el transcurso de un programa de riego. A su vez, expone el flujo que ocurre al apagar o prender una bomba o válvula individualmente que no son mostradas en diagramas aparte debido a su simplicidad.

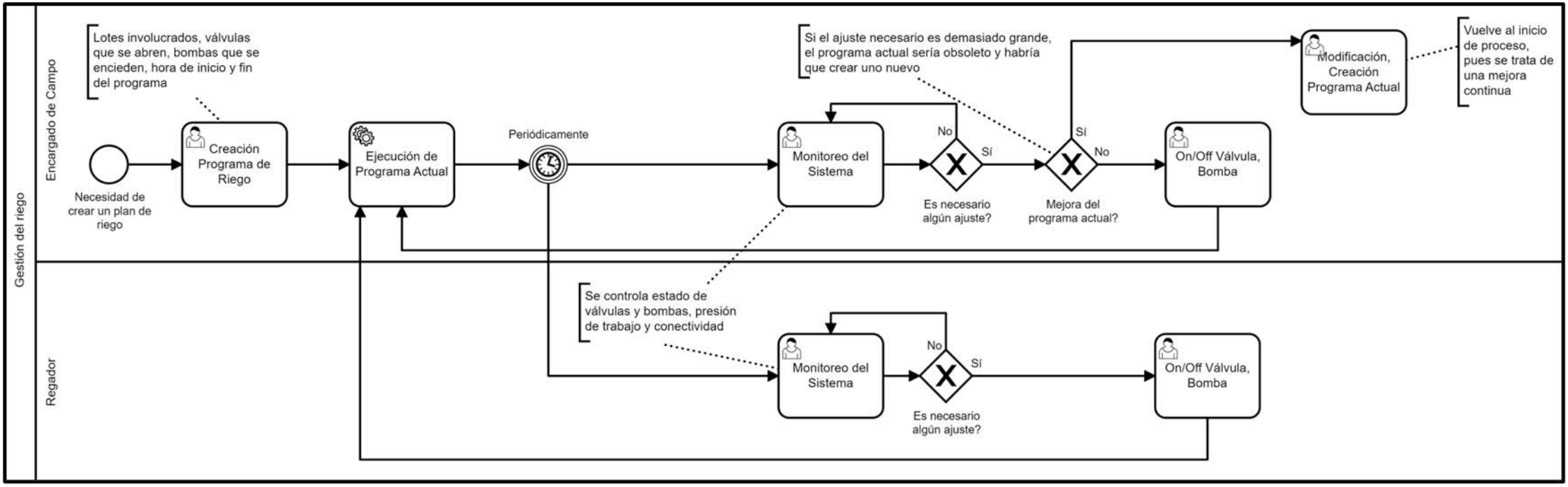

Figura 7 - BPMN principal

7.4 Requerimientos

A continuación se listan todos los requerimientos para el sistema que fueron relevados durante la etapa de análisis. Sólo los esperados fueron implementados, mientras que el resto quedan como trabajo futuro.

7.4.1 Requerimientos Funcionales Esperados

Monitoreo

- RFM1: el sistema deberá permitir al Usuario monitorear el estado de la conectividad con el subsistema en campo.
- RFM2: el sistema deberá permitir al Usuario monitorear el estado (ON/OFF) de las bombas eléctricas.
- RFM3: el sistema deberá permitir al Usuario monitorear el estado (ON/OFF) de las válvulas.
- RFM4: el sistema deberá permitir al Usuario monitorear la presión de trabajo de riego, correspondiente a la bomba.

Control

- RFC1: el sistema deberá permitir al Usuario encender/apagar remotamente las bombas.
- RFC2: el sistema deberá permitir al Usuario encender/apagar remotamente las válvulas.
- RFC3: el sistema deberá permitir al Usuario la creación, modificación y eliminación de programas de riego.
    - El sistema deberá permitir al Usuario definir el tiempo de apertura y cierre de cada válvula y bomba en un programa.
    - El sistema deberá permitir al Usuario agrupar diferentes válvulas conformando lotes, cada uno con un nombre asociado. Las válvulas pueden estar en más de un lote a la vez.

    - El sistema deberá permitir al Usuario utilizar la misma válvula o bomba más de una vez en el mismo programa, con el fin de crear programas de mayor duración.
- RFC4: el sistema deberá permitir al Usuario la selección del programa de riego a ejecutar, definido previamente.
- RFC5: el sistema deberá encenderse de acuerdo al horario de inicio del programa seleccionado, y pararse de acuerdo al horario de parada del programa.
- RFC6: el sistema deberá permitir al Usuario forzar (*override*) una válvula o bomba: siendo parte de un programa de riego, debe pasar a control manual y ya no seguir los pasos.

Acceso

- RFA1: el sistema permitirá al Usuario acceder a la plataforma web remota mediante el ingreso de nombre de usuario y contraseña. En cambio, el Usuario accederá directamente a la web del campo y la casilla, sin necesidad de autenticarse.
- RFA2: el sistema registrará para cada usuario los siguientes datos: nombre, apellido, email, teléfono y organización.
- RFA3: el sistema deberá permitir al administrador realizar todas las acciones de un usuario normal, pudiendo también realizar:
    - Altas, bajas y modificaciones de usuarios, campos y organizaciones.
    - Altas, bajas y modificaciones de bombas y válvulas.

7.4.2 Requerimientos No Funcionales Esperados

- RNF1: el sistema constará de una plataforma web dividida en tres instancias:
    - El sistema podrá ser accedido desde cualquier lugar del mundo y cualquier dispositivo.
    - El sistema podrá ser accedido desde el *hotspot* (red WiFi) propio de la placa Raspberry Pi, dentro de la casilla de bombas del campo. Esta versión de la plataforma se servirá desde un servidor web en la misma placa.
    - El sistema podrá ser accedido desde la red local del campo, mediante cualquier dispositivo. Se debe proveer la plataforma del punto anterior conectando la Raspberry Pi a una red entre la casilla de bombas y el campo, la cual queda fuera del alcance de este proyecto.
- RNF2: la UI de la plataforma web será adaptable a cualquier dispositivo móvil y de escritorio.
- RNF3: el sistema deberá ser implementado con las siguientes tecnologías:
    - Para interfaces web se usará Vue.js
    - Para base de datos se usará PostgreSQL.
    - Para servidores se usará Node.js.
- RNF4: el sistema deberá utilizar en cuanto a hardware:
    - Una Raspberry Pi como unidad central, provista por Ponce AgTech.
    - Un conjunto de placas Arduino que serán esclavos de la Raspberry Pi, provisto por Ponce AgTech.
    - El protocolo de comunicación entre la Raspberry Pi y las Arduino será sobre serial, utilizando uno existente o uno hecho a medida.
    - Las Arduino usarán *shields* hechos a medida y provistos por Ponce AgTech.

- RNF5: el sistema deberá ser dimensionado de manera tal que no se vea limitada la cantidad de válvulas a usar.
- RNF6: el monitoreo de la conectividad del sistema en campo se hará mediante la técnica de *Heartbeat*.
- RNF7: la comunicación entre los sistemas y la nube debe realizarse a través de un protocolo lo suficientemente liviano, en términos de bytes por mensaje, para que en un futuro, fuera del alcance de este proyecto, se pueda implementar comunicación satelital. De momento, se hará por TCP en capa de transporte.
- RNF8: el mapeo de válvulas y bombas a puertos deberá estar localizado en un archivo de configuración almacenado en la Raspberry Pi. Este podrá ser modificado utilizando el puerto serie de la placa o mediante SSH.
- RNF9: las válvulas de alivio no se mostrarán al usuario desde la interfaz, de manera de no poder apagarlas/prenderlas.
- RNF10: el sistema llevará un log de lo acontecido en el backend, con las siguientes características:
    - Habrá un log de errores para fallas en el sistema.
    - Habrá un log de información sobre las acciones de los usuarios, que ayudará en la comprensión de los errores.
    - Habrá un log de debug, que se escribiría al mismo tiempo que el de información, pero con más detalles sobre las acciones de los usuarios.
    - El log de debug no funciona por defecto, sino que debería poder activarse sólo cuando se necesita mediante una variable específica para ello.
    - Los archivos de log deben ser comprimidos una vez finalizado el día.
- RNF11: el sistema debe ser construido de manera tal que soporte varios sistemas de riego en simultáneo, cada uno con su *hardware* propio.

7.4.3 Requerimientos Funcionales Deseados

Disponibilidad

- RFDD1: el sistema deberá, en caso de perder conectividad a Internet, almacenar y reenviar los mensajes de comunicación general entre el campo y la nube, una vez restaurada la conexión. Se deben mantener los mensajes hasta que puedan ser enviados o durante una campaña de riego, lo que suceda primero.

Monitoreo

- RFDM1: el sistema deberá detectar cuando comienza a llover. Si en ese momento el sistema estaba regando, y se supera un cierto valor mínimo de lluvia, se detendrá el riego.

Control

- RFDC1: el sistema deberá permitir al Usuario definir límites de presión por cada lote.
    - Presión máxima admitida: si el equipo sobrepasa esta presión deberá apagarse.
    - Presión mínima admitida: si el equipo se encuentra por debajo de esta presión deberá notificar al usuario.
    - Presión de alerta: Es un valor menor a la presión máxima admitida y dispara un mensaje de alerta al usuario para avisarle que la presión es demasiado alta.
    - El sistema deberá proceder a detener el equipo y apagar las bombas, si detecta que el valor de presión sobrepasó el límite definido.
    - El sistema deberá permitir al Usuario deshabilitar este control.

- RFDC2: el sistema permitirá al Usuario seleccionar simultáneamente más de un programa, no solapados temporalmente. El sistema ejecutará cada uno de los programas seleccionados en el momento correspondiente.

Alarmas

- RFDA1: el sistema deberá notificar al Usuario cuando se detenga involuntariamente antes de finalizar el programa de riego seteado, debido a fallas en las bombas, en la alimentación eléctrica o cuando se alcance la presión máxima admitida (RFDC1).
- RFDA2: el sistema deberá notificar al Usuario cuando el sistema de riego sea iniciado o detenido localmente.
- RFDA3: el sistema deberá notificar al Usuario cuando la presión de un lote se encuentre por debajo de un límite inferior definido o cuando se alcance la presión de alerta (RFDC1).
- RFDA4: el sistema deberá notificar al Usuario cuando se inicie un programa de riego.
- RFDA5: el sistema deberá permitir al Usuario desactivar las alarmas según los tipos definidos en los requerimientos anteriores.

Reportes

- RFDR1: el sistema deberá llevar un registro de los siguientes datos, teniendo cada valor una marca de tiempo asociada: cuándo se abren y cierran las válvulas, las presiones reportadas, hora local de la Raspberry Pi, cuándo se activa un programa, cuánta agua fue regada y, en caso de lluvia, cuánta agua cayó.
- RFDR2: el sistema deberá permitir al usuario descargar los datos de RFDR1.

- RFDR3: el sistema deberá permitir al usuario borrar los datos almacenados, pero si no son borrados por él deben ser persistentes en el tiempo, sin tiempo máximo.

Acceso

- RFDAC1: el sistema deberá permitir al usuario añadir uno o más números de celular y elegir de esa lista a cuáles se notificará.
- RFDAC2: el sistema permitirá al Usuario acceder a la plataforma web del campo y casilla mediante el ingreso de nombre de usuario y contraseña.

7.4.4 Requerimientos No Funcionales Deseados

- RNFD1: las notificaciones serán enviadas por mensajes SMS o por WhatsApp.
- RNFD2: los datos para reportes se almacenarán en una base de datos de serie de tiempo.
- RNFD3: el sistema deberá ofrecer la descarga de RFDR2 en formato .csv o .xlsx.
- RNFD4: el sistema deberá permitir que mediante la interfaz web de la casilla se pueda reemplazar el archivo de configuración nombrado en RNF8.
- RNFD5: los componentes de la interfaz de usuarios serán desarrollados en forma de librería propia, utilizando *Storybook* para su presentación y *npm* para su administración.
- RNFD6: la gestión de usuarios será independiente para el front-end remoto y los de casilla/campo. Para el primero será administrado por un servicio en la nube, mientras que para el segundo se administrará localmente desde la Raspberry Pi. Aún así, existirá cierta sincronización que permitirá al administrador agregar y eliminar usuarios de casilla/campo remotamente.

Capítulo 8: Diseño del sistema

En este capítulo se mencionan las decisiones de diseño tomadas a lo largo del proyecto. Se incluirá la arquitectura del sistema general, la arquitectura del subsistema correspondiente a este proyecto, las tecnologías seleccionadas y el modelo de datos utilizado.

8.1 Arquitectura General

En el siguiente diagrama se puede apreciar la distribución de los componentes del sistema y cómo están relacionados entre sí. La arquitectura general representa un modelo cliente-servidor[5] en la que se separa el sistema en dos grupos, por un lado están los servidores que se encargan de proveer recursos y servicios (Mapper, servidor campo y servidor remoto), y por otro lado están quienes demandan estos servicios denominados clientes (*frontend* remoto y *frontend* campo y casilla de bombas). La arquitectura propuesta cuenta con dos puntos de acceso distintos y un componente que funciona de puente entre un subsistema y el otro. A su vez, los tres componentes internos del sistema (Mapper, servidor campo y servidor remoto) funcionan por momentos como servidores y en otros como clientes.

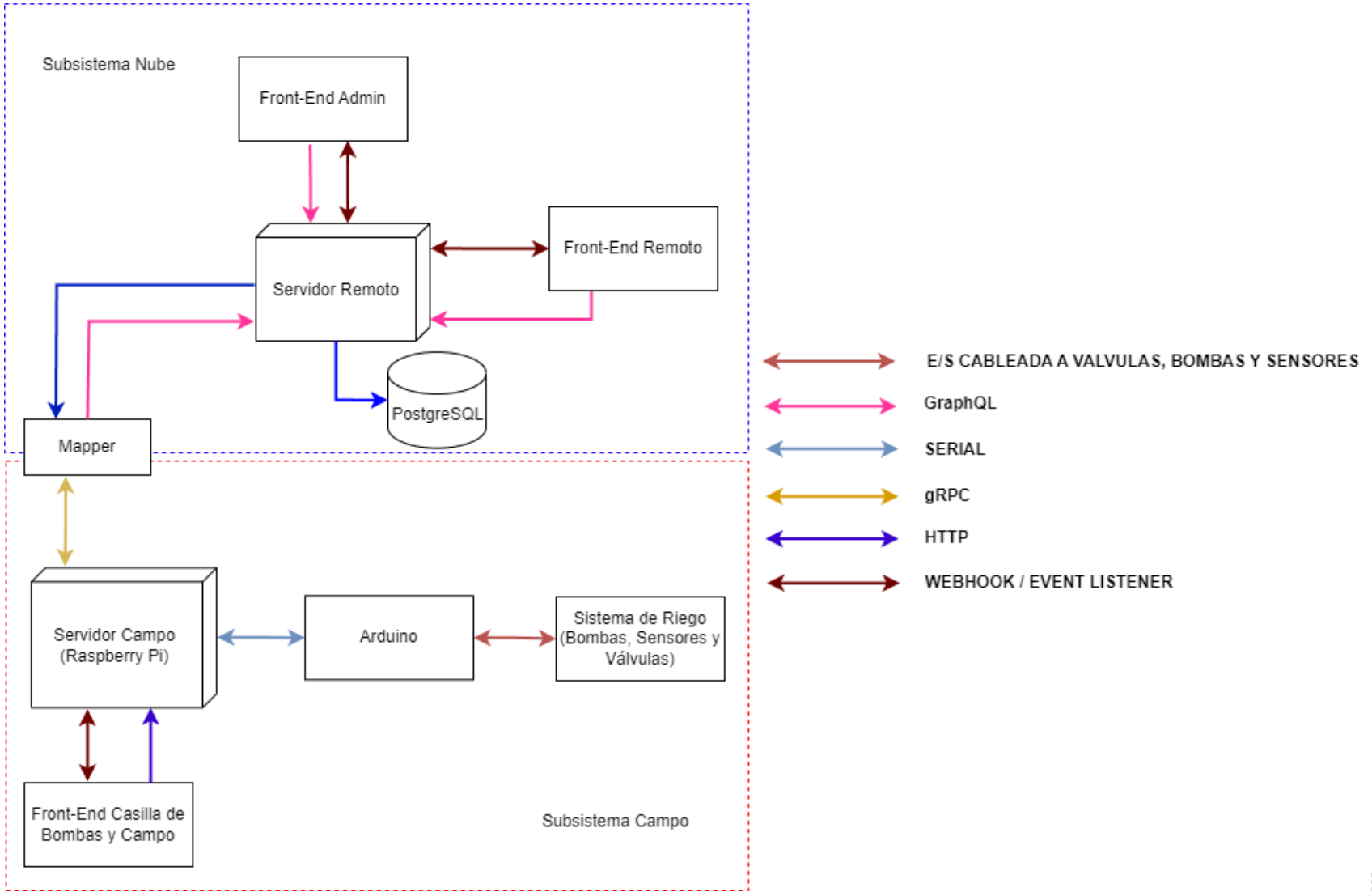

Figura 8 - Arquitectura general Sistema automatizado de riego

Un aspecto a destacar es que el subsistema nube es único, pero el subsistema campo puede estar replicado en cada plantación donde se busque instalar riego automatizado. Primero se hará un resumen del subsistema nube para comprender el sistema general y luego se detallarán los componentes que corresponden a este proyecto.

