UNIVERSIDAD DE TARAPACÁ

FACULTAD DE INGENIERÍA

Departamento de Ingeniería en Computación e Informática

Plan de proyecto
BotaniDrip: Sistema de riego automatizado, remoto e intuitivo para casas inteligentes con enfoque asistencial

Autor(es): Patricio Chang

Francisco Pantoja

Hernan Vazque

Asignatura: Proyecto II

Profesor y Académico: Diego Aracena P.

ARICA, 02 de enero de 2024

Historial de Cambios

Fecha	Versión	Descripción	Autor(es)
12/09/2023	1.0	Versión preliminar del formato	Patricio Chang
23/09/2023	1.1	Terminar informe 1	Patricio Chang Francisco Pantoja Hernan Vazque
21/11/2023	2.1	Corrección Informe 1	Francisco Pantoja
26/11/2023	2.2	Terminar Informe 2	Patricio Chang Francisco Pantoja
17/12/2023	3.1	Corrección Informe 2	Francisco Pantoja
27/12/2023	3.2	Terminar Informe 3	Francisco Pantoja Hernán Vazque

Tabla de contenidos

1. Panorama General

1.1.1 Introducción

El pasatiempo de regar y cuidar plantas tiene un impacto positivo en adultos mayores, algunos de sus beneficios son el prevenir enfermedades mentales y mejorar el estado emocional. El proyecto Botani Drip busca acercar este pasatiempo a personas con problemas de movilidad reducida o pertenecientes al grupo etario de la tercera edad. Gracias al auge de la tecnología en la vida cotidiana de las personas, este tipo de proyectos es viable e implementable, dando soluciones concretas a problemas específicos.

1.1.2 Propósito, alcance, objetivos

Propósito:
El propósito de este proyecto es lograr que personas pertenecientes a la tercera edad y/o problemas de movilidad reducida, no vean impedidas sus formas de mantenerse activos debido a sus capacidades limitadas.

Alcance:
Este proyecto suple la necesidad de regar plantas sin la necesidad de un esfuerzo físico y proveer información al respecto gracias a un sistema de monitoreo, siempre enfocado de manera asistencial. Si bien puede usarse para otros fines y otro tipo de usuario, el público objetivo de nuestra propuesta es acotado. Además de realizar la operación básica de riego, las herramientas estadísticas que provee el monitoreo del proyecto es en lugar de detallada, precisa y concisa para que cualquier usuario pueda usar la información de sus riegos de la forma que estime conveniente.

Objetivos:
General: Desarrollar un sistema de riego automatizado, remoto e intuitivo para ser implementado en casas inteligentes con enfoque asistencial.

Específicos:
- Conocer, estudiar y usar herramientas de Internet de las cosas (IOT) como Raspberry Pi y sensores analógicos
- Desarrollar software del sistema que permita interactuar al usuario final con la información de sus plantas
- Implementar Botani Drip como sistema de riego automatizado

1.1.3 Suposiciones

Al momento de realizar este proyecto, hay que considerar lo siguiente: Todas las herramientas de trabajo y sensores requeridos estarán disponibles para ser usados cuando se necesite dentro del proyecto. El grupo cumplirá con los plazos establecidos por su organización interna y el profesor. El producto elaborado cumple con todo lo requerido y propuesto en este informe. El grupo aprenderá uniformemente y en conjunto los conocimientos básicos para desarrollar e implementar el producto.

1.1.4 Restricciones

-Se debe utilizar una Raspberry Pi como elemento principal del producto -La comunicación con la Raspberry Pi debe ser mediante una aplicación para dispositivos móviles
-Los recursos son limitados, en caso de averías se deberá buscar otro tipo de tecnología para suplir el requerimiento
-No se permiten más de un usuario asociado al producto implementado

1.1.5 Entregables

Los entregables de este proyecto son:
-Carta Gantt
-Bitácoras Semanales
-Planificación de Gestión de Riesgos
-Planificación del Personal y Recursos

2. Organización del Proyecto

2.1 Personal y entidades internas

Los miembros de este proyecto son:

1. Patricio Chang Reyes
2. Francisco Pantoja González
3. Hernán Vazque Lorca

2.2 Roles y responsabilidades

Todos los miembros del equipo cumplen la función de desarrolladores y programadores. Sin embargo, los roles específicos definidos para el proyecto son:

Jefe de Grupo: Encargado de coordinar, liderar y motivar al equipo de trabajo en el cumplimiento de los plazos y tareas establecidas

Ejecutivo de Repartos: Planifica, redacta y guía la escritura de bitácoras, informes por fases y contenido de la Wiki

Analista y Programador Jefe: Encargado de recomendar e introducir a la capacitación de los demás miembros del equipo en el uso de diversas herramientas de software. Corrige inminentes errores de implementación y programación.