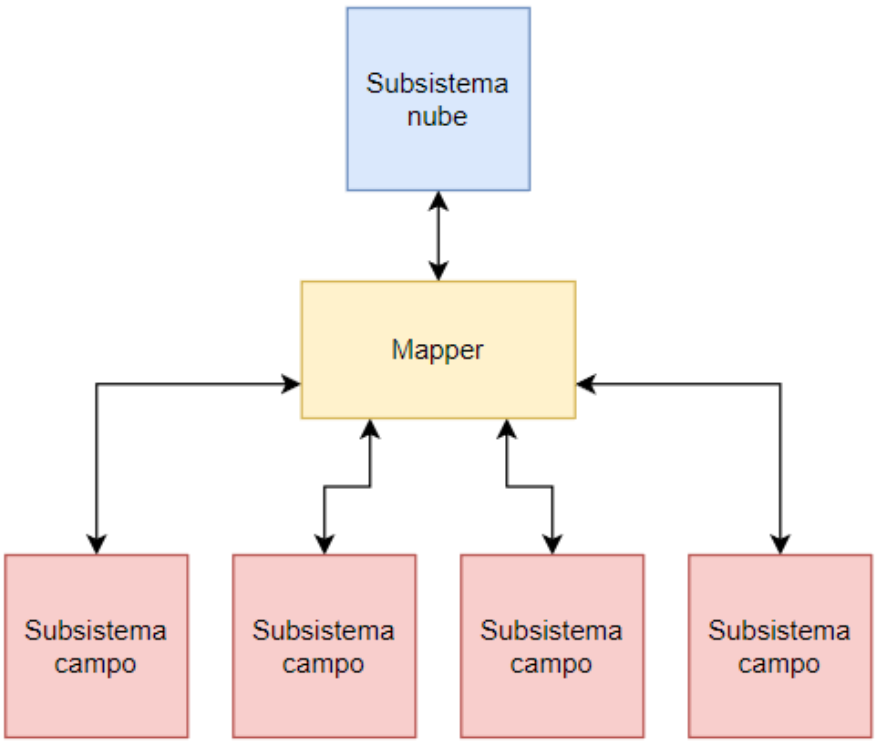

Figura 9 - Vista del subsistema nube interactuando con múltiples Subsistemas campo

Subsistema nube: La totalidad de las acciones están disponibles desde este subsistema, en él se pueden realizar altas, bajas y modificaciones de las entidades y realizar *request* para ejecutar acciones sobre los elementos en campo. Cuenta con un administrador realizado con un sistema de gestión de contenidos (*CMS*), una herramienta online que permite administrar una web de contenidos de forma simplificada. En este caso, también se facilita la interacción con la base de datos PostgreSQL y la comunicación con el Mapper. A su vez, el subsistema permite tener acceso a los *frontend* remotos tanto para usuario como para administrador. También se encarga de la administración del acceso desde los clientes remotos ya que almacena usuarios además de la información del sistema.

8.2 Arquitectura Subsistema Campo

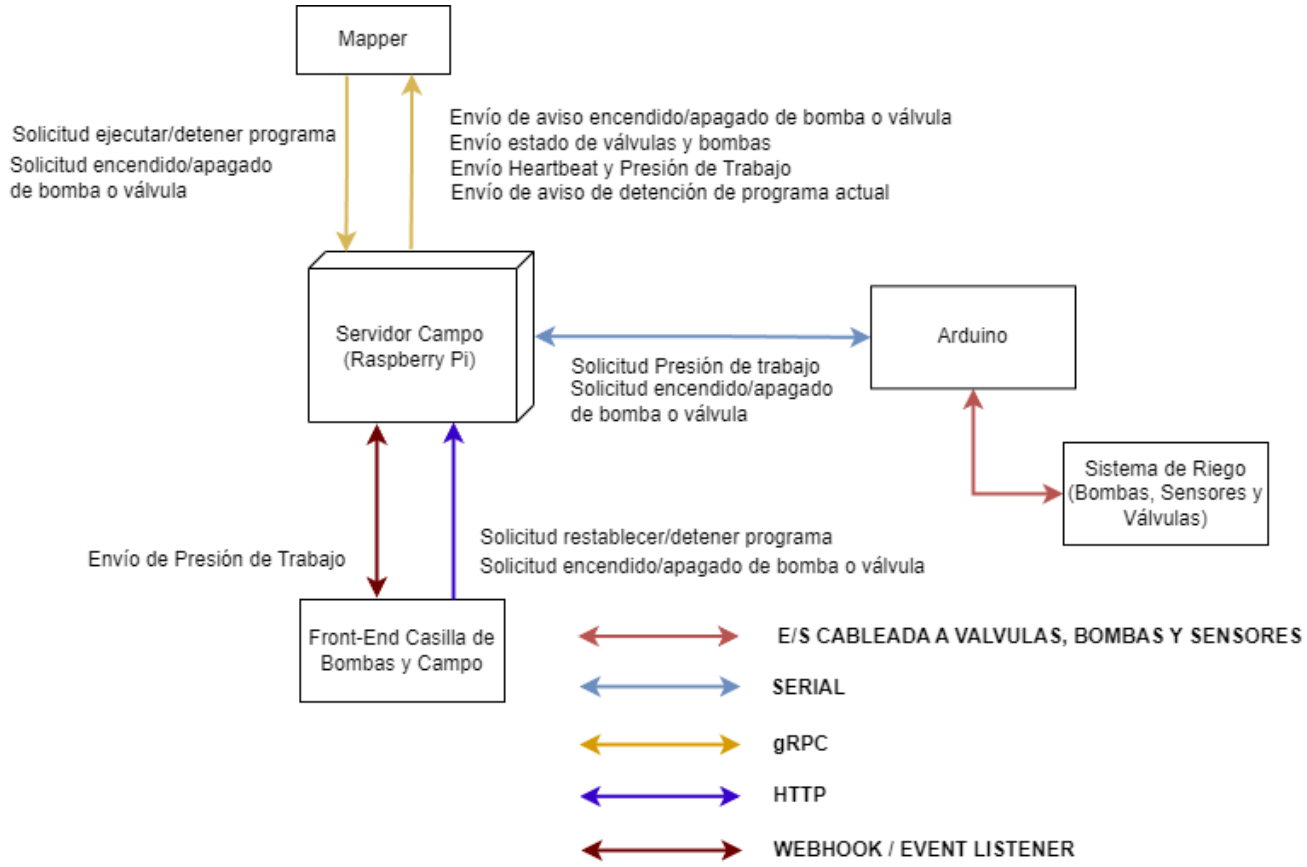

Figura 10 - Vista del Subsistema campo y funcionalidad

8.2.1 Mapper

Toda la comunicación fluye a través de este componente. Existe como una barrera de protección entre ambos subsistemas y a su vez como un traductor para modificar los datos recibidos al formato requerido por la interfaz del servidor remoto y viceversa.

La infraestructura de red en el campo no siempre es la apropiada, es por esto que Ponce AgTech comunica gran parte de sus productos de forma satelital y no se busca dejar al producto de este proyecto fuera. Cuando se hace referencia a este tipo de comunicaciones hay que tener en cuenta que se debe intentar minimizar la

carga. Por esto para la comunicación se eligió gRPC, un protocolo de comunicación liviano que se caracteriza por su baja latencia aun teniendo un flujo elevado de datos y mensajes. A su vez gRPC permite la comunicación entre aplicaciones que utilicen distintos lenguajes de programación, por lo que genera una independencia del lenguaje del cliente y del servidor.

8.2.2 Servidor Campo

Este componente se encarga de la realización de las funcionalidades en el campo, toda la lógica para llevar a cabo las acciones se encuentra almacenada en él. El servidor expone dos interfaces, una REST que recibe los *request* desde el *frontend* local y otra para la comunicación gRPC definida en el archivo ".proto". Por lo que las peticiones pueden llegar desde dos posibles direcciones, a través del *frontend* del campo o desde el Mapper, el servidor las toma y en base a la directiva realiza una acción que tendrá o no consecuencias en los dispositivos del campo según si es una consulta o una modificación.

El servidor almacena un archivo local de inicialización con información de los equipos que administra, que se define manualmente basado en los datos del subsistema nube. La existencia del archivo permite una de las cualidades principales del subsistema campo que es continuar operativo y prestando servicios de forma local a pesar de perder conexión a internet. Todos los cambios que se realizan desde el servidor campo buscan ser comunicados al servidor remoto para que este mantenga constancia del estado de los equipos en su conjunto. Un error al comunicar un cambio desde el entorno local no es un motivo para que el sistema falle y no realice la acción. Dentro de los requerimientos deseados se encuentra añadir una cola de mensajes en busca de comunicar los eventos al servidor remoto cuando se retome la conexión.

El servidor también está a cargo del envío del *heartbeat*, metodología utilizada para informar al servidor remoto que se mantiene activa la conexión con el sistema en campo. Fue elegida debido a su fácil implementación. Desde el inicio del servidor de

campo se intentará realizar el envío del *heartbeat* al subsistema nube de forma periódica para indicar que la conexión se mantiene establecida. Si luego de un cierto período definido en el subsistema remoto no se recibe un *heartbeat*, se supondrá que se desconectó.

La presión de las bombas también es comunicada de forma constante, no solo al servidor remoto sino también a los clientes que están conectados al frontend de la casilla de bombas o el campo, a través de eventos *Server-Sent Events (SSE)* que son enviados desde el servidor a quienes estén conectados a él y se suscribieron (más información puede encontrarse en el apéndice E). Estos permitirán a los usuarios mantener una visión actualizada de la presión de trabajo de las bombas y poder detectar irregularidades.

El servidor está almacenado en un dispositivo Raspberry Pi que se comunica por un puerto serie al microcontrolador Atmel para enviar las acciones a realizar. Este tipo de placas no cuentan con una gran cantidad de memoria por lo que se intentó minimizar el almacenamiento de datos.

8.2.3 Arduino (Microcontrolador Atmega2560)

Los microcontroladores Arduino son sistemas de poca capacidad de almacenamiento y procesamiento, es por esto que la lógica se encuentra en la Raspberry Pi y ellos solo acatan sus órdenes. El programa interno del Arduino cuenta con un *loop* infinito que se ejecuta constantemente mientras esté encendido. Dentro de la iteración se colocó una lectura de entrada y en el caso de que se reciba información a través del *buffer* se definirá qué acción llevar a cabo. Para cada tarea se seleccionó un código de operación que diferencia entre la modificación de estado o la consulta del estado o valor de un dispositivo. Los códigos son los siguientes:

Código	Descripción
101	Obtener estado del elemento
102	Cambiar el estado del elemento entre prendido y apagado
103	Obtener la lectura de presión de un sensor

Figura 11 - Códigos de operación para acciones en microcontrolador

El microcontrolador recibe entre dos o tres parámetros por su *buffer*: el código de operación mencionado anteriormente, la interfaz sobre la que se llevará a cabo la acción (hace referencia al pin al que está conectado el dispositivo a modificar o consultar) y, en caso de ser necesario, el nuevo estado.

Al trabajar con la lectura de presión desde el microcontrolador, es necesario conocer con qué sensor se realizará la lectura ya que no todos lo hacen de la misma manera. En algunos casos se entrega un valor de voltaje que debe ser *mapeado* a hectopascales y en otros cuentan con librerías que pueden añadirse al código de la placa para leer un valor de presión en la medida solicitada y no tener que realizar ningún cálculo.

8.2.4 *Frontend* casilla de bombas y campo

Es la aplicación utilizada por los usuarios del campo para poder realizar acciones de consulta de estado, apagado y encendido de válvulas y bombas y detención del programa actual. El cliente web puede ser accedido a través de la red local del campo y no maneja usuarios. Toda la información expuesta en él corresponde a datos del campo de donde se la esté consultando y no se puede acceder a una visión de otro externo. Las consideraciones de seguridad para este caso son expuestas en el apartado Seguridad.

Este *frontend* no tiene comunicación directa con el servidor remoto, sino que todo el tráfico debe primero atravesar el servidor en campo. La comunicación impacta en la API REST expuesta desde el dispositivo Raspberry Pi y él continúa la propagación del mensaje.

8.3 Tecnologías

A continuación se presentan las distintas tecnologías utilizadas para desarrollar el producto, la mayoría elegidas por Ponce AgTech. Esto se debe a que como el producto entregado es un MVP, la empresa debe continuar con el desarrollo y por lo tanto, debía ser realizado con tecnologías que se utilicen para otros de sus proyectos.

8.3.1 Dispositivos

Dentro de esta categoría se encuentran los dos dispositivos claves para el funcionamiento del sistema automatizado de riego: el microcontrolador Atmel y la placa Raspberry Pi. La utilización de ambos surge como requerimiento no funcional de la empresa demandante para que el sistema se mantenga alineado con su línea de productos. Además, existen una serie de ventajas en el uso de estos elementos, compartidas con Ponce AgTech, por los que se decide su utilización:

- Bajo costo: ambos dispositivos se utilizarán en zonas donde estarán expuestos, en mayor o menor medida, a las inclemencias climáticas que podrían decantar en desperfectos irreparables. Que cuenten con un bajo costo impactará positivamente en el valor del producto y en la reducción de pérdidas ante desperfectos.
- Tamaño: el tamaño reducido de los componentes los hace óptimos para ser colocados en las casillas de bombas sin demandar mucho espacio.

- Sencillos de utilizar: el microcontrolador Arduino cuenta con una gran comunidad detrás y plataformas de estudio que facilitan su aprendizaje y uso, además de un entorno de desarrollo amigable. Por otro lado la Raspberry Pi es similar a una computadora pequeña a la que se le puede colocar un sistema operativo y utilizar como una laptop.

8.3.2 Dispositivos IOT

El Internet de las cosas (IoT)[6] describe una red de objetos físicos que llevan sensores, *software* y otras tecnologías con el fin de conectarse e intercambiar datos con otros dispositivos y sistemas a través de internet. Van desde objetos domésticos comunes hasta herramientas industriales sofisticadas. En este proyecto se hace uso de bombas y válvulas eléctricas y sensores de presión que pueden ser controlados desde el sistema.

Algunas ventajas que se desprenden de este tipo de tecnología son:

- Obtener información basada en los datos de IoT para ayudar a administrar mejor el negocio.
- Aumentar la productividad y eficiencia de las operaciones.
- Crear nuevos modelos de negocio y flujos de ingresos.

Los elementos IoT de este trabajo son el punto de partida para la solución. La implementación utilizada hasta el momento cuenta con sensores de presión, válvulas y bombas controlados por IoT. Estos componentes son muy limitados en términos de tamaño, fuente de alimentación, procesamiento y red; por lo que, están programados usando microcontroladores que tienen capacidades muy reducidas. Los microcontroladores que alimentan dispositivos IoT están especializados para una tarea y están diseñados para la producción en masa y bajo costo.

Dentro de los dispositivos vinculados al IoT podemos distinguir dos tipos:

- Sensores: dispositivos que convierten cantidades físicas en señales eléctricas y que aportan a la obtención de datos, por ejemplo, el medidor de presión.

Figura 12 - Sensor/Transductor de Presión marca Autex

- Actuadores: dispositivos que controlan parámetros externos a partir de órdenes, por ejemplo, motores, válvulas o bombas.

Figura 13 - Electrovalvula marca Rain Bird

Hay tres tipos de interacciones que pueden darse en los dispositivos IoT:

- Interacción máquina-humano, donde se usa un dispositivo IoT para registrar datos en un servidor, que luego se utiliza para mostrar un gráfico que puede ser entendido y utilizado por el usuario final.
- Interacción humano-máquina, donde el usuario está desencadenando un comando a un dispositivo remoto, por ejemplo, para activar una válvula de forma remota.
- Interacción máquina-máquina, donde dos o más dispositivos están hablando directamente entre sí, sin la intervención de ningún ser humano, como cuando se solicita la presión de forma periódica.

8.3.3 Arduino Mega (Microcontrolador ATmega2560)

Figura 14 - Microcontrolador Atmega2560

Arduino[7] es una plataforma de creación de electrónica de código abierto, que está basada en los fundamentos del *hardware* y *software* libre. La placa cuenta con todos los elementos necesarios para conectar periféricos a las entradas y salidas de un microcontrolador, y puede ser programada tanto en Windows como macOS o GNU/Linux.

La placa utiliza un microcontrolador Atmel en el que se pueden grabar instrucciones utilizando el entorno Arduino IDE que permite la codificación del programa y posterior grabado en ella. La mayoría están alimentadas por una entrada USB o *Jack* de 2.5 mm y son programadas a través del puerto serie. Si un programa nuevo

llega, el *Bootloader* se encarga de ejecutarlo y sino, ejecutará el que esté cargado desde antes. Cada una cuenta con una cantidad de puertos Entrada/Salida de Propósito General (*gpio*) que pueden ser digitales o analógicos para la conexión con otros circuitos. Existe una gran variedad de tipos de placas Arduino, la utilizada en este trabajo es la Arduino Mega (microcontrolador ATmega2560[8]) elegida por la amplia cantidad de pines *gpio* con los que cuenta, 40 más que su versión más simple (Arduino UNO), y el doble de entradas y salidas analógicas.

Figura 15 - Shields utilizados por Ponce AgTech en el sistema actual sobre el microcontrolador Arduino conectados a placa de relés

Al microcontrolador a su vez se pueden conectar módulos adicionales llamados *shields*. Estos dispositivos aumentan las características técnicas de la placa base, debido a que tienen circuitos propios que añaden funcionalidades. Los dispositivos Arduino utilizados por Ponce AgTech utilizan *shields* personalizados que cuentan con relés encargados de permitir o no el paso de corriente y así manejar el estado de los dispositivos del campo.

8.3.4 Raspberry Pi

El servidor campo se aloja en una placa Raspberry Pi como requerimiento no funcional de Ponce AgTech, ya que el sistema utilizado hasta el momento lo hace de esta forma.

Figura 16 - Raspberry Pi B 3+

Para este sistema se utilizó una placa Raspberry Pi B 3+ que ofrece una mayor cantidad de beneficios que su versión anterior como mejor conexión a internet. En esta aplicación se vuelve fundamental contar con este beneficio. Además, para este sistema no se justifica el gasto ni la utilización de una computadora de mayor capacidad que la ofrecida por la Raspberry Pi.

Una gran ventaja de la similitud que tienen con una computadora es la posibilidad de agregar un sistema operativo. En este caso se utilizó Raspbian, una distribución de código abierto del sistema operativo GNU/Linux basado en Debian. Este se graba en una tarjeta SD que luego se incorpora a la Raspberry Pi. A la placa también, se pueden conectar pantallas y periféricos que facilitan la interacción.

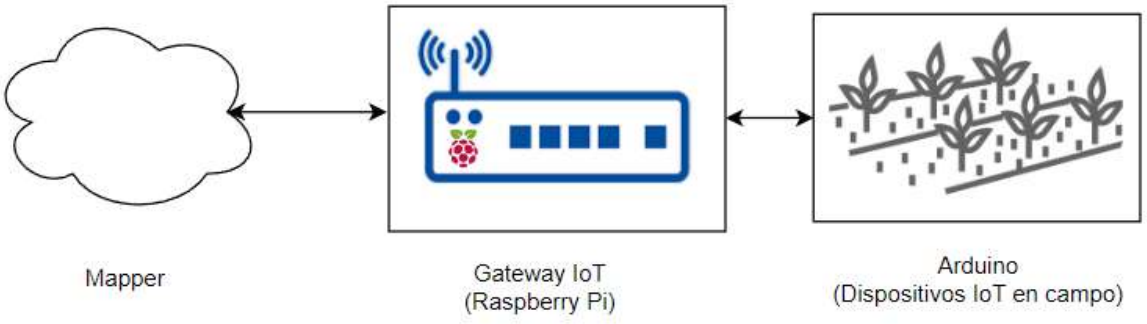

Figura 17 - Disposición de la Raspberry Pi como Gateway

Dentro de un sistema con IoT se debe contar con un *Gateway* IoT[9] que será en vínculo entre los dispositivos inteligentes que se encuentren en la red local y la nube. En este caso el dispositivo Raspberry Pi oficiará de *Gateway*. Este elemento ofrece el procesamiento de datos “en el borde” (realiza un preprocesado de los datos antes de enviarlos a la nube) y capacidad de almacenamientos para hacer frente a la latencia de la red y fiabilidad. Debe lidiar con los problemas de interoperabilidad que puedan surgir por la incompatibilidad de dispositivos. Además, ofrece una primera línea de defensa para estos elemento IoT potencialmente inseguros frente a internet. Todos deberán pasar por él para comunicarse con el sistema en la nube, lo que lo convierte en una especie de *proxy*.

8.4 Lenguajes de programación

El lenguaje de programación elegido y usado ha sido JavaScript, utilizado por Ponce AgTech y por los integrantes del proyecto en la mayoría de sus actividades profesionales. Debido a esto, no se dudó al momento de elegir NodeJS como el entorno para desarrollar tanto los *backends* como los *frontends*. Además de las ventajas en cuanto a la experiencia sobre el lenguaje, según un artículo de la web de la Universidad Berkeley de California, JavaScript es el lenguaje más comúnmente usado mundialmente en 2022[10]. Esto significa que aparte de que es soportado por una gran cantidad de plataformas, existen muchas comunidades de

ayuda y documentación en la mayoría de idiomas y librerías para todo tipo de usos. Esto último implica que, aunque originalmente JavaScript nació orientado al desarrollo web, actualmente se usa como lenguaje en el lado servidor en proyectos de gran escala. Esto permitió aplicarlo en el *frontend* y en el *backend*, lo que implicó una mayor facilidad para el desarrollo.

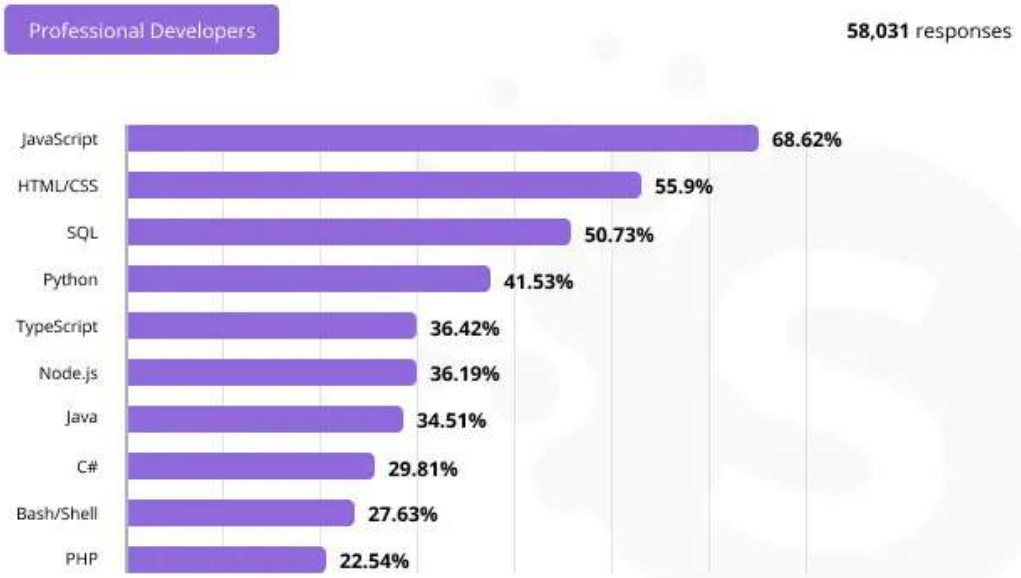

Figura 18 - Resultados de la encuesta de Stack Overflow en 2021 sobre lenguajes de programación más usados por profesionales. Fuente: https://www.softermii.com/

Por último, para el microcontrolador se ha utilizado el lenguaje Processing (similar a C++) impuesto y estandarizado para el entorno de desarrollo que se compila a un lenguaje máquina específico compatible con la placa que se esté utilizando. Además, incluye algunas funciones y variables especiales para controlar el dispositivo, escritas en C++. Sin embargo, debido a la reducida memoria *RAM* del Arduino Mega (8KB), utilizado para este proyecto, existen librerías de C++ que no funcionan correctamente en la plataforma. En cualquier caso, hay una gran cantidad de documentación sobre esta tecnología debido a que es ampliamente utilizada para proyectos académicos y didácticos en general, sumado al bajo coste que tiene y que la empresa está asociada a grandes compañías como Google, Intel, Raspberry Pi, entre otras.

8.4.1 *Frontend*

Originalmente, Ponce AgTech había decidido utilizar el *framework* de *frontend* Vue.JS para el desarrollo de las interfaces gráficas de los dos subsistemas, dado que es la que se utiliza en el resto de proyectos de la empresa. Esto representaba una oportunidad de aprendizaje para los integrantes, ya que ninguno contaba con experiencia en esta tecnología. Sin embargo, al iniciar el desarrollo se notó que la librería de componentes visuales a utilizar, Vuetify, no era compatible con Vue 3. Esto implicaba tener que realizar los *frontend* en una versión anterior de Vue.JS, perdiendo algunas funcionalidades de la nueva y teniendo en cuenta que a futuro debía ser actualizada. Por otro lado, en el subsistema remoto no era posible utilizar Keystone JS, que dispone de una interfaz en React y no contaba con una versión en Vue 3. Por estos motivos, y para mantener consistencia en las tecnologías utilizadas por los dos subsistemas, se decidió utilizar React, *framework* en el cual todos los integrantes tenían experiencia previa, además de que cuenta con mayor cantidad de documentación y recursos en general que Vue, debido a su amplio uso mundialmente y la financiación con la que cuenta (su desarrollador es Meta Platforms Inc.). El demandante aceptó esta decisión y se determinó usar Ionic como librería de componentes, ya que los integrantes la habían utilizado anteriormente para el producto desarrollado en las Prácticas Profesionales Supervisadas de la universidad y el demandante también la había aplicado en algunos proyectos.

8.4.2 *Backend*

De forma de obtener un *backend* sencillo y escalable, se decidió utilizar el *framework* de código abierto para NodeJS llamado Express.JS, siendo el más popular existente del ecosistema de JavaScript y ya utilizado por los integrantes en distintos proyectos personales y laborales. La herramienta dispone de un conjunto de facilidades para comunicarse con aplicaciones web, facilita el enrutamiento y la creación de APIs *RESTful*, es decir, que solo utilizan métodos HTTP como GET y POST. De este modo, la posibilidad de incorporar nuevas funcionalidades y utilizar

middlewares en el futuro resulta más sencillo. Otro aspecto a destacar gracias a su simplicidad es la gran escalabilidad que ofrece al no requerir de grandes codificaciones para habilitar un nuevo *endpoint*.

Para el *backend* en campo se agregó un *middleware* llamado Morgan. Esta librería es ampliamente utilizada en conjunto con Express.JS para obtener información sobre los *request* que llegan a la API. En este proyecto se la integró con la librería seleccionada para auditoría llamada Winston, de forma que todas las peticiones sean almacenadas en el archivo de *logging* del sistema.

8.5 Comunicación (protocolos e interfaces)

8.5.1 REST

Desde el servidor en campo se decidió no agregar GraphQL, un protocolo enfocado en consultas que define esquemas para la comunicación entre partes con el fin de optimizar tiempos de proceso. La decisión se tomó teniendo en cuenta las funcionalidades reducidas que ofrece el *frontend* frente a la complejidad y esfuerzo de agregarlo. Por lo que como alternativa se añadió un servidor HTTP en el *backend* de campo que expone los *endpoints* REST necesarios para las funcionalidades, como obtener datos sobre los dispositivos en campo, modificar su estado y detener o restablecer un programa. Se buscó separar los *endpoints* de forma que sean claros para su desarrollo futuro y que cada uno tenga una responsabilidad concreta dentro del sistema.

8.5.2 SSE (*Server-Sent Events*)

Como uno de los requerimientos funcionales surgió la posibilidad de obtener una actualización en tiempo real de los valores de presión que se miden desde el campo, así como el estado de conectividad. En base a esto se definió la utilización de SSE, que a diferencia de WebSocket que es bi-direccional, se enfoca en enviar eventos desde el servidor al cliente para mantenerlo actualizado con los datos que le llegan. Desde el *backend* se expone un *endpoint* que los clientes utilizarán para suscribirse a los mensajes. Solicitan una conexión usando HTTP como protocolo de comunicación, una vez aceptada se abre un canal que permitirá al servidor enviar mensajes. El canal se mantendrá abierto hasta que se decida cerrar la conexión. Si el servidor es quien cierra el canal, desde el cliente se verá como un error e inmediatamente se buscará abrir uno nuevo.

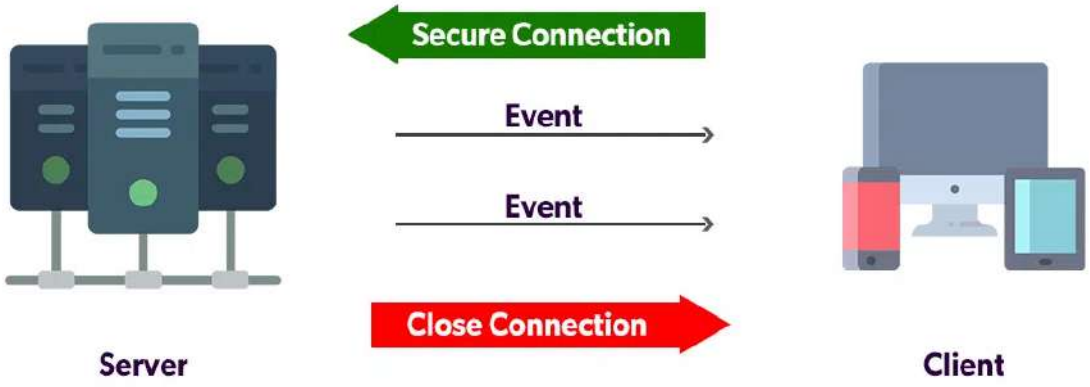

Figura 19 - Flujo de comunicación mediante SSE

Un aspecto a destacar de la implementación realizada es que no se necesitó hacer uso de módulos externos para aplicar la funcionalidad, solo se hace uso del *framework* Express utilizado para la creación del servidor en campo. Cada mensaje que se envía se encuentra en texto plano, pero define un tipo para poder diferenciarlo de los demás. Más información sobre el formato de los mensajes enviados se encuentra en el apéndice C.

8.5.3 Protocol Buffer

Desde la empresa demandante se hizo foco en la utilización de internet satelital debido a la falta de infraestructura de red presente en varios de los campos. Con esto en mente se definió la utilización de Protocol Buffer (también llamado Protobuf)[11]. Este es un formato de intercambio de datos desarrollado por Google para uso interno y que fue lanzado al público en 2008 como un proyecto de código abierto. Este formato permite a las aplicaciones intercambiar y almacenar datos estructurados fácilmente, incluso si los programas están compilados en lenguajes diferentes. Este protocolo se puede combinar tanto con HTTP como Remote Procedure Call (RPC) para llevar a cabo la comunicación cliente-servidor local y remota.

Se seleccionó Protocol Buffer como formato por los siguientes motivos:

- Las estructuras se mantienen de ambos lados de la comunicación.
- Los campos de datos son numerados por lo que se genera una compatibilidad entre versiones y no es necesario adaptar el código.
- En el protocolo se permite definir si el campo es obligatorio, opcional o repetido.
- Gracias a la traducción realizada, no es necesario agregar esfuerzo para realizar una comunicación entre aplicaciones que utilicen distinto lenguaje.
- Es entre tres y diez veces más pequeño y entre 20 y 100 veces más rápido que una estructura XML, según datos de Google, por lo que es óptimo para aplicaciones que requieren minimizar el consumo.

Hay dos aspectos claves en los motivos del uso de Protocol Buffer [12]. Primero, este protocolo funciona como un adaptador entre dos sistemas que podrían tener lenguajes diferentes (en este caso entre el subsistema campo y el Mapper) permitiendo que en un futuro se pueda realizar una implementación de cualquiera de los dos en otro lenguaje y continuaría funcionando correctamente. Segundo,

se adapta perfecto a la utilización de una comunicación de bajos recursos como es la satelital ya que minimiza el consumo de red en gran medida.

Para la implementación de este protocolo, tanto en la aplicación cliente como en el servidor, debe existir el archivo ".proto" correspondiente que almacenará la estructura de datos. Dentro del archivo se definirá la versión de Protocol Buffer utilizada, un nombre asignado al paquete que sea descriptivo de lo que almacena y por último los *messages* o mensajes que guarda la estructura de datos.

```
message ProgramStepsDef {
  string startTime = 1;
  int64 duration = 2;
  string deviceType = 3;
  string deviceId = 4;
}

message ProgramDataDef {
  string id = 1;
  string name = 2;
  string startTime = 3;
  repeated ProgramStepsDef steps = 4;
}
```

Figura 20 - Ejemplo de estructura de datos en Protocol Buffer tomada del producto

8.5.4 gRPC

En base a la utilización de Protocol Buffer como protocolo de intercambio de datos, se definió utilizar gRPC [13] para la comunicación remota entre el Mapper y el servidor en campo. Cuenta con una extensa documentación de su empresa creadora, Google. RPC es una llamada a procedimiento remoto, es decir, un programa que ejecuta código de forma remota en otro pero simulando una llamada local. Es muy utilizado en comunicaciones cliente-servidor donde el cliente inicia el proceso solicitando al servidor que ejecute un procedimiento y

luego de un tiempo recibe el resultado de dicha operación. gRPC está basado en RPC, pero presenta ciertos cambios. Por un lado, deja el uso de HTTP/1.1 para hacer uso de HTTP/2 que permite multiplexación (en una sola conexión TCP se agrupan varias señales que se pueden transmitir de manera simultánea por la red), y por el otro, utiliza Protocol Buffers para gestionar la estructura de los datos. Esta metodología no está alejada de un RPC común pero permite a su vez todas las ventajas del uso de Protocol Buffer como lo es realizar comunicaciones entre aplicaciones con distintos lenguajes. Además, brinda ventajas en la realización de *streaming* y se ajusta muy bien para casos de uso que requieren minimizar el ancho de banda utilizado [14].

```
service RaspberryController {
  rpc ChangeElementState(ChangeElementStateRequest) returns (ChangeElementStateReply) {}
  rpc SetProgram(SetProgramRequest) returns (SetProgramReply) {}
  rpc OverrideElement(OverrideElementRequest) returns (OverrideElementReply) {}
}
```

Figura 21 - Ejemplo de definición de servicios en archivo .proto

La interfaz de un servicio gRPC se define en el archivo “.proto” al colocar un nombre y la estructura de datos que será enviada y recibida. Cabe destacar que para la interacción debe existir un cliente que realice el pedido y un servidor que responda, cada uno debe utilizar gRPC y contar con el mismo archivo “.proto” que defina la interfaz. La interacción se da de la siguiente manera: en primer lugar el cliente genera el *stub*, es decir, el módulo que hace que la llamada remota sea vista como una local para la aplicación. Luego se realiza la llamada a la función correspondiente del *stub* y se envía la consulta. Cuando el servidor la recibe lleva a cabo el procedimiento y devuelve la respuesta al *stub* del cliente que, a su vez, transmite el resultado a la instancia original.