2.3 Mecanismos de Comunicación

Se ha establecido la herramienta Redmine, donde está la Wiki, Carta Gantt, Bitácoras semanales
Para coordinar reuniones fuera del horario de clases se ha creado un grupo de Whatsapp
Para compartir y editar documentos de los entregables se ha establecido una carpeta de Google Drive

3. Planificación de los procesos de gestión

3.1 Planificación de estimaciones

Jefe de Grupo: $23.000, horas totales: 192, $4.416.000
Ejecutivo de Entregables: $25.000, horas totales: 150, $3.750.000
Analista y Programador en Jefe: $18.000, horas totales: 200, $3.600.000 Tiempo para programación: 4 meses

Hardware		
Nombre	Cantidad	Total
Raspberry Pi 4 Modelo B	1	$85.000
Notebook de Trabajo	2	$600.000
Teléfono Inteligente	1	$250.000
Set Sensor Higrométrico con pines	1	$1.600
Mini Bomba de Agua	1	$2.100
Módulo de Relay	1	$2.500
Sensor de Conductividad	1	$5.000
Tubo PVC	2	$2.000
Macetas	4	$7.200
Alfombrilla Impermeable	1	$500
Total		$ 955.900

Tabla 1: Costos de Hardware

Software	
Nombre	Total
Sistema Operativo Raspbian	$0
Sistema Operativo Android 14	$0
Lenguaje de Programación Python 3	$0
Lenguaje de Programación Kotlin	$0
Entorno de Desarrollo integrado Visual Studio Code	$0
Entorno de Desarrollo Integrado Android Studio	$0
Plataforma de Trabajo Redmine	$0
Total	$0

Tabla 2: Costos de Software

3.2 Planificación de recursos humanos

Analistas: 1, Diseñador:1, Programador: 3, Jefe de Proyecto: 1.
Se han asignado los siguientes cargos especiales a cada integrante:
Jefe de Grupo: Francisco Pantoja
Ejecutivo de Repartos: Patricio Chang
Analista y Programador en Jefe: Hernán Vazque

3.3 Riesgos

Algunos riesgos a considerar dentro del proyecto están clasificados según probabilidad de ocurrencia y nivel de impacto (siendo 1 el menor y 2 el mayor) en la siguiente tabla:

RIESGOS	PROBABILIDAD DE OCURRENCIA	NIVEL DE IMPACTO	ACCIÓN REMEDIAL
Cambios en los requerimientos	75%	2	Rastrear la información para valorar el impacto de los nuevos requerimientos, pensar en soluciones que integren nuevos requerimientos con los antiguos para aminorar el tiempo extra de planeación.
Reestructuración organizacional	50%	1	Asegurarse que la reestructuración permita cumplir con los requerimientos más eficientemente que de la forma antigua, hacer contribuciones muy importantes a las metas del proyecto.
Problemas financieros de la organización	50%	2	Solicitar más recursos al encargado de recursos o en su defecto planificar y rehacer estimaciones de costos
Materiales dañados	75%	1	Reemplazar el material dañado por uno nuevo
Problemas en el equipo	50%	2	Conversar entre los miembros del equipo afectados para llegar a un acuerdo y/o decisión al respecto
Pequeños problemas en el proyecto	50%	2	Reconocer los problemas encontrados en el proyecto, para encontrar una solución óptima y eficaz.
Fallo en la programación	75%	2	Conversar entre todos los miembros del equipo para ajustar y/o incurrir en reprogramaciones
Exposición a condiciones climáticas no favorables	75%	2	Ajustar la protección de la raspberry pi a las condiciones climáticas de la zona
Mantenimiento a largo plazo	50%	2	Hacer el mantenimiento o reemplazo lo más simple posible, además de crear un manual para poder dar mantenimiento al proyecto
Enfermedades del Personal	25%	1	Cubrir el trabajo del miembro afectado. De ser una situación grave se debe pensar en reestructurar la organización del proyecto
Problemas en el Reclutamiento	25%	2	Reorganizar el equipo de tal forma que haya traslape en el trabajo y las personas comprenden el de los demás

Tabla 3: Riesgos y Acciones Remediales

3.4 Factores de Riesgo

Otros tópicos que pueden llegar a producir riesgos a considerar son los siguientes:

TIPO DE RIESGO	INDICADORES POTENCIALES
Tecnología	Entrega retrasada del hardware o de la ayuda del software, muchos problemas tecnológicos reportados
Personas	Baja moral del personal, malas relaciones entre los miembros del equipo, disponibilidad de empleo
Organizacional	Chismorreo organizacional, falta de acciones por el administrador principal
Herramientas	Rechazo de los miembros del equipo para utilizar herramientas, peticiones de estaciones de trabajo más potentes
Requerimientos	Peticiones de muchos cambios en los requerimientos
Estimación	Fracaso en el cumplimiento de los tiempos acordados y en la eliminación de defectos reportados
Mantenimiento	Fallo del hardware o de los materiales usados