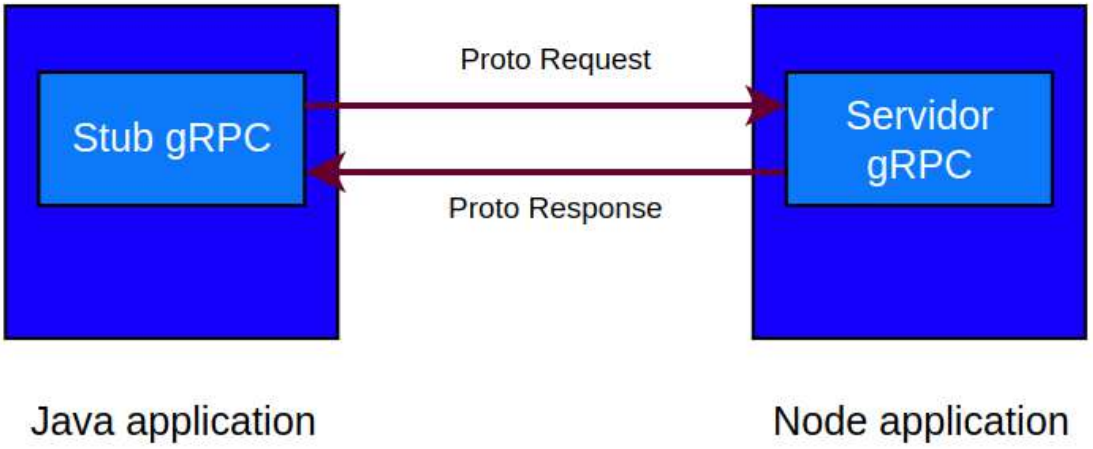

Figura 22 - Comunicación gRPC

8.6 *Testing*

La plataforma elegida para escribir y ejecutar *tests* ha sido Jest, una popular librería creada y mantenida por Meta para proyectos y aplicaciones web en JavaScript. Se ha elegido debido a su simplicidad, cantidad de documentación disponible y la facilidad que brinda para simular APIs y métodos ajenos a la funcionalidad que se quiere *testear*, situación ampliamente presente en este proyecto. Además, los integrantes ya contaban con experiencia sobre esta librería al igual que el equipo de Ponce AgTech, responsable a futuro de aumentar la cantidad de *tests* disponibles.

8.7 Modelo de datos

Luego de realizar el relevamiento, se decidió colaborar entre ambos equipos en la creación de un modelo de datos validado por la empresa demandante. Este modelo facilitó la definición de la comunicación de la aplicación y al equipo del subsistema nube en la creación de las estructuras en la base de datos, ya que en él se plantearon los campos correspondientes a cada entidad, nombre, tipo y la relación de las entidades con las demás. El resultado final fue el siguiente Diagrama Entidad-Relación.

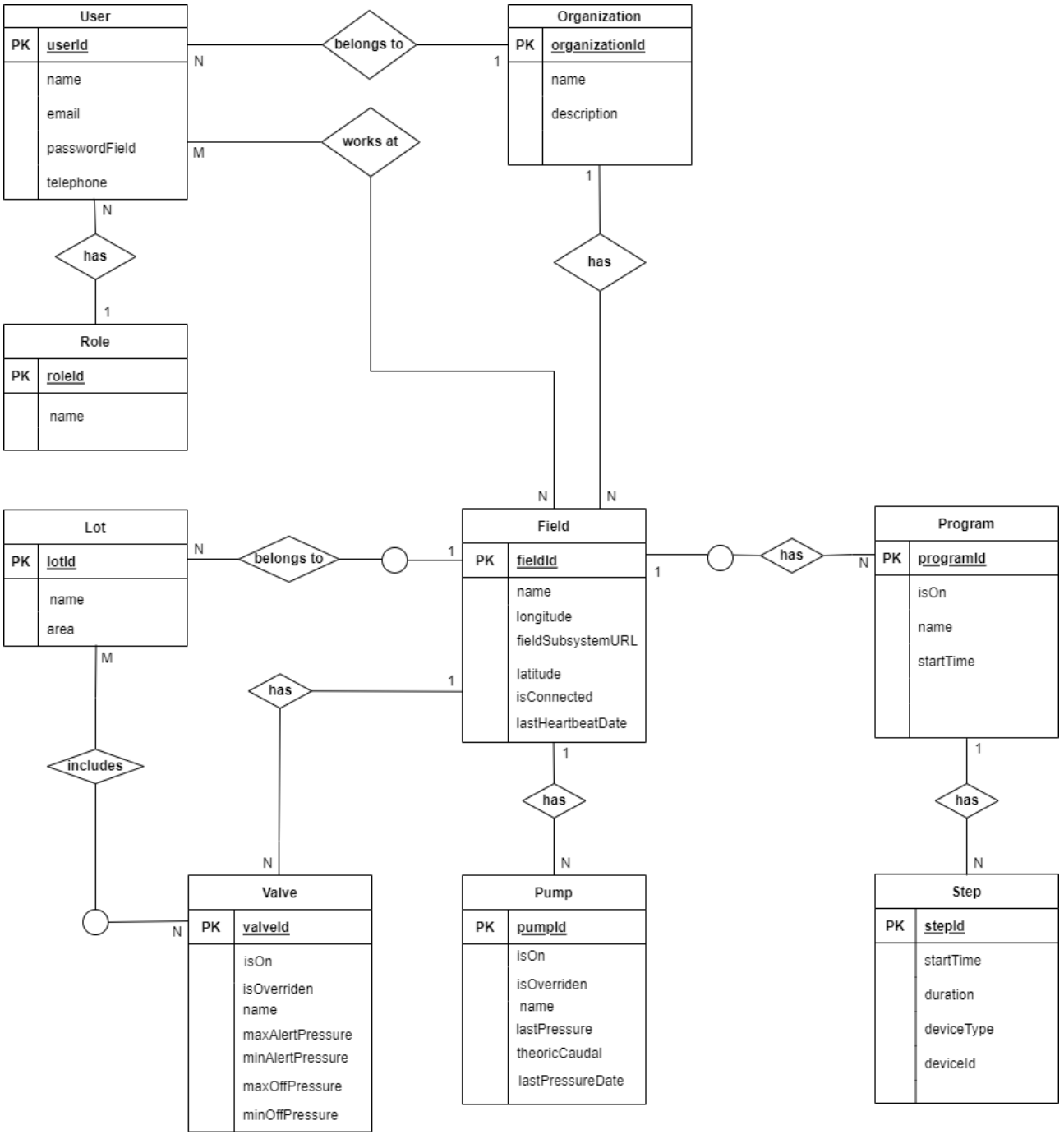

Figura 23 - Diagrama entidad-relación

Capítulo 9: Producto

9.1 Producto obtenido

A continuación se presentan las principales características del producto desarrollado, el cual consiste en una aplicación web *responsive* que se ejecuta en un navegador. Que sea *responsive* quiere decir que puede visualizarse correctamente en varios dispositivos con diferentes resoluciones, ya sean celulares, tablets o computadoras, siempre mediante navegadores. La aplicación permite a los usuarios gestionar los dispositivos de riego en el sistema, es decir, el control y visualización del estado de las válvulas, bombas y programa de riego cargado.

Al no existir manejo de usuarios, una vez ingresado al sistema se pueden acceder a las tres partes de la aplicación: la lista de bombas, la de válvulas y el control del programa actual. En el caso de las dos primeras, se presentan unas tarjetas que le permiten realizar al usuario acciones básicas como ver el nombre del elemento, su código de identificación único, prenderla o apagarla y en el caso de que se encuentre dentro de un programa, forzarla.

Forzarla u *override* es la acción que se puede realizar cuando el elemento está dentro de un programa y se desea que no responda más a los pasos en los que se involucra. Mientras se encuentre *overridden* no se seguirán los pasos pero no se detendrá el paso actual si ya comenzó e involucraba al elemento en cuestión. El forzado se suele realizar en caso de tener que hacer algún trabajo puntual sobre un elemento del campo, ya sea reparar una falla, como una rotura física o una desconexión.

Por otro lado, al funcionar *offline*, no tener una conexión garantizada con el *backend* remoto y no contar con una base de datos local para almacenar metadatos relacionados a elementos del sistema, se cuenta con un archivo de configuración inicial. Se debe popular antes de iniciar el subsistema y contiene información que no puede ser modificada en tiempo de ejecución, como los identificadores de las

bombas, válvulas y sensores y su correspondiente pin en el microcontrolador, nombres opcionales sobre elementos del sistema (para ser mostrados en el *frontend*), entre otros.

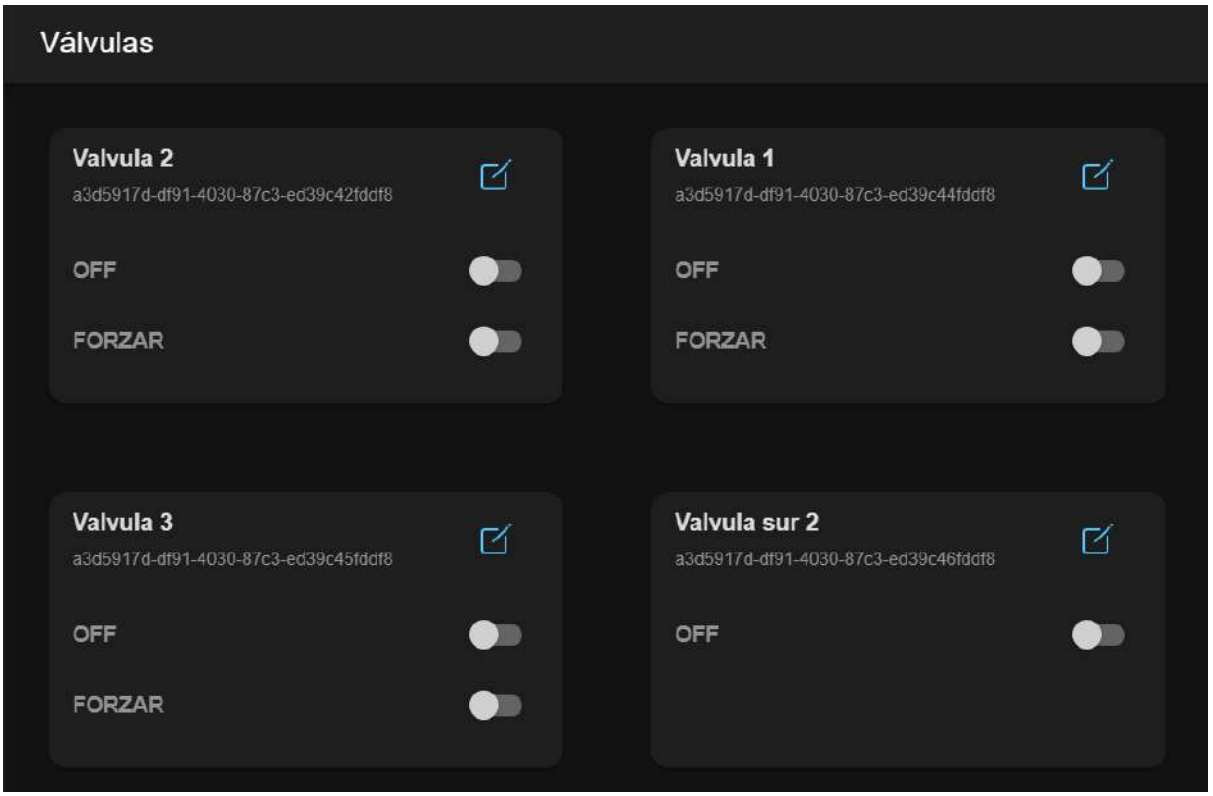

Figura 24 - Listado de válvulas

En el caso de que el elemento sea una bomba se cuenta con la información de presión brindada por el sensor y actualizada en vivo cada vez que se recibe una actualización.

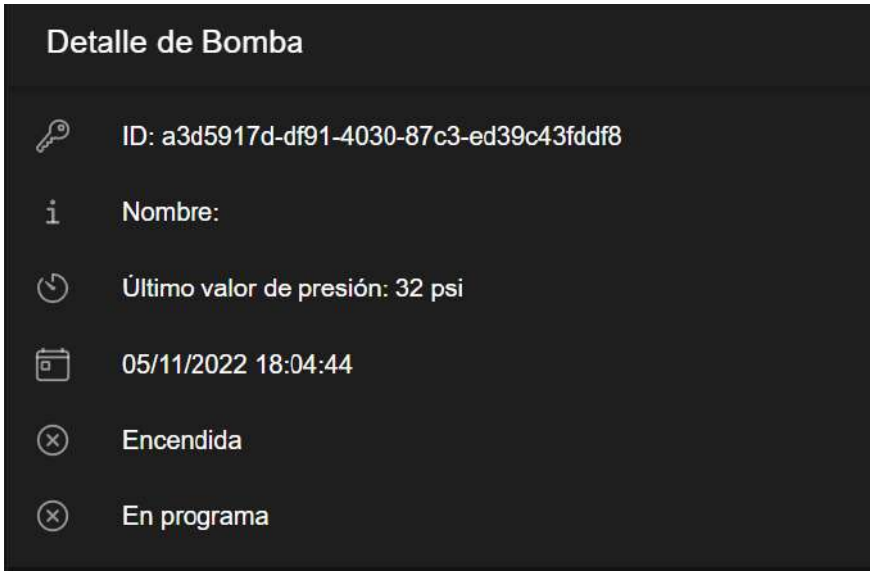

Figura 25 - Valor en vivo de presión

Por otro lado, con respecto a la gestión del programa actual se presentan dos partes. En la primera se puede ver información básica sobre el que se encuentre cargado (recibido anteriormente desde el *backend* remoto mediante el Mapper).

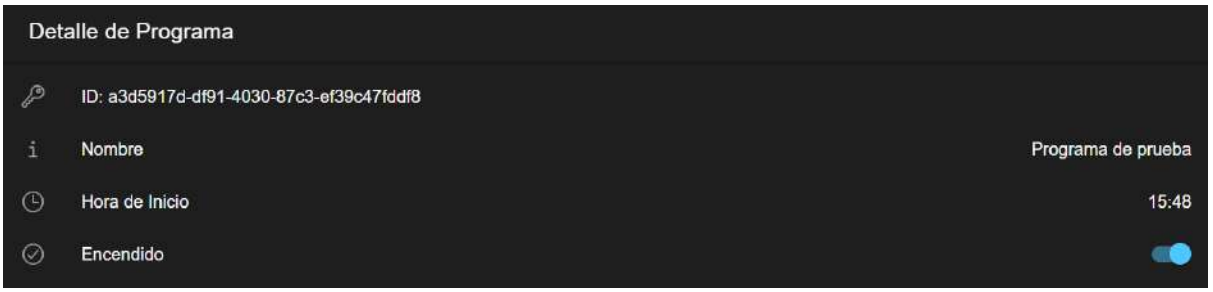

Figura 26 - Detalle de programa actual

En la segunda parte, se puede ver un componente que lista los pasos, con toda la información disponible para cada uno.

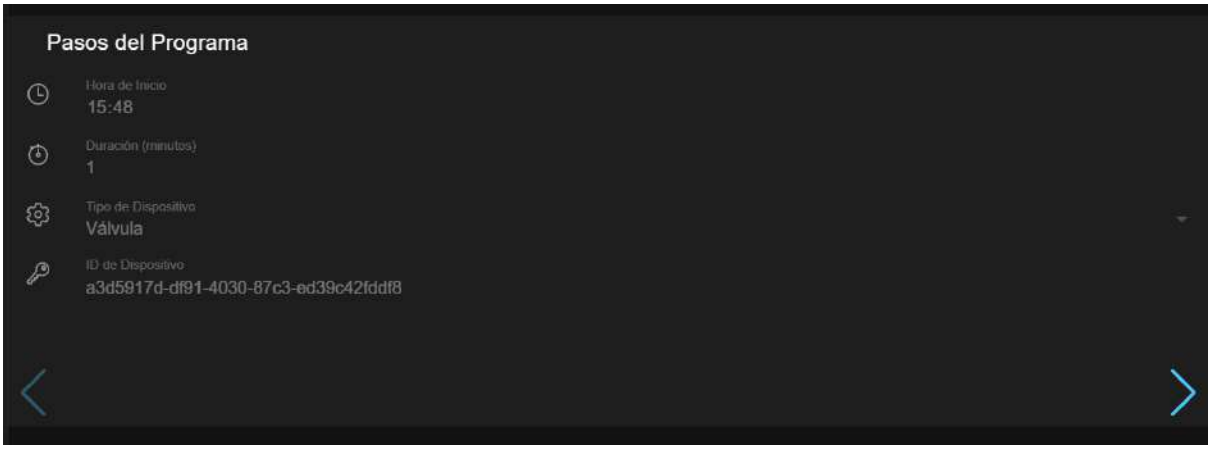

Figura 27 - Pasos del programa

9.2 Comparación con producto predecesor

Característica	Producto predecesor	Producto nuevo
Experiencia de usuario	Interfaz con bugs y funcionalidades no disponibles que figuran como opciones (como la emisión de reportes y gráficos).	Interfaces de usuario aptas para todo tipo de dispositivos (*responsives)* adaptables a pantallas de celulares y tablets. Interfaz más simple.
Datos en tiempo real	Para poder acceder a los datos más actualizados se los debe solicitar de forma manual.	Cada cierto período de tiempo definido en el sistema, se reciben actualizaciones del estado del campo.
Gestión centralizada de campos	Interfaz web exclusiva para cada campo.	Cada usuario puede administrar desde una única cuenta todos los campos a los que tenga acceso.
Mantenibilidad	Imposibilidad de mantener *frontend* debido a ausencia de código fuente. Código poco mantenible y ausencia de documentación.	Documentación correspondiente a funcionalidades generales, arquitectura y a las API. Código de microcontrolador simple y mantenible.
Cambio de autonomía sobre microcontroladores	Alta responsabilidad y lógica dentro del código de la placa Arduino. Desde Ponce AgTech se expresó que esto impide continuar su desarrollo y limita la funcionalidad a la capacidad del microcontrolador.	La lógica se encuentra dentro del servidor alojado en la placa Raspberry Pi y el microcontrolador solo funciona como un esclavo que acata sus órdenes. Esto brinda una mayor potencia y facilidad para el desarrollo futuro.

Figura 28 - Comparación con producto predecesor

9.3 *Benchmarking*

Al tratarse de un mercado reducido, se ha determinado que las principales empresas competidoras son dos: Hunter Industries y Rain Bird, las cuales se encuentran en Estados Unidos y ninguna se dedica exclusivamente al riego para kiwis ni similares, a diferencia de Ponce AgTech.

9.3.1 Empresas competidoras

Hunter Industries

Hunter[15] es una compañía familiar fundada en 1981, que se encarga de varios sectores como el riego de campos y grandes extensiones de tierras, iluminación y fabricación de tecnologías de dispensación relacionadas con la higiene en general. En 2016 Hunter compra a la compañía Senninger[16], que se encarga desde ese año de la fabricación de dispositivos de riego para aplicaciones agrícolas, hortícolas e industriales. Se encuentra en más de 50 países de todo el mundo, incluido Argentina, donde su distribuidor es EQUIPO MUNDITOL S.A. en todo el país.

Además de proveer los equipos mecánicos de riego, Senninger provee un *software* llamado Centralus, que sirve para manipular equipos de riego en general mediante una aplicación web. Se anuncia que posee varias características, como el acceso seguro por la nube, interfaz *responsive*, mensajes SMS de alerta ante eventos, control de agua en caso de lluvias, mapa interactivo para acceso rápido, entre otros. El producto de *software* solo funciona como un módulo que se integra únicamente con dos controladores también propiedad de Hunter, ACC2 e ICC2, y permite que se controlen remotamente con las características antes mencionadas. Por este motivo, este producto solo puede funcionar con conexión a internet, ya sea mediante red 4G, Wi-Fi (Wi-Fi 802.11 b/g/n) o Ethernet (RJ-45). Los kits de conexión a Internet Wi-Fi y Ethernet rondan los 100 dólares de costo (al 01/10/2022), mientras que el de 4G cuenta con valores de aproximadamente 10 dólares por mes y 100 por año (al

01/10/2022). No se cuenta el costo de instalación ya que en la web de Hunter se anuncia que los kits pueden ser fácilmente instalados por los mismos usuarios. Si bien estos son los costos de comunicar el controlador con Internet, se debe considerar que el uso de Centralus es gratis registrándose en la web de Hunter, pero se debe sumar el valor adicional de poder utilizar únicamente dispositivos propietarios de Hunter, que implican un costo mínimo de 1300 dólares, sumado a aquellos adicionales como instalación, entre otros.

Figura 29 - Centralus de Hunter

Rain Bird

Rain Bird Corporation es una empresa estadounidense fundada en 1933. Opera en 130 países del mundo, ofrece 4000 productos(17) distintos y se centra principalmente en productos de riego para plantaciones, campos de golf, hogares e instalaciones deportivas. Hacen hincapié en el ahorro de agua, mediante la premisa “uso inteligente del agua”. Su mercado está orientado principalmente a los aspectos físicos del riego, tales como cañerías, válvulas, mangueras, pinzas para soportes, entre otros. Sin embargo, al igual que Hunter también cuentan con *software* para la administración de dispositivos de riego. En la web de Rain Bird se presentan varios

programas, la mayoría para administración de equipos de riego para campos de golf. Los *software* de Rain Bird[18] son específicos para algunos controladores de la marca, por lo que según cuál se tenga se utilizara uno u otro. En este sentido, también varían las plataformas donde se pueden ejecutar, ya que algunos programas son exclusivos para Windows, mientras que otros son web o *mobile*. La mayoría son gratis aunque se debe contar con controladores de la marca, que tienen una amplia variedad de precios.

El distribuidor en Argentina de los productos Rain Bird es AGUAS S.R.L.[19], anunciado en su propia página. En su web se anuncia la venta de cañerías y mangueras de Rain Bird, pero no se nombran a los controladores, por lo que se desconoce si pueden proveerlos localmente.

Para obtener más información sobre los productos ofrecidos, se decidió contactar directamente con AGUAS S.R.L., los cuales dieron una respuesta inmediata y se pusieron a disposición para contribuir con información al proyecto final a través del Ingeniero Agrónomo Esteban Juliano de la división de Riego y Conducción de Fluidos de la empresa. Se mantuvo una conversación telefónica donde el Ingeniero Juliano explicó que su empresa está dedicada especialmente a distribuir aquellos productos de Rain Bird orientados a controlar pocas estaciones, es decir, baja cantidad de válvulas, utilizados en riego doméstico y de extensiones de tierra usadas para deportes. Por otro lado, especificó que los productos para riego de campo, incluidos los de microaspersión y goteo, pueden ser vendidos si así se requiriera, pero no se cuenta con ningún ingeniero especializado en ellos. Además, relató que el *software* que brinda Rain Bird para estos casos es IQ, siendo de descarga gratuita desde la web oficial contando con una controladora instalada propietaria de la empresa. El programa en cuestión, dictaminó el Ingeniero Juliano, cuenta con funciones avanzadas para el manejo del riego en un campo, que van más allá de las funciones básicas climáticas, y de manejos de programas de riego dinámicas. Adicionalmente, también es posible utilizarlo *offline*, de forma similar al producto de este trabajo final.

Figura 30 - IQ de Rain Bird

9.3.2 Comparación

Para realizar la comparación de los tres productos de *software*, se tomarán seis aspectos: el producto, las plataformas donde se ejecutan, la infraestructura, el soporte, la conectividad necesaria para utilizarlo y el costo para el usuario.

- Producto: tanto los productos de Hunter como los de Rain Bird, no son específicos para el riego por microaspersión y goteo. De hecho, están orientados a la agricultura en general, por lo que tanto actualmente como a futuro no se espera que contengan funcionalidades ni facilidades para el tipo de riego objetivo. Se puede concluir que el *software* de Rain Bird es superior al de Ponce AgTech en prestaciones, pero no puede considerarse como un problema actual teniendo en cuenta que el desarrollado en este trabajo final es un *MVP* y no un producto final. Se tiene en cuenta además que Ponce AgTech planea agregar funcionalidad en un futuro cercano, por lo que eventualmente podría competir en ese aspecto.

- Plataformas: los productos de Hunter, Rain Bird y Ponce AgTech pueden utilizarse en varias plataformas ya que son aplicaciones web *responsives*, aunque en el caso de Rain Bird también existe una aplicación *mobile* para Android y iOS con la funcionalidad completa del *software*.

- Infraestructura: el producto de Ponce AgTech puede usarse sobre cualquier infraestructura de riego por microaspersión y goteo, mientras que los otros dos productos solo pueden usarse bajo controladores propios de las marcas.

- Soporte: tanto Hunter como Rain Bird tienen representación en el país únicamente mediante sus distribuidores, localizados en Buenos Aires. En el caso de un problema, no se tiene información que estos proveedores cuenten con asistencia en Mar del Plata y la zona, a diferencia de Ponce AgTech. Esto significa que ante un problema o desperfecto, los técnicos pueden brindar una asistencia más rápida y eficaz.

- Conectividad: el producto de Hunter solo puede ser utilizado mediante conexión a Internet, utilizando uno de sus kits de conectividad. Por otro lado, el producto de Ponce AgTech y el de Rain Bird pueden ser utilizados *offline*.

- Costo para el usuario: se prevé que Ponce AgTech ofrezca el servicio a 200 dólares por mes, por lo que resulta un coste menor comparado al de las controladoras de Hunter y Rain Bird, que adicionalmente de los valores originales que superan los 1000 USD, deberían tenerse en cuenta los costos por provenir del extranjero, los de instalación y de mantenimiento, aparte de la ausencia de soporte. Además, estos equipos de las dos empresas competidoras son aquellos que podrían encontrarse expuestos a las inclemencias climáticas y su reparación resultaría mucho más costosa en comparación al reemplazo de un microcontrolador Atmel o una Raspberry Pi, placas de bajo costo donde se ejecuta el sistema de Ponce AgTech.

Fabricante	Hunter Industries	Rain Bird	Ponce AgTech
Producto	Genérico para agricultura en general / complejidad similar al de Ponce AgTech	Genérico para agricultura - incluye más funcionalidades que el de Ponce AgTech	Específico para microaspersión y goteo / se planifica agregar más funcionalidad en futuro cercano
Plataforma	*Web / Responsive*	*Web / Mobile*	*Web / Responsive*
Infraestructura	Funciona sólo sobre productos Hunter	Funciona sólo sobre productos Rain Bird	Funciona sobre cualquier sistema de microaspersión y goteo
Soporte	A través de proveedor generico en Buenos Aires - Se desconoce si esta disponible	A través de proveedor generico en Buenos Aires - No se cuenta actualmente con técnico / ingeniero especializado	Local, a medida
Conectividad	Necesita Internet	Puede funcionar *offline*	Puede funcionar *offline*
Costo	Gratis utilizando *hardware* propietario (alrededor de los 1300 dólares)	Gratis utilizando *hardware* propietario (alrededor de los 1000 dólares)	200 dólares mensuales

Figura 31 - Resumen de benchmarking

Capítulo 10: Seguridad

10.1 Seguridad del subsistema en general

La seguridad es un factor crucial en este sistema dado que tiene control total sobre el riego del campo. Es esperable que se busque abarcar la totalidad del riego de los lotes de kiwi disponibles, por lo que un fallo de seguridad implicaría en el peor caso una pérdida de la totalidad de la producción en ese instante. Sin embargo, fue pactado desde las etapas iniciales del proyecto que el subsistema campo no contaría con manejo de usuarios ni contraseña para el acceso a la interfaz web. Esto es porque el producto realizado es un *MVP*, y no fue considerado necesario ya que añadiria un módulo más a desarrollar, por lo que quedó fuera del alcance. El subsistema cuenta con una conexión inestable a Internet, por lo que el problema de autenticación no pudo ser resuelto con un email y contraseña que sean verificados en un *backend* ya que no está garantizado que cuando se quiera acceder al sistema se cuente con conexión. Por otro lado, las soluciones locales fueron discutidas en las primeras instancias del proyecto, como tener una base de datos local que se sincronice cuando haya conexión, pero se acordó que esto podría tener complicaciones adicionales que serán resueltas en instancias superiores al desarrollo del *MVP*. Por último, es importante aclarar que la placa Raspberry Pi cuenta con un sistema de autenticación con usuario y contraseña, propio del sistema operativo que utiliza, Raspbian.

En cualquier caso, la primera barrera de seguridad del sistema es la seguridad física, la cual sí ha sido contemplada en este caso. Sergio Talens-Oliag define la seguridad física como: “aquellos mecanismos --generalmente de prevención y detección-- destinados a proteger físicamente cualquier recurso del sistema; estos recursos son desde un simple teclado hasta una cinta de *backup* con toda la información que hay en el sistema, pasando por la propia CPU de la máquina”[(20)]. En este caso, los mecanismos son exclusivamente de prevención, ya que el

subsistema compuesto por la placa Raspberry Pi y el microcontrolador se encuentran en una casilla cercana a la plantación de kiwis. El edificio está cerrado con llave y solo puede ser accedido por personal autorizado. Sin embargo, esta seguridad es responsabilidad exclusiva de la empresa que sea usuario final del producto, es decir, no es responsabilidad en ningún caso de este proyecto.

10.2 Seguridad en el componente Mapper

Dado que el Mapper es un componente que está entre los dos subsistemas, es decir, no es directamente utilizado por usuarios humanos, no se considera la autenticación ni el manejo de sesiones. Tanto para conectarse al *backend* remoto como al del campo, podría haberse optado por incluir un *middleware* que únicamente tenga en cuenta las solicitudes de conexión de los servidores autorizados, mediante una característica fija y única, como la dirección IP. Opciones de este estilo fueron contempladas durante el desarrollo del Mapper, sin embargo fue considerado que se añadiría excesiva sobrecarga teniendo en cuenta que habría que hacerlo tanto para los *backends* como para los *frontends*. A partir de este punto los dos subsistemas resolvieron el problema de formas distintas.

Por un lado, el *backend* remoto optó por un header que dispone HTTP, Authorization, que es utilizado para proveer credenciales a un servidor y poder acceder a un recurso protegido, en este caso el Mapper. Claro está que se deben mantener las contraseñas en ambos servidores y protegerlas de accesos no autorizados.

Por otro lado, el *backend* en campo solo se conecta al Mapper mediante gRPC, que provee distintas soluciones al problema de autenticación:

- ALTS (Application Layer Transport Security): gRPC soporta Seguridad de transporte de la capa de aplicación para programas que se ejecutan sobre la plataforma Google Cloud Platform, que en este caso no aplica.

- Autenticación basada en *tokens* de Google: utilizada para enviar *tokens* OAuth2 u otros en solicitudes y respuestas en gRPC.
- SSL/TLS: gRPC permite utilizar la tecnología SSL/TLS para encriptar toda la información que se envía entre cliente y servidor.

En vista de que hubo dos opciones posibles, se tuvo en cuenta que la comunicación que se quiso realizar fue servidor a servidor, por lo que si bien OAuth es más sencillo de configurar que SSL, está orientado a autenticar individuos y no servidores. Adicionalmente, se consideró que no ofrece seguridad por sobre SSL, ya que es tan seguro como el *Consumer Secret* sea y SSL es tan seguro como la clave privada sea. Finalmente, se definió que el método elegido sea SSL/TLS mutuo, que además es el recomendado por la documentación del protocolo, aunque la configuración queda fuera del alcance de este proyecto.

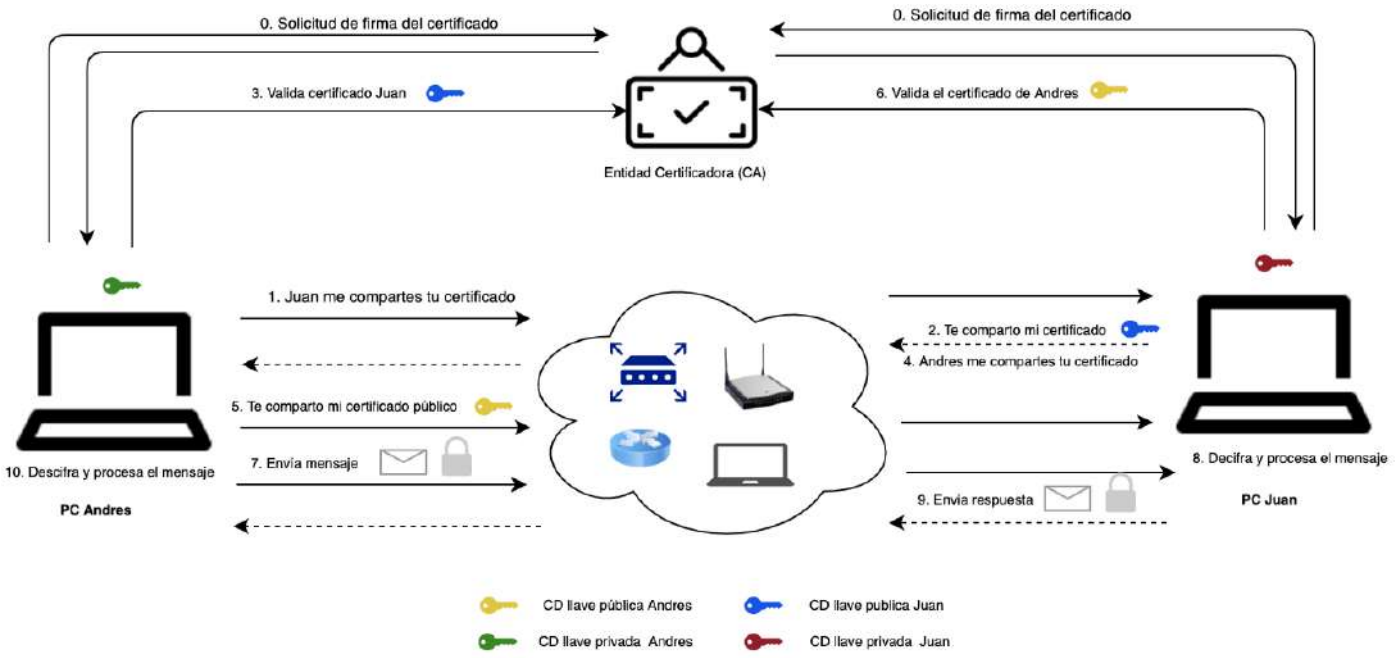

Figura 32 - Flujo de trabajo con TLS mutuo

La TLS mutua[21] es un requisito frecuente para el Internet de las cosas y las aplicaciones entre empresas. Consiste en realizar una autenticación mutua por TLS. Mediante esta metodología el cliente verifica la identidad del servidor, y el servidor

también verifica la identidad del cliente. Los certificados se usan para encriptar el canal de comunicación.