Tabla 4: Factores de Riesgo

4. Planificación de Actividades

El desarrollo del proyecto ha sido plasmado en la siguiente carta Gantt, la cual va desde comienzos de Agosto hasta finales de Diciembre de 2023:

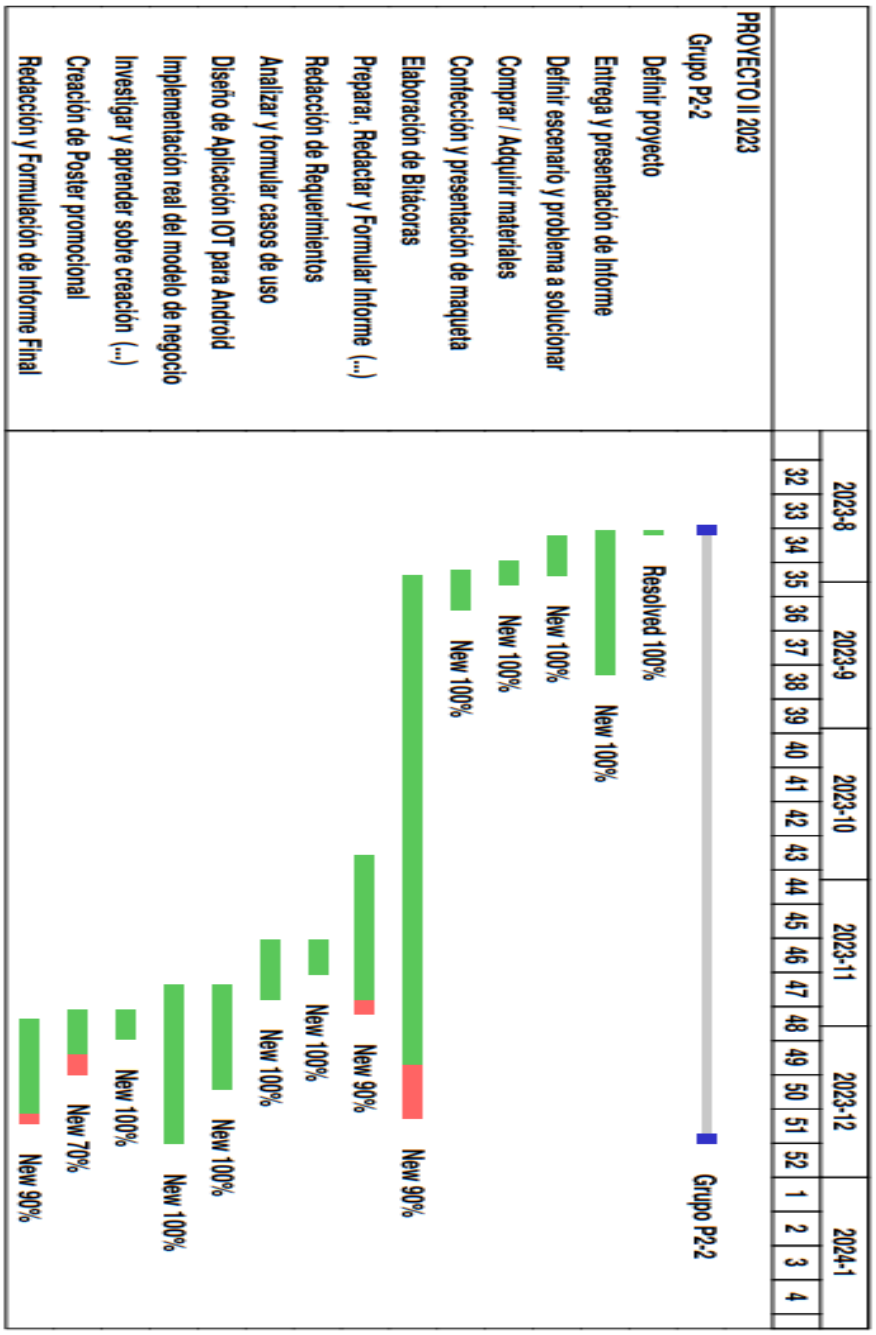


Figura 1: Carta Gantt (actualizado 27/12/2023)

5. Planificación de Procesos Técnicos

5.1 Modelo de Proceso

Dado el objetivo estratégico de Regar plantas se desglosan los Casos de Uso:
- Regar
- Regar manualmente
- Programar riego automático
- Regar automáticamente
- Mostrar condiciones del suelo
- Mostrar historial de riego