10.3 Auditoría

En la publicación especial del NIST[22] (Instituto Nacional de Estándares y Tecnología, agencia de la Administración de Tecnología del Departamento de Comercio de los Estados Unidos), publicada bajo el nombre NIST 800-92, se establecen normas y pautas para el registro de eventos y su almacenamiento. En el documento se enuncia que mantener una bitácora de eventos es beneficioso y necesario para una empresa en cuanto a que permite identificar fácilmente incidentes de seguridad, violaciones de políticas de seguridad, actividad fraudulenta, problemas operacionales, entre otros. Además, se nombra que en una aplicación (definida en el documento como aquellas destinadas a almacenar, acceder y manipular la información del proceso de negocio de la empresa), se debe mantener un registro especialmente de cuatro tipos de eventos (*logs*):

- Solicitudes y pedidos a sistemas externos
- Información de sesiones. Cambios de perfil, cierres de sesión, etc. No aplicable en el caso del subsistema campo.
- Datos de utilización. Cantidad de transacciones ocurridas en un período.
- Acciones significativas. Errores de sistemas, arranques y apagados, borrado o modificaciones de registros importantes.

En el caso particular del subsistema campo, se ha configurado una bitácora tanto en el Mapper como en el *backend*. De acuerdo a la tecnología utilizada, puede ser configurada antes del inicio del sistema, donde se puede establecer ciertas características de los archivos que almacenan los eventos registrados, como el tamaño máximo, frecuencia de creación, cantidad máxima de archivos a conservar, formato para el nombre y compresión de archivos luego de cierto tiempo (rotación de *logs*). Es fundamental en este dominio la correcta administración de los *logs*

debido al valor monetario de los bienes que se desean administrar, es decir, los cultivos. Esto es ya que el sistema debe proveer a los clientes soporte en caso de errores y asegurarse que se cuente con la información necesaria de los eventos que podrían haber causado desperfectos, al menos dentro del plazo que se ofrece el servicio.

Adicionalmente, se han definido errores con códigos de error internos y descripciones únicas para lograr un mejor seguimiento de los problemas en la bitácora. Dado que para distintas entidades y situaciones posibles, la variedad de errores es grande, por lo que de no existir una lista de errores personalizados se tardaría un tiempo excesivo en encontrar las causas de los problemas. Además, se han distinguido varias categorías para facilitar el agregado de nuevos errores a futuro, para nuevas funcionalidades o para mayores validaciones de las existentes. El protocolo de errores puede verse en el apéndice C.

Por último, adicionalmente la herramienta de bitácora cuenta con tres distintos niveles de granularidad: “error”, “información” y “*debug*”, los cuales consisten en el registro únicamente de errores en el caso de “error”, de acciones del usuario y eventos de servicios en el caso de “información” e información registrada únicamente para facilitar información a desarrolladores en el caso de “*debug*”. Se debe tener en cuenta que los niveles más altos contienen a los anteriores, siendo el de *debug* el más alto y el de error el más bajo.

Capítulo 11: Memoria del proyecto

A continuación se presentan algunas reflexiones obtenidas como resultado del desarrollo del proyecto final, de forma de contrastar hechos con expectativas iniciales y expresar conclusiones personales.

11.1 Cumplimiento de objetivos del proyecto

Objetivo principal

- **Desarrollar un sistema IoT de riego que le permita al productor, tanto de forma local como remota, monitorear y controlar bombas y válvulas, así como establecer programas de riego. De esta manera, le permitirá reducir los tiempos destinados a la supervisión, registrar los eventos en campo, minimizar errores, alertar sobre posibles desperfectos y tomar decisiones y ejecutarlas en el momento.** El objetivo se ha cumplido totalmente y se ha desarrollado el sistema completamente. Toda la funcionalidad comprometida está implementada. Además, se ha tenido en cuenta siempre la potencial mejora continua a futuro por parte de la empresa demandante, por lo que la mantenibilidad y escalabilidad han sido atributos de calidad principales.

Objetivos secundarios

- **Desarrollar competencias al gestionar un proyecto que atraviesa la mayoría de las etapas del ciclo de vida del software y es solicitado por un demandante real.** Objetivo cumplido. Aunque el mantenimiento lo realizará el demandante, las demás etapas del ciclo de vida del software fueron completadas. A lo largo de él se tuvo que hacer frente a diversas complicaciones que surgieron como la definición de qué formaría parte del sistema o riesgos que fueron contemplados y aparecieron, por lo que debían

mitigarse. La incorporación del demandante a la totalidad del proyecto fue una experiencia enriquecedora ya que nos aportó conocimientos sobre esta relación.

- **Impactar positivamente en la región al no existir oferta de herramientas similares en el mercado local.** Aunque este objetivo no ha sido cumplido aún, tiene un gran potencial para lograrlo dado que, luego de realizar el *benchmarking*, quedó claro que si bien existe una diferencia en cuanto a funcionalidad frente a productos similares, no hay competencia en cuanto a la calidad de soporte y mejoras continuas que puede lograr Ponce AgTech por su cercanía a los clientes.

- **Trabajar de manera conjunta con otro grupo en la realización del análisis inicial y manteniendo una comunicación constante a lo largo del proyecto para lograr una solución funcional y fortalecer el trabajo en equipo.** Este objetivo fue cumplido ya que a lo largo del desarrollo del trabajo se mantuvo una comunicación fluida entre los dos grupos, de forma que toda decisión que impactara en los dos subsistemas estuviera analizada y aceptada por ambos. Esto permitió lograr consensos en las decisiones de diseño más importantes del sistema y por lo tanto, se logró la calidad que se buscaba, la cual está caracterizada también por la coherencia entre ambos subsistemas.

- **Ampliar el conocimiento sobre las tecnologías aplicadas en el sistema a partir de su uso e investigación.** Los requerimientos específicos sobre tecnologías a utilizar por parte de la empresa demandante nos impulsó a investigar y ampliar el conocimiento, no solo a otras tecnologías de programación, sino a otras áreas como la electrónica al hacer uso de los microcontroladores Arduino y la placa Raspberry Pi. Además, las decisiones de tecnologías a utilizar se debían tomar en consenso con el otro grupo y el referente técnico, por lo que se debieron realizar comparaciones para

seleccionar la mejor opción. Por lo mencionado el objetivo se considera cumplido.

11.2 Planificación y tiempos reales

Como se ha mencionado anteriormente, el uso de una metodología ágil por sobre la de cascada ha causado que las etapas no se realicen de forma escalonada, sino que se solapen en determinadas partes. Además, también se puede ver que la fase de análisis se extendió más de lo proyectado inicialmente, naturalmente por las reuniones con el demandante y escritura de los documentos iniciales. Los tiempos de imprevistos tenidos en cuenta en la estimación inicial fueron consumidos en su totalidad, incluso sobrepasados debido a la dedicación de los integrantes a otras tareas laborales y académicas, ya que en algunos períodos se encontraban cursando más de tres materias y preparando exámenes finales. Por este último motivo, es importante remarcar que hubo semanas donde no se realizaron actividades. Estas se lograron situar luego de terminar implementaciones complejas, de forma que no quedaran tareas incompletas por largos períodos de tiempo. Por esto, los diseños realizados luego de estas semanas vacías estaban relacionados a nuevas funcionalidades, de forma de que no se perdieran ideas ni tampoco se documentaran con días o semanas de diferencia, sino inmediatamente luego de su concepción.

Por otro lado, las tareas de *testing* no consumieron el tiempo que se había estimado inicialmente. Al iniciar, se contaba con que se utilizaría una herramienta de código para automatizar el testing de la comunicación del subsistema campo con el mapper (gRPC) y la del microcontrolador con la Raspberry Pi (serial). Al no encontrar una forma de realizarlo, se procedió a hacerlo de forma manual, lo que supuso menos tiempo de aprendizaje.

A continuación se presentan los dos gráficos correspondientes a la estimación inicial y los tiempos reales resultantes del proyecto. Las cantidades de horas corresponden totalmente a los integrantes del subsistema campo.

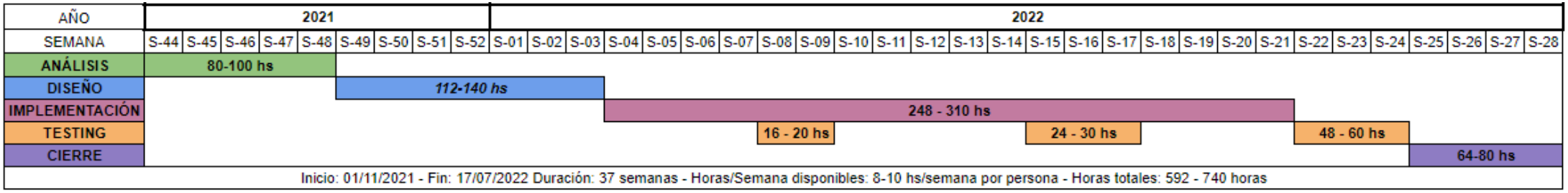

Figura 33 - Cronograma inicial

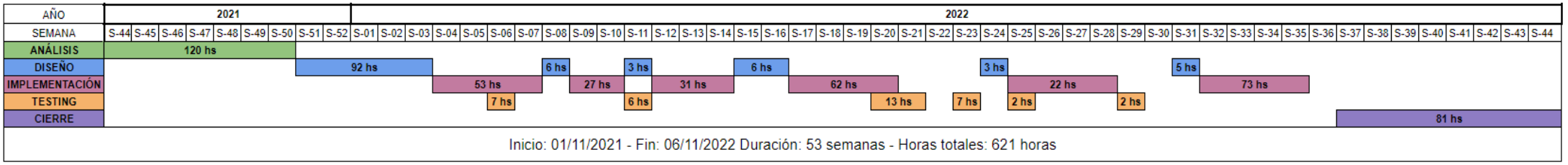

Figura 34 - Cronograma real

11.3 Análisis de etapas

11.3.1 Análisis

Como fue mencionado en el capítulo 6, para la realización del proyecto en un principio se decidió llevar a cabo una metodología en cascada donde la evolución se separa en cinco etapas: análisis, diseño, implementación, *testing* y cierre. Con esto en mente, primero se realizó la tarea de análisis en conjunto con el grupo subsistema nube que llevó un total de 120 horas de trabajo por integrante, más de lo estimado en un principio.

En el comienzo del proyecto se definieron dos días a la semana para tener reuniones con el referente de la empresa demandante. Tenían como finalidad obtener un análisis general de todo el dominio del problema. Una parte de este análisis fue realizada previamente a la entrega del protocolo del proyecto final, debido a que se necesitaba definir su alcance. No se profundizó ni en el diseño ni en la implementación de los módulos ya que se buscaba entender completamente el dominio. Durante esta etapa se tuvo una reunión con un referente técnico de la empresa demandante que ayudó a entender las limitaciones del sistema utilizado hasta el momento y otra con un cliente actual de Ponce AgTech que permitió visualizar la disposición del sistema en campo.

De esta tarea surgieron una gran cantidad de requerimientos que fueron separados en módulos, y debido a la extensión de este relevamiento es que el tiempo fue sobrepasado. A su vez, no todos los requerimientos formaron parte de este proyecto, sino que se definieron tanto requerimientos esperados como deseados, es decir, que serán desarrollados a futuro por quien continúe con el proyecto.

11.3.2 Diseño

La etapa de diseño debía completarse en su totalidad luego de la de análisis y antes de la implementación. Consumió un total de 115 horas de trabajo por integrante, poco más del límite inferior estimado. Al comenzar con ella se notó que no era conveniente definir el diseño de tareas futuras que dependieran de otras, ya que no se contaba con una gran experiencia en proyectos de gran magnitud y se podría caer en errores solo por buscar definir como realizarlas todas, por otro lado, al inicio del proyecto las ventajas de usar *Scrum* frente a la metodología cascada se encontraban difusas y no permitió tomar una decision lo suficientemente fundamentada y que se mantega en el tiempo. Ante estas apreciaciones, se estableció la utilización de una metodología ágil que permitió realizar un primer diseño y refinarlo durante el desarrollo del proyecto. Este cambio trajo sus frutos ya que a medida que el trabajo avanzaba, se dedicaba cierto tiempo a validar el diseño, definir la forma en que se realizarían las tareas y, de ser necesario, aplicar cambios.

Durante esta etapa se tuvieron reuniones de validación con el referente técnico de la empresa demandante que fueron muy enriquecedoras ya que se generaba un debate interesante en cuanto a la selección de las tecnologías o el por qué de las ya utilizadas en los productos de Ponce AgTech. Dentro de una de estas reuniones se propuso la incorporación de la entidad Mapper para cumplir con el requerimiento que solicitaba la utilización de un protocolo liviano para en el futuro añadir comunicación satelital.

Desde ambos equipos las tecnologías utilizadas por la empresa demandante resultaban interesantes y algunas ya conocidas, por lo que fueron aceptadas. Al momento de la definición, la librería de componentes Vuetify que se utilizaría en el desarrollo del cliente web se encontraba en desarrollo en su compatibilidad con Vue 3, pero de igual manera, se lo consideró dentro de las tecnologías a utilizar ya que la fecha de lanzamiento era cercana al inicio del proyecto.

11.3.3 Implementación y pruebas

La etapa de implementación tuvo lugar en un período de 268 horas de trabajo por integrante, un poco más del mínimo estimado en la apreciación inicial. En el momento de comenzar el desarrollo, se dedicó un tiempo a definir con el grupo subsistema nube los repositorios que serían utilizados y las configuraciones iniciales que se aplicarían. Dentro de estas configuraciones se encontraba la implementación de las ya mencionadas librerías que definen reglas de escritura o las que se encargan de verificar el código antes de enviarlo al servidor de Github.

La etapa de implementación se extendió más de lo pensado, debido a que los integrantes se encontraban todavía realizando la cursada de algunas materias y a su vez trabajando, por lo que el tiempo que se dedicó semanalmente se veía reducido y hasta hubo semanas en las que no se pudo avanzar con el proyecto debido a esto. Sin embargo, gracias a la planeación anticipada por parte de ambos grupos y la definición de un trabajo asincrónico, el equipo del subsistema nube pudo continuar trabajando sin necesidad de verse retrasado. A pesar de las demoras, se mantuvieron las reuniones semanales, por lo que cualquier problema o duda era debatido.

La utilización de un dispositivo nuevo como fue el microcontrolador Arduino llevó una curva de aprendizaje, por ello en el inicio, todas las tareas que lo implicaban tenían una duración mayor. Con el tiempo de desarrollo, se obtuvo experiencia que agilizó el trabajo. Durante su entendimiento, se realizaron reuniones con el referente técnico que brindó su conocimiento y enseñó el código utilizado hasta el momento en el sistema que se encontraba en producción.

El desarrollo del cliente web se retrasó hasta el final de la etapa de implementación esperando el lanzamiento de la actualización de la librería Vuetify. A lo largo de este tiempo se postergó varias veces y llegado el momento de construir el *frontend*, solo ofrecía una beta que era recomendada para proyectos de prueba. Por lo que entre los integrantes de ambos grupos se buscó una respuesta a esta problemática con el fin de no retrasar el desarrollo. Como solución, se propuso a Ponce AgTech realizar

un cambio en la tecnología y pasar de utilizar la librería Vue a React, herramienta con la que todos los integrantes trabajaron y de gran comunidad. La presentación a la empresa demandante fue positiva y se aceptó el cambio por lo que se continuó con el desarrollo.

A diferencia de como se estimó inicialmente, esta etapa no se llevó a cabo por módulos. Teniendo en cuenta que mucho del código se iba a repetir, se decidió ordenar el proceso al implementar las funcionalidades de cada entidad. Primero se desarrolló lo referido a bombas, luego válvulas y por último programas e informes de presión en tiempo real.

A la etapa de *testing* se le dedicó un tiempo de 37 horas por integrante, menos del mínimo calculado al momento de estimar. Esto se debe a las siguientes razones: por un lado, la herramienta utilizada para el *testing* de funcionalidad del servidor en campo, Jest, ya era conocida por los integrantes y no se demoró tiempo en escribir el código; por el otro, a diferencia de como se esperaba, no se encontró una herramienta que permitiese realizar *testing* automatizado de la comunicación gRPC y, entre el microcontrolador y la Raspberry Pi. Al realizar la primera aproximación, se tuvo en cuenta la necesidad de dedicar un tiempo de aprendizaje, pero finalmente ambos se probaron de forma manual. Entre las placas se decidió realizar *tests* que permitieran ver cómo impactaba en el microcontrolador las señales enviadas por la placa Raspberry Pi. Como resultado de las pruebas realizadas se definió un protocolo de errores (apéndice C).