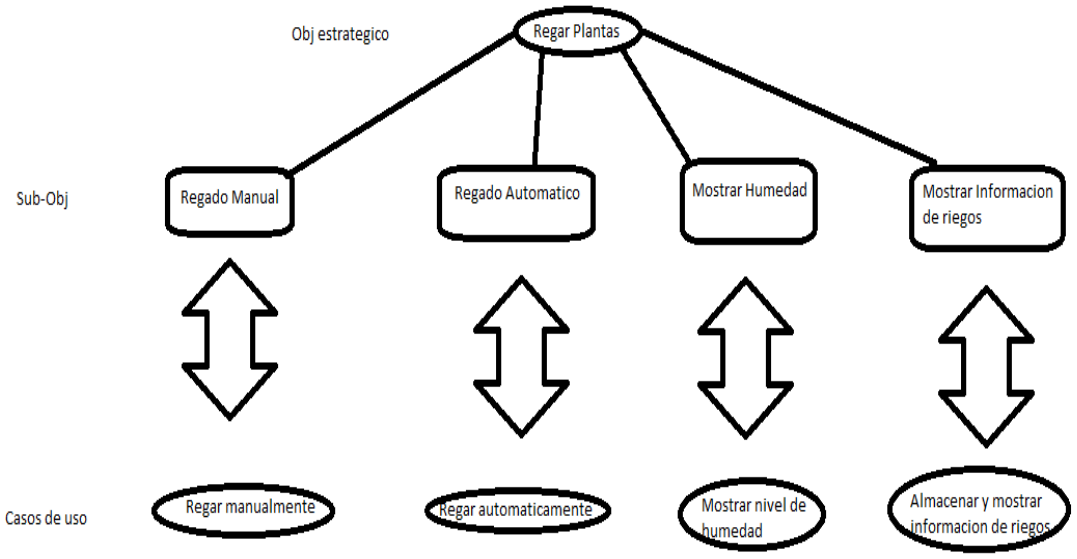


Figura 2: Modelo del Proceso

5.1.1 Requerimientos establecidos

Requerimientos funcionales:
- Efectuar riegos remotamente por accionamiento manual y/o automático
- Recopilar y desplegar estadísticas sobre las condiciones del suelo
- Notificar al usuario cuando se haya efectuado un riego
- Almacenar información de Riegos efectuados
- Visualizar listado de Riegos efectuados

Requerimientos no funcionales:
- Conexión entre usuario y raspberry debe ser por medio de una red Wi-fi implementando el modelo de paradigma cliente-servidor
- El cliente debe encontrarse en el dispositivo móvil del usuario
- El servidor debe estar alojado en la Raspberry Pi y recibir peticiones del cliente por medio del protocolo https

5.1.2 Modelo de Diseño (Caso de uso general)

El modelo de proceso cuenta con los siguientes actores:
-Cliente
-Raspberry Pi
-Bomba de agua
-Sensor de humedad
-Sensor de conductividad

Se ha modelado el funcionamiento del sistema e interacción entre actores de la siguiente manera:

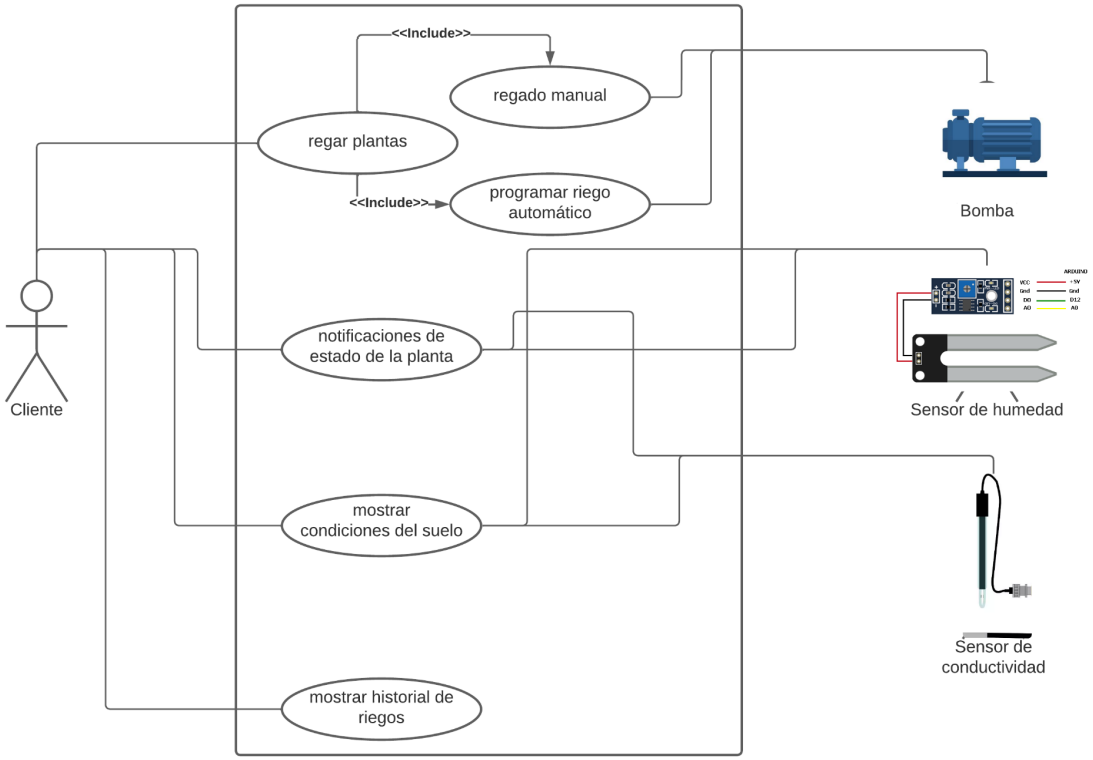
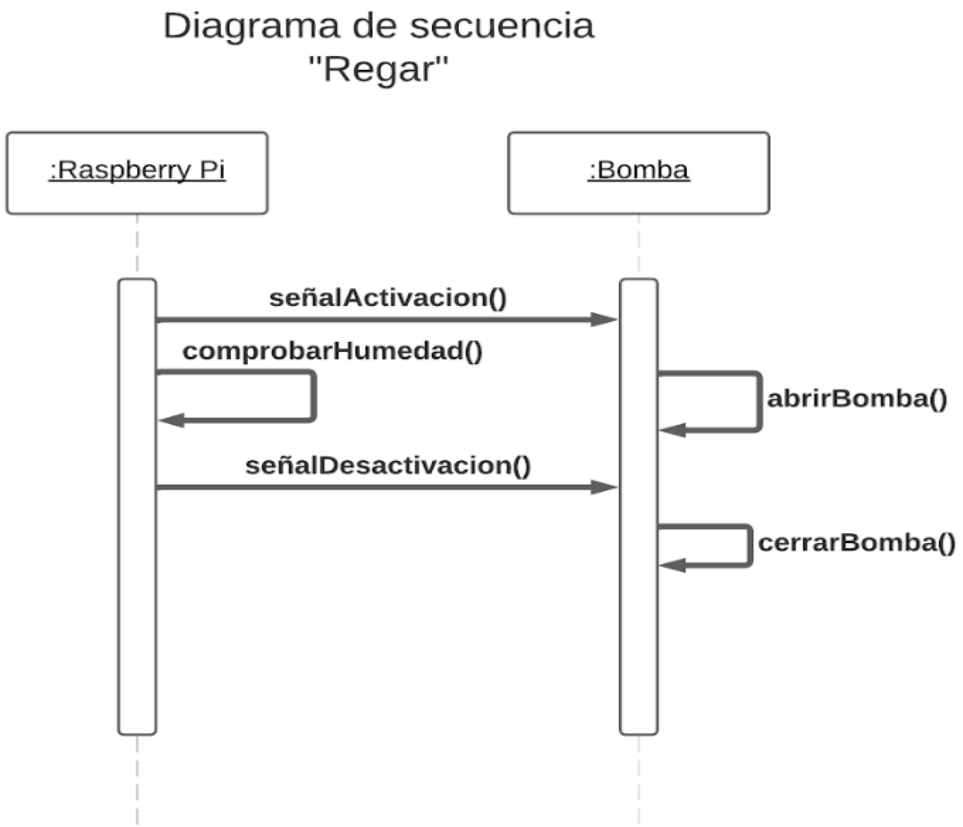


Figura 3: Diagrama de Casos de Uso

Caso de uso	Regar
Descripción	Las plantas son regadas por el sistema a través de la Raspberry Pi
Actor	**Bomba, Raspberry Pi**
Precondición	La bomba debe estar conectada a la Raspberry Pi y se debe recibir una señal de activación de esta.
Flujo Principal	1. Al recibir una señal de activación de la Raspberry Pi se activan los mecanismos de regado 2. Mientras el porcentaje de humedad sea inferior a 100% los mecanismos de riego funcionan activando la bomba de agua
	Las plantas habrán sido regadas y la información del riego efectuado se añadirá al historial de riegos



Caso de uso	Regar manualmente
Descripción	El cliente puede regar de forma manual las plantas
Actor	**Cliente, Raspberry Pi**
precondición	El cliente debe tener conexión a la Raspberry Pi por medio del sistema
Flujo Principal	1. El cliente ve el estado de humedad de la planta y decide regar la planta 2. Si la humedad de la planta es menor al 100% esta mandará una señal de activación manual a la Raspberry Pi. 1.2 <<include>> Regar 3. El sistema notifica al cliente que su riego manual fue realizado
Flujo Alternativo	3. Si la humedad de la planta es 100% esta no se regará y se enviará un aviso al cliente.
Postcondición	Las plantas se habrán regado de forma manual