11.3.4 Cierre

La etapa de cierre se llevó a cabo una vez finalizada toda la implementación del sistema. Tuvo una duración de 81 horas por integrante, un poco más del rango máximo estimado. A lo largo del desarrollo del producto se elaboró documentación que facilitó la escritura del documento final a entregar. A su vez, se tuvieron reuniones con el referente de la empresa demandante, el co-director y el director con el fin de mostrar el resultado final.

11.3.5 Resumen

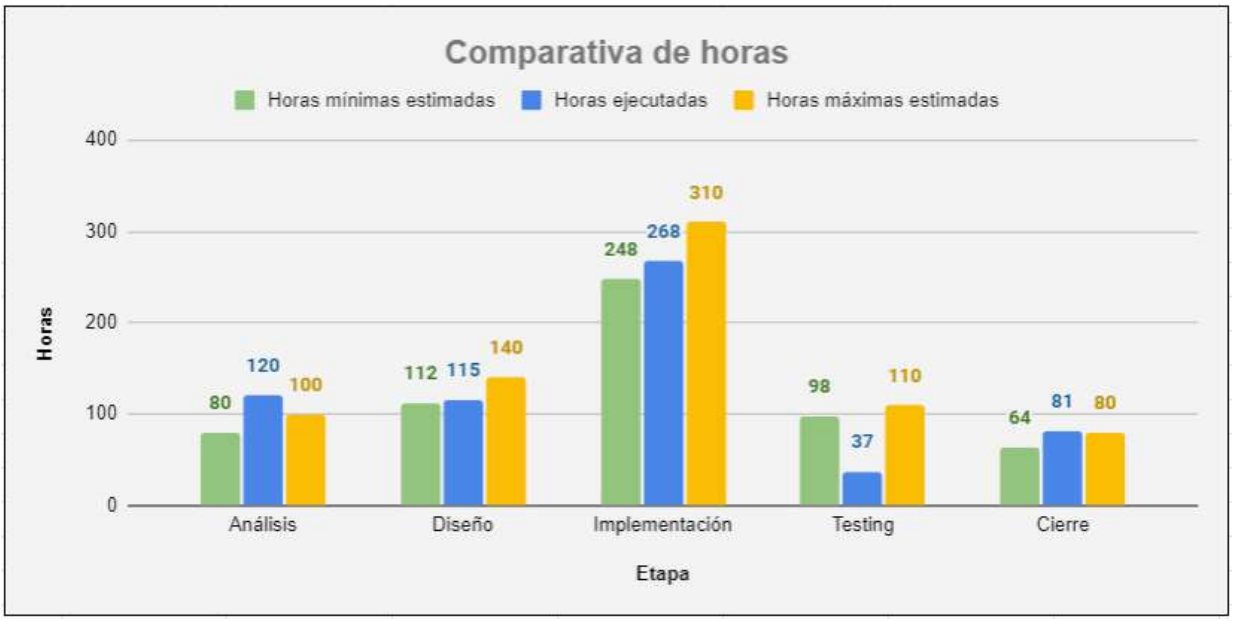

Figura 35 - Gráfico de barras: Horas mínimas estimadas vs Horas ejecutadas vs Horas máximas estimadas

A modo de resumen, se puede ver que no todas las tareas se encontraron dentro del rango establecido previamente, en algunas debido a que resultaron interesantes, y a modo de aprendizaje y experiencia se decidió hacer un mayor foco, como fue la de análisis. Otras debido a problemas surgidos a lo largo del proyecto llevaron menos, como la de *testing*.

Llevándolo a un punto de vista económico, definir la duración de un proyecto implica comprometerse a finalizar en un tiempo estimado. Que sobre o que se supere el tiempo, puede tener consecuencias económicas para el cliente o quien realice el trabajo. En sus bandas mínimas, se podría perder dinero al utilizar más tiempo del acordado debido a malas estimaciones o problemas que surgen durante el desarrollo y que no fueron previstos. Pero una estimación máxima, con tiempos altos, puede ser muy costosa y rechazada por el cliente.

11.4 Trabajo en grupo

Mediante la fragmentación del proyecto en dos subgrupos, se ha logrado un sistema completo. Ambos participaron en las reuniones con la empresa demandante y definieron los requerimientos, interfaces, etc. de modo de lograr un producto lo suficientemente completo para aportar valor y al mismo tiempo para constituirse como un trabajo final independiente y válido. Debido a las experiencias previas similares se lograron enfoques distintos para problemas que surgían a lo largo del desarrollo. Al contar con varias soluciones potenciales, se pudo trabajar siempre de forma eficiente, elegir las mejores y lograr consensos entre todos los integrantes, situación que ocurre frecuentemente en la práctica laboral.

Respecto a la implementación, la experiencia grupal resultó igualmente satisfactoria. La forma de trabajo fue similar en toda la etapa: se comenzaba en una reunión virtual con la selección de una tarea del *backlog*, de forma que se elegía sobre qué se iba a desarrollar y posteriormente se definía cómo. De esta forma, se creaban las subtareas correspondientes a ambos grupos y se asignaban a los integrantes, paralelizando el trabajo y centrándose posteriormente en el propio de cada subsistema. Sin embargo, al enfocarse en el mismo problema, las dudas que pudieran surgir se discutían entre ambos grupos si era necesario, al igual que con las decisiones sobre nuevas tecnologías a utilizar (como ocurrió con los *Server Sent Events*).

11.5 Utilización de nuevas tecnologías

La necesidad de dividir el sistema general en dos subsistemas creó dos proyectos que debían enfocarse en tecnologías diferentes. El interés de los integrantes de cada subgrupo tuvo gran peso en la elección del subsistema a desarrollar, con el fin de ampliar el conocimiento en el área.

Dado que los integrantes de este proyecto tienen interés por las actividades que involucran manipulación de *hardware* y productos IoT fue que se optó por la elección del desarrollo del subsistema en campo. Este subsistema cuenta con dos componentes clave que debieron ser estudiados para ser utilizados como parte del sistema, la placa Raspberry Pi y el microcontrolador Arduino.

Ponce AgTech brindó las placas para que se pueda desarrollar y realizar pruebas usándolas, sin la necesidad de simularlas. Esto llevó a que se deba aprender cómo conectar los actuadores y sensores a la placa física. Para esto no se utilizaron los relés y bombas o válvulas, sino que se reemplazó el sistema que sigue a la placa Arduino por leds que se encendían o apagaban simulando el comportamiento del relé.

La falta de experiencia en la utilización de estos dispositivos no fue un impedimento ya que se mantuvieron reuniones con el referente técnico de la empresa demandante que brindó su conocimiento y experiencia en estos dispositivos. Además, existe una gran cantidad de documentación sobre proyectos realizados con estos componentes que facilitan el aprendizaje.

La utilización de estos dispositivos brindó una gran experiencia en el área del IoT que resulta beneficiosa en la posibilidad de realizar una búsqueda laboral o el desarrollo de nuevos proyectos personales relacionados a esta tecnología.

Capítulo 12: Conclusiones

Desde el comienzo del proyecto se tuvieron dos objetivos claros que luego se segregaron en los planteados anteriormente: el sistema debía aumentar la productividad de quienes lo utilicen y facilitar su accionar evitando tiempos muertos y además, se tenía que adaptar a la gama de productos elaborada por la empresa demandante. Con los resultados del proyecto, se puede concluir que se cumplieron estas necesidades y que se obtuvo una herramienta escalable y mantenible a futuro. Durante el proyecto se obtuvo una gran experiencia, no solo del dominio del problema, sino también de cómo actuar en la interacción con el cliente. Actividad a la que se le da mucha importancia de forma teórica en diversas asignaturas de la carrera.

Al ver la magnitud del proyecto luego de las charlas iniciales con el demandante, estuvo claro que la falta de experiencia en estimaciones y gestión completa de la implementación iba a resultar en imprecisiones y probablemente en no cumplir los tiempos totalmente. Luego de realizar el trabajo, se puede concluir que así fue aunque los tiempos estuvieron cerca de cumplirse por considerar casos excesivamente pesimistas, que se vieron hechos realidad por las obligaciones laborales y universitarias de los integrantes.

El proyecto tuvo como aspecto a destacar la colaboración de dos grupos que trabajaron para desarrollar un mismo sistema. La comunicación entre ambos grupos fue constante y se toma como una experiencia constructiva dado que permitió elaborar un trabajo de una magnitud mayor a la que se podría alcanzar en un grupo de dos integrantes. Además, permitió que cada grupo utilizara un conjunto de herramientas propias para trabajar en su parte, lo que en general no se podría realizar en grupos de dos integrantes debido a las necesidades de tiempo en aprendizaje y de aplicación que tiene el uso de varias tecnologías.

Otro aspecto a destacar es la interacción con un cliente, en este caso, Ponce AgTech. Esto permitió hacer frente a una problemática real, desarrollar un producto

que tiene potencial de ser comercializado y tener que adaptarse a requerimientos impuestos por ellos. Desde ambos grupos se intentó integrar al referente funcional de la empresa demandante en los procesos de análisis en busca de poder definir los requerimientos que mejor se ajusten a sus necesidades y luego para validar los avances obtenidos. La experiencia fue muy enriquecedora, obligó a confrontar situaciones que surgen de esta relación como cambios en los requerimientos o mantener una comunicación a alto nivel.

12.1 Trabajo futuro

Algunas funcionalidades relevadas han quedado fuera de este *MVP*, por lo que quedan pendientes para su implementación futura por parte de la empresa demandante.

- Acceso
  Añadir un mecanismo de autenticación al subsistema campo, de forma que sea consistente con la del subsistema nube, ya sea mediante la sincronización de usuarios, existencia de usuarios offline u otro.

- Alarmas
  Añadir la posibilidad de activar alarmas para el usuario en caso de que surjan eventos que resulten de su interés, como el inicio o parada del sistema, presiones superando valores límites establecidos, entre otros. Además, el usuario podrá activar o desactivar los distintos tipos de alarmas, y establecer números de celular para recibirlas.

- Control
  Establecer límites de presión, tanto de alarma como de corte, de forma que si se sobrepasan se notifique al usuario (mediante el módulo anterior) o se realice la parada total del sistema, con el fin de evitar daños en los equipos. Además, agrega la posibilidad de tener más de un programa activo a la vez.

- Disponibilidad
  Implementar una cola de mensajes de forma que al perder la conexión entre subsistemas, los mensajes pendientes se envien cuando se recupere. El tiempo establecido para mantener los mensajes deberá ser definido, ya que no todos los valores tienen validez luego de un cierto tiempo, por ejemplo, si se envía un aviso de encendido cuando también se debe enviar el de apagado.

- Monitoreo
  Añadir un módulo que detecte lluvias, de forma de parar el riego para no sobrepasar una cantidad de agua preestablecida. Esto implicaría contar con un nuevo tipo de controlador similar al que se implementó para los sensores de presión.

- Reportes
  Almacenar datos de eventos relevantes, como apertura y cierre de válvulas o valores de presión por un tiempo indefinido y permitirle al usuario descargar un reporte con esa información. Además, permitir borrar los datos históricos guardados.

Es importante aclarar que además, existen otras funciones que no han sido relevadas como requerimientos pero a lo largo del desarrollo se identificaron como útiles y necesarias en etapas posteriores:

- Proteger los archivos de bitácora mediante un sistema de firmas, de forma que se garantice la integridad.
- Permitir el acceso a los archivos de bitácora de forma remota.
- Permitir el acceso remoto a métricas del estado de las placas tanto Raspberry Pi como el microcontrolador Arduino.
- Permitir monitorear el estado de la pila en la Raspberry Pi.
- Mantener una sincronización con el backend remoto sobre la información básica del usuario y los códigos identificadores cargados en el archivo de

configuración inicial del backend campo, que actualmente se cargan manualmente.

- Teniendo en cuenta la limitación física de puertos *gpio* del microcontrolador Arduino se debería agregar soporte múltiple a la placa Raspberry Pi.
- Aumentar la cantidad de *tests*. Especialmente realizar las pruebas de integración entre subsistemas antes de la puesta en producción.
- Implementar el método elegido para mantener la seguridad del protocolo en la comunicación entre el Mapper y la Raspberry Pi (SSL/TLS mutuo).

Apéndices

Apéndice A - Glosario

- *Backlog*: lista de tareas pendientes para el equipo de desarrollo, que otorgan valor al cliente y es obtenida de los requerimientos del sistema.

- Git: sistema de control de versiones más utilizado del mundo. Es un proyecto de código abierto con varios años de antigüedad y mantenimiento activo. Tiene como prioridad mantener la integridad del código fuente que gestiona, así como la flexibilidad para trabajar con flujos no lineales y mantener la eficiencia.

- *Internet of Things* (IoT) [23]: red de objetos físicos que llevan incorporados sensores, *software* y tecnologías afines con el objetivo de conectarse entre sí e intercambiar datos con otros sistemas a través de Internet.

- *Linting*: proceso en el cual un programa inspecciona código fuente para encontrar errores comunes y hacer cumplir estándares de escritura, como longitudes máximas de línea, cantidad de tabulaciones, entre otros. En general también detecta código sospechoso de ser un bug. Normalmente es utilizado en forma de extensiones para los programas utilizados para escribir código fuente, como Visual Studio Code.

- *Server Sent Events* (SSE): estándar que describe un tipo de comunicación unidireccional entre un servidor y un cliente, caracterizada por el envío de un flujo de datos hacia el segundo, que normalmente es un navegador.

- REST: estilo de arquitectura de *software* para realizar una comunicación entre cliente y servidor, utilizando HTTP y no almacenando estado entre

solicitudes, es decir, no teniendo memoria entre cada una sobre quien las hizo.

- Microcontrolador: es un circuito integrado programable, contiene todos los componentes de una computadora y es utilizado para realizar una tarea específica, cuyas instrucciones están grabadas en su memoria.

- Raspberry Pi: computadora de placa única, desarrollada por Raspberry Pi Foundation. Tienen un bajo costo y se utilizan para trabajos donde la compatibilidad con una computadora tradicional es necesaria pero al mismo tiempo se necesita un tamaño más reducido y se tolera la limitación de recursos de memoria y almacenamiento.

- *Camel case:* estilo de escritura que se aplica a frases o palabras compuestas, donde se unen palabras sin espacios entre ellas y con la primera letra en mayúscula. Por ejemplo, “camel case” se estila como “camelCase”.

- Remote Procedure Call (RPC): una llamada a procedimiento remoto, en el contexto de la computación distribuida, define la ejecución de código en una computadora remota sin centrar la atención en la implementación de la comunicación de red entre ambas.

- Puertos *gpio*: pin genérico en un microcontrolador que puede actuar como de entrada o salida, y cuyo comportamiento se puede controlar por el usuario en tiempo de ejecución.

- *Logs*: término informático usado para referirse a una persistencia de una serie de eventos que ocurren en un proceso o sistema particular. En general se registra la fecha y hora de ocurrencia junto con una descripción o categorización de lo ocurrido.

Apéndice B - Casos de uso

Los casos de uso mencionados a continuación corresponden a acciones que pueden desarrollarse en el subsistema campo.

CU01: Visualizar estado de bombas o válvulas

Actores: Usuario (Regador/Encargado de campo)

Precondición:

- Deben existir entidades asociadas al sistema.
- Las bombas y válvulas se deben encontrar cargadas en el archivo de configuración inicial.

Postcondición: El usuario conoce fehacientemente el estado de las bombas o válvulas.

Propósito: Permitir al usuario conocer qué elementos están asociados a ese subsistema y el estado tanto de las bombas como de las válvulas.

Resumen: El usuario visualiza un listado de bombas o válvulas en el que se muestra el estado actual (ON/OFF) y si están forzadas a salir de un programa.

Curso normal de eventos:

Acción del usuario	Respuesta del sistema
1. El usuario ingresa a la vista de bombas o válvulas.	2. El sistema muestra el listado con las bombas o válvulas asociadas y el estado de cada una.

Cursos alternos:
2.1 No existen equipos registrados en el sistema. La lista se verá vacía.

CU02: Modificar estado de una bomba o válvula

Actores: Usuario (Regador/Encargado de campo)

Precondición:

- Deben existir entidades asociadas al sistema.
- Las bombas y válvulas se deben encontrar cargadas en el archivo de configuración inicial.

Postcondición: Se modifica el estado (ON/OFF) de una bomba o válvula.

Propósito: Permitir al usuario modificar el estado de funcionamiento de una bomba o válvula.

Resumen: El usuario visualiza un listado de bombas o válvulas en el que se muestra el estado actual (ON/OFF) y modifica el estado de funcionamiento.

Curso normal de eventos:

Acción del usuario	Respuesta del sistema
1. El usuario ingresa a la lista de bombas o válvulas.	2. El sistema muestra el listado con las bombas o válvulas asociadas y el estado de cada una.
3. El usuario modifica el estado de una bomba o válvula deseada.	4. El sistema modifica el estado en campo de la bomba o válvula.
	5. El sistema informa al subsistema nube del cambio realizado desde el subsistema campo.

Cursos alternos:

2.1 No existen equipos registrados en el sistema. La lista se verá vacía.

5.1 No existe conexión a internet o el subsistema nube está caído.

CU03: Forzar una válvula o bomba

Actores: Usuario (Regador/Encargado de campo)

Precondición:

- Deben existir entidades asociadas al sistema.
- Las bombas y válvulas se deben encontrar cargadas en el archivo de configuración inicial.
- La bomba o válvula pertenece a un programa activo.

Postcondición: Se fuerza a la bomba o válvula a salir del programa.