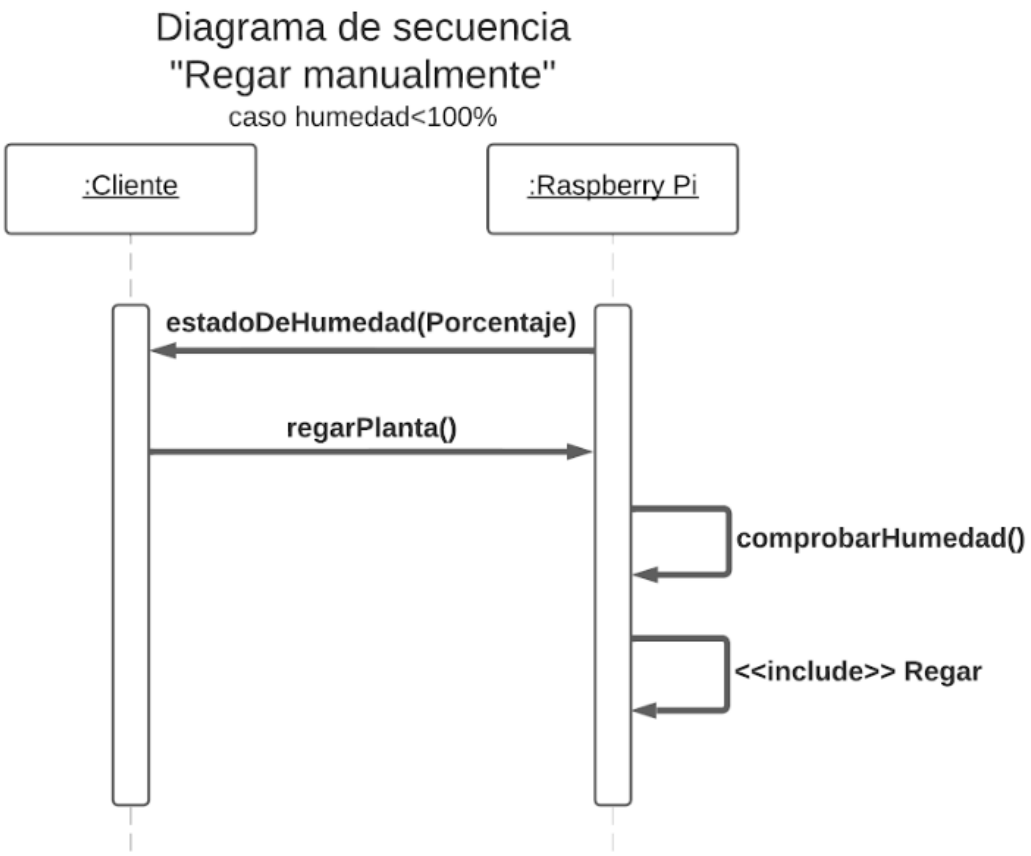


Diagrama de secuencia
"Regar manualmente"
caso humedad<100%
:Cliente
:Raspberry Pi
estadoDeHumedad(Porcentaje)
regarPlanta()
comprobarHumedad()
<<include>> Regar

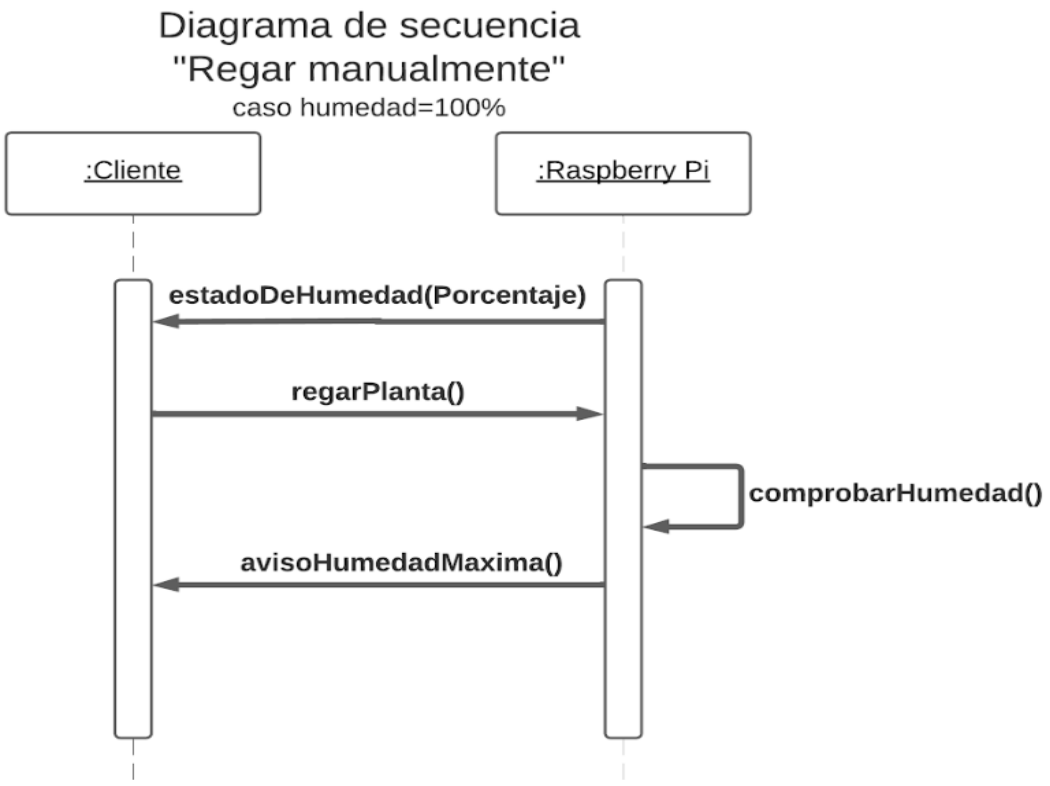


Diagrama de secuencia
"Regar manualmente"
caso humedad=100%
:Cliente
:Raspberry Pi
estadoDeHumedad(Porcentaje)
regarPlanta()
comprobarHumedad()
avisoHumedadMaxima()

Caso de uso	Programar riego automático
Descripción	El cliente agenda horarios en los cuales se efectúan riegos a lo largo del día
Actor	**Cliente**
Flujo Principal	1. El cliente indica al sistema una hora del día en la cual regar las plantas 2. El sistema verifica que haya un intervalo de más de una hora de diferencia hacia adelante y hacia atrás del horario escogido 2.1 De existir el intervalo de tiempo adecuado, agenda el horario elegido por el cliente para el regado automático 2.2 De no existir el intervalo, indica al cliente que ingrese otro horario
Flujo Alternativo	2. Si la hora ya está agendada, el sistema indica al cliente que ya existe ese horario
Postcondición	El riego queda agendado para ser utilizado

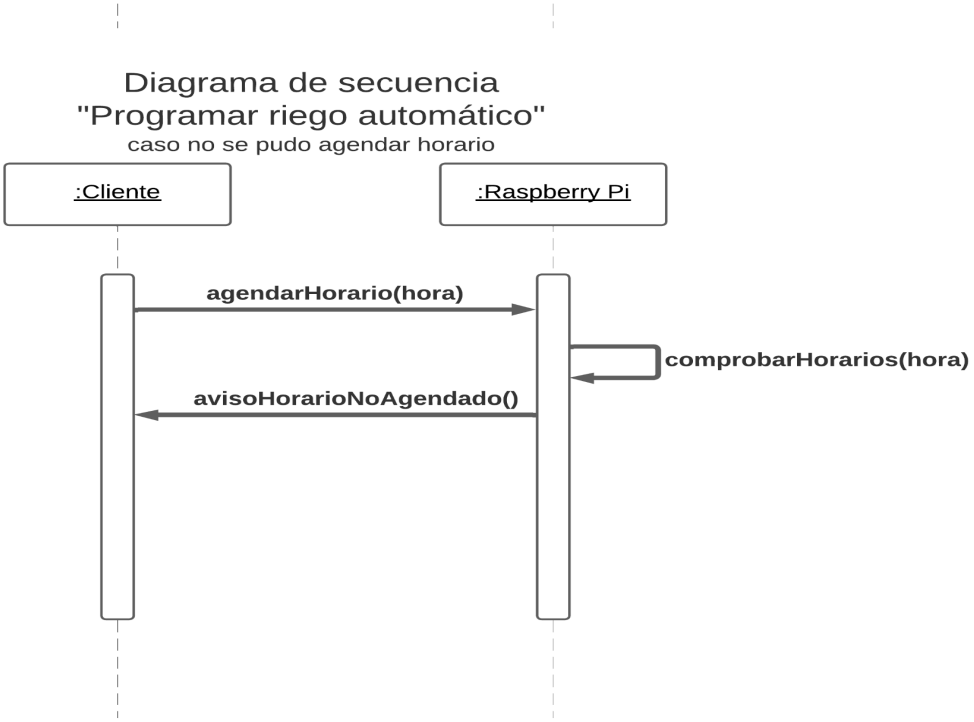


Diagrama de secuencia
"Programar riego automático"
caso no se pudo agendar horario
:Cliente
:Raspberry Pi
agendarHorario(hora)
comprobarHorarios(hora)
avisoHorarioNoAgendado()

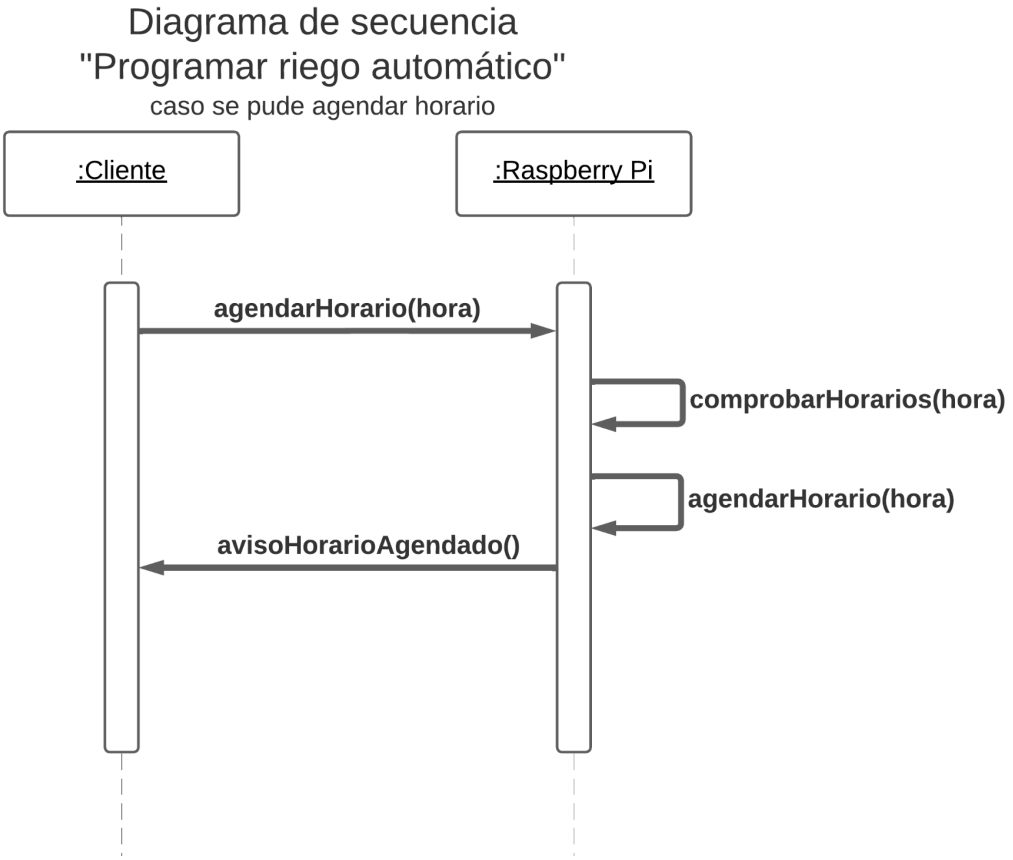
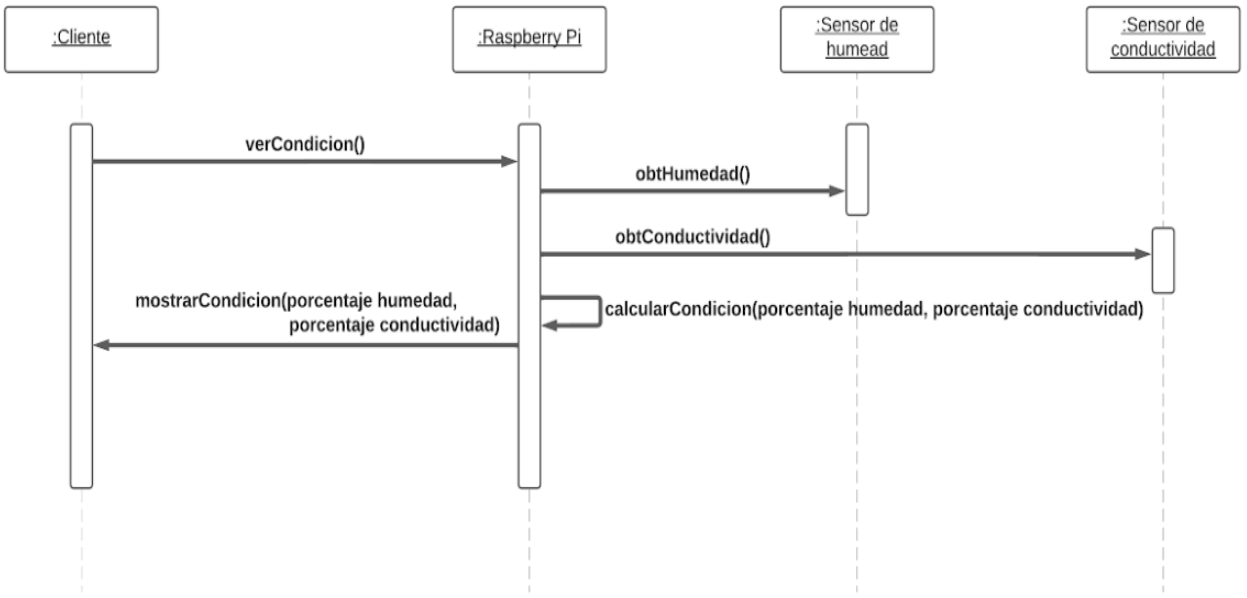


Diagrama de secuencia
"Programar riego automático"
caso se pude agendar horario
:Cliente
:Raspberry