Propósito: Permitir al usuario quitar una bomba o válvula de un programa que se encuentre activo.

Resumen: El usuario visualiza un listado de bombas o válvulas y la fuerza a salir del programa actual.

Curso normal de eventos:

Acción del usuario	Respuesta del sistema
1. El usuario ingresa a la lista de bombas o válvulas.	2. El sistema muestra el listado con las bombas o válvulas asociadas, el estado de cada una y si pertenecen al programa activo.
3. El usuario fuerza a salir del programa activo a una bomba o válvula deseada.	4. El sistema agrega a la bomba o válvula a su lista de forzadas y deja de respetar el programa.

	5. El sistema informa al subsistema nube del cambio realizado desde el subsistema campo.

Cursos alternos:

2.1 No existen equipos registrados en el sistema. La lista se verá vacía.
2.2 No existe ningún programa activo.
2.3 La válvula o bomba que se quiere forzar no pertenece a un programa activo.
5.1 No existe conexión a internet o el subsistema nube está caído.

CU03: Informar presión en campo de una bomba

Actores: Usuario (Regador/Encargado de campo)

Precondición:

- Deben existir bombas asociadas al sistema.
- Las bombas se deben encontrar cargadas en el archivo de configuración inicial.
- El sensor funciona correctamente.

Postcondición: El usuario ve la presión de trabajo de la bomba.

Propósito: Permitir al usuario obtener información actualizada de la presión a la que está funcionando la bomba.

Resumen: El usuario visualiza el detalle de una bomba en específico y puede ver el valor de presión y como se actualiza cada cierto tiempo definido.

Curso normal de eventos:

Acción del usuario	Respuesta del sistema
1. El usuario solicita el detalle de una bomba.	2. El sistema entrega el estado de la bomba solicitada.
3. El usuario se subscribe a los cambios de presión.	4. El sistema suscribe al usuario a los cambios de presión.

6. El usuario visualiza el cambio de presión en el detalle de la bomba.	5. El sistema informa luego de cierto tiempo definido en su configuración un cambio de presión a quienes están suscritos.
	7. El sistema informa al subsistema nube el cambio de presión.

Cursos alternos:

5.1 Se pierde la conexión con un suscriptor.
7.1 No existe conexión a internet o el subsistema nube está caído.

CU05: Modificar estado del programa

Actores: Usuario (Regador/Encargado de campo)

Precondición:

- Debe existir un programa cargado.

Postcondición: El usuario modifica el estado del programa.

Propósito: Permitir al usuario detener o restablecer un programa cargado en el servidor.

Resumen: El usuario visualiza el programa y puede detenerlo o restablecerlo.

Curso normal de eventos:

Acción del usuario	Respuesta del sistema
1. El usuario solicita el programa cargado.	2. El sistema entrega el programa cargado y su estado actual.
3. El usuario modifica el estado del programa cargado.	4. El sistema modifica el estado del programa.
	5. El sistema informa el cambio al subsistema nube.

Cursos alternos:

2.1 No existe un programa cargado
5.1 No existe conexión a internet o el subsistema nube está caído.

Apéndice C - Protocolo de errores internos de *backend* campo

Código	Tipo	Descripción
Relativos a identificadores no válidos		
101	Validación	The pump id is not valid
102	Validación	The valve id is not valid
Relativos a datos no válidos		
110	Validación	Status code provided not valid
111	Validación	Element type code provided not valid
112	Validación	Status code provided does not exist
113	Validación	Element type code provided does not exist
Relativos a elementos no encontrados		
120	Validación	The valve with the provided id has not been found
121	Validación	The pump with the provided id has not been found
Relativos a cambios de estado		
130	Validación	Error changing the valve state
131	Validación	Error changing the pump state
Relativos a archivos de configuración		
140	Archivo	The configuration file does not exist
Relativos al Arduino		
150	Arduino	Arduino unreachable
Relativos a programa		
160	Validación	Invalid program name
161	Validación	Invalid program start hour
162	Validación	Invalid program steps
Relativos a pasos de programa		
170	Validación	Invalid step start hour
171	Validación	Invalid step duration

172	Validación	Invalid step element type
173	Validación	Invalid device ID
174	Validación	The device ID does not exist in the field data
Relativos a la ejecución de un programa		
180	Validación	The device ID does not exist or the program did not start yet
181	Validación	Invalid element type
182	Validación	Invalid element id
183	Validación	Invalid override value

Apéndice D - APIs

Características generales de API REST en *backend* campo:

- Interfaz REST, funciona en HTTP.
- Interfaz desarrollada en Express.
- La ruta siempre es */api/{entidad}*, pudiendo tomar entidad el valor “pump”, “valve”, “current-program”, “override”, “user-static-data” o “suscribe”.
- Sirve para comunicar al *backend* con el *frontend* campo, facilitando la información necesaria para mostrar en las pantallas y realizar las distintas acciones.
- No se realiza control de acceso. Es decir, no hay identificación de usuarios o restricciones para roles.
- Los errores devueltos por la API pueden ser genéricos de HTTP (en general 400 o 404), pero también pueden incluir a aquellos descriptos en el apéndice C. El formato cuando se devuelven este tipo de errores personalizados es el siguiente:

```
{
    errorCode: número de codigo de error interno,
    msg: mensaje descriptivo,
}
```

- Las validaciones se realizan especialmente en *backend*, a pesar de que los campos utilizados para tipos de datos específicos realizan validaciones en *frontend* per se.
- Los errores que no son específicos de datos enviados en las solicitudes a la API también son informados en los errores de las respuestas del servidor. Por ejemplo, si falla la comunicación con el Arduino.
- Para alterar el estado de una entidad, en general se ha utilizado una estructura donde en la URL se envía el ID del elemento, y en el *body* el nuevo estado. Por ejemplo:

```
URL: http://{{host}}:{{port}}/api/pump/a3d5917d-df91-4030-87c3-ed39c43fddf8

{
    newState: “ON”,
}
```

Nótese que en el caso del programa actual no es necesario el envío del ID, debido a que solo puede alterar el estado de un único programa.

Características generales de API gRPC de *backend* campo y Mapper.

- Utiliza HTTP/2 pero no está expuesto a los desarrolladores.
- La herramienta utilizada para cargar los archivos “.proto” ha sido la oficial suministrada en el paquete de *grpc-node*, gRPC Protobuf Loader [(24)] .
- Sirve para comunicar al Mapper con el *backend* campo. Los dos actúan como clientes y servidores gRPC, y cuentan con interfaces independientes definidas en el archivo “.proto”.
- Los pasos para ejecutar un procedimiento remoto son:
    - Elegir un procedimiento de los especificados en el archivo proto.
    - Especificar los parámetros necesarios.
    - Utilizar el *stub* para llamar al procedimiento remoto.
- Las fechas enviadas a través de gRPC se envían en el formato ISO.
- En el caso de que no se envíen los parámetros necesarios, se enviaran valores nulos dependiendo el tipo de los parámetros del mensaje. Esto está definido por la propiedad *defaults*. Por ejemplo, si se quiere enviar un mensaje para forzar el estado de un elemento, se envía el siguiente mensaje:

```
message OverrideElementRequest {
    string elementType = 1;
    string elementId = 2;
    bool override = 3;
}
```

Si al momento de enviar el mensaje, no se especifica ningún parámetro, el mensaje se enviará de la siguiente forma:

```
{
    elementType: “”,
    elementId: “”,
    override: false,
}
```

Se han definido las siguientes reglas al momento de inicializar los clientes y servidores gRPC, con su correspondiente explicación extraída de la documentación oficial.

Opción	Valores posibles	Valor elegido	Explicación
keepCase	true/false	true	Preserva el nombre de los campos. El valor por defecto hace que cambien a camel case.
longs	String/ Number	String	El tipo utilizado para representar valores numéricos largos.
defaults	true/false	true	Establece valores por defecto a aquellos que no se especifiquen pero están presentes en la definición en el archivo “.proto”.
arrays	true/false	false	Establece valores por defecto a aquellos *arrays* que no se especifiquen pero están presentes en la definición en el archivo “.proto”.

objects	true/false	false	Establece valores por defecto a aquellos objetos que no se especifiquen pero están presentes en la definición en el archivo ".proto".

Existen otras reglas que se han dejado en sus valores por defecto, por no aportar valor significativo al sistema. Sin embargo, las propiedades *arrays* y *objects* se han seteado a propósito con valores falsos por defecto debido a que se desea que este tipo de errores no se propaguen hacia el sistema donde se envían, sino que fallen durante el envío. Un ejemplo práctico de esta situación es en el envío de programas (del Mapper hacia el *backend* campo), donde existe un campo llamado "steps" que contiene los pasos del programa.

Servicios y mensajes del protocolo gRPC. En cada *message* se define el tipo de los datos que se enviarán en el formato definido por Protocol Buffer. Los service representan las funciones que se pueden ejecutar de forma remota.

API gRPC Raspberry (servidor campo):

```
service RaspberryController {
    rpc ChangeElementState(ChangeElementStateRequest)
    returns (ChangeElementStateReply) {}
    rpc SetProgram(SetProgramRequest)
    returns (SetProgramReply) {}
    rpc OverrideElement(OverrideElementRequest)
    returns (OverrideElementReply) {}
}

message ProgramStepsDef {
    string startTime = 1;
    int64 duration = 2;
```

```
    string deviceType = 3;
    string deviceId = 4;
}

message ProgramDataDef {
    string id = 1;
    string name = 2;
    string startTime = 3;
    repeated ProgramStepsDef steps = 4;
}

message OverrideElementRequest {
    string elementType = 1;
    string elementId = 2;
    bool override = 3;
}

message OverrideElementReply {
    string elementType = 1;
    string elementId = 2;
    bool override = 3;
}

message SetProgramRequest {
    string elementType = 1;
    ProgramDataDef programData = 2;
    string targetState = 3;
}

message SetProgramReply {
    string elementType = 1;
    ProgramDataDef programData = 2;
```

```
    string targetState = 3;
}

message ChangeElementStateRequest {
    string elementType = 1;
    string elementId = 2;
    string targetState = 3;
}

message ChangeElementStateReply {
    string elementType = 1;
    string elementId = 2;
    string newState = 3;
}
```

API gRPC Mapper:

```
service MapperController {
    rpc HandleFieldUpdate(HandleFieldUpdateRequest)
    returns (HandleFieldUpdateReply) {}
    rpc InformHeartbeat(HeartbeatRequest)
    returns (HeartbeatReply) {}
    rpc HandleFieldOverrideUpdate(HandleFieldOverrideUpdateRequest)
    returns (HandleFieldOverrideUpdateReply) {}
    rpc NotifyPressure(PressureRequest)
    returns (PressureReply) {}
}

message HeartbeatRequest {
    string date = 1;
    string fieldId = 2;
```

```
}

message HeartbeatReply {
    int64 statusCode = 1;
    string fieldId = 2;
}

message HandleFieldUpdateRequest {
    string elementType = 1;
    string elementId = 2;
    string targetState = 3;
}

message HandleFieldUpdateReply {
    bool actionPerformed = 1;
}

message HandleFieldOverrideUpdateRequest {
    string elementType = 1;
    string elementId = 2;
    bool override = 3;
}

message HandleFieldOverrideUpdateReply {
    bool actionPerformed = 1;
}

message PressureRequest {
    string elementId = 1;
    float pressure = 2;
    string date = 3;
}
```

```
message PressureReply {
    int64 statusCode = 1;
    string elementId = 2;
}
```

Apéndice E - Server Sent Events

Para permitir la visualización de los valores en vivo de presión de bombas, se ha utilizado un mecanismo llamado *Server Sent Events*[25], mensajería de una vía que permite recibir actualizaciones automáticas desde el servidor. Para esto se ha habilitado un *endpoint "*/api/suscribe", para que desde el *frontend* se haga una solicitud con los siguientes headers:

```
'Cache-Control': 'no-cache',
'Content-Type': 'text/event-stream',
'Connection': 'keep-alive'
```

De esta forma el controlador de eventos ubicado en el servidor de campo almacena la conexión para que cuando se releva un nuevo valor de presión, brindado por el controlador de sensores, se envía automáticamente y se actualiza en el *frontend*.

El formato del mensaje enviado es:

```
{
    id: id de la bomba
    pressure: número que indica el valor de presión
    pressureDate: fecha de la toma de presión
}
```

Debe ser aclarado que las distintas instancias de *frontend* se suscriben a este tipo de eventos para todas las bombas, de forma que deben ser filtrados en él solo los valores de la bomba que se esté mostrando, mediante el ID.

Bibliografía

(0) *La startup del mes Ponce AgTech*. Club AgTech. Recuperado el 28/09/2022 de https://clubagtech.com/la-startup-del-mes-ponce/.

(1) *¿Qué es scrum?* Claire Drumond. Recuperado el 15/12/2021 de https://www.atlassian.com/es/agile/scrum#:~:text=Scrum%20de%20scrums%20es%20una,ejemplos%20de%20Atlassian%20y%20otros.

(2) *Flujo de trabajo de Gitflow.* Atlassian. Recuperado el 05/01/2022 de https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow.

(3) *El cultivo de kiwi*. Infoagro. Recuperado el 15/10/2022 de https://www.infoagro.com/frutas/frutas_tropicales/kiwi.htm#:~:text=El%20cultivo%20del%20kiwi%20necesita,de%20agua%20en%20el%20terreno.

(4) *"En Sierra de los Padres se produce el 80% del kiwi nacional que se consume en el país"*. El marplatense. Recuperado el 15/10/2022 de https://elmarplatense.com/2022/08/03/en-sierra-de-los-padres-se-produce-el-80-del-kiwi-nacional-que-se-consume-en-el-pais/#:~:text=Los%20kiwis%20de%20Mar%20del,pueda%20crecer%20a%20la%20perfecci%C3%B3n.

(5) Maarten van Steen y Andrew S. Tanenbaum (2018) *Distributed Systems (Tercera edición)*. CreateSpace.

(6) *What is IoT?* Amazon AWS. Recuperado el 10/02/2022 de https://aws.amazon.com/es/what-is/iot/.

(7) *Qué es Arduino, cómo funciona y qué puedo hacer con uno*. Yúbal Fernández. Recuperado el 26/09/2022 de https://www.xataka.com/basics/que-arduino-como-funciona-que-puedes-hacer-uno.

(8) *Arduino Mega 2560, el hermano mayor de Arduino UNO*. José Guerra Carmenate. Recuperado el 28/09/2022 de https://programarfacil.com/blog/arduino-blog/arduino-mega-2560/.

(9) *¿Qué es un Gateway IoT?* Lanner. Recuperado el 01/10/2021 de https://www.lanner-america.com/es/blog-es/que-es-un-gateway-iot/.

(10) *11 Most In-Demand Programming Languages in 2022*. Berkeley University of California. Recuperado el 23/09/2022 de https://bootcamp.berkeley.edu/blog/most-in-demand-programming-languages/.

(11) *Protobuf ¿Qué es Protocol Buffer?* Ionos. Recuperado el 20/02/2022 de https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/protocol-buffers/.

(12) *¿Qué es gRPC?* Ionos. Recuperado el 05/03/2022 de https://www.ionos.es/digitalguide/servidores/know-how/que-es-grpc/.

(13) *What is gRPC? Official documentation*. Cloud Native Computing Foundation. Recuperado el 05/03/2022 de https://grpc.io/docs/what-is-grpc/.

(14) *gRPC vs REST*. Martin Nally. Recuperado el 12/03/2022 de https://cloud.google.com/blog/products/api-management/understanding-grpc-openapi-and-rest-and-when-to-use-them.

(15) *Web oficial Hunter Industries*. Hunter Industries. Recuperado el 08/11/2021 de https://www.hunterindustries.com/.

(16) *IrriExpress*. Senninger Irrigation. Recuperado el 13/11/2021 de https://www.senninger.com/irrigation-product/irriexpress.

(17) *Drip Irrigation*. Rain Bird. Recuperado el 15/11/2021 de https://www.rainbird.com/agriculture/products/drip-irrigation.

(18) *Rain Bird Application*. Rain Bird. Recuperado el 22/10/2022 de https://play.google.com/store/apps/details?id=com.rainbird&hl=es_AR&gl=US.

(19) *Web oficial Aguas SRL*. Aguas SRL. Recuperado el 20/10/2022 de https://www.aguas.ar/.

(20) *Seguridad física*. Sergio Talens-Oliag. Recuperado el 15/02/2022 de https://www.uv.es/sto/cursos/icssu/html/ar01s04.html.

(21) *Autenticación mutua con TLS — Del requisito a la implementación*. Giovany Villegas Gómez. Recuperado el 05/10/2022 de https://medium.com/bancolombia-tech/autenticaci%C3%B3n-mutua-con-tls-del-requisito-a-la-implementaci%C3%B3n-2cd71fea3e90.

(22) Karen Kent y Murugiah Souppaya (2006) *Guide to Computer Security Log Management*. CreateSpace.

(23)*¿Qué es el IoT?* Oracle. Recuperado el 10/02/2022 de https://www.oracle.com/ar/internet-of-things/what-is-iot/.

(24) *Protocol Buffer: Documentación oficial*. Google. Recuperado el 20/02/2022 https://developers.google.com/protocol-buffers/docs/overview.

(25) *How To Use Server-Sent Events in Node.js to Build a Realtime App*. Sebastian Alvarez. Recuperado el 14/06/2022 de https://www.digitalocean.com/community/tutorials/nodejs-server-sent-events-build-realtime-app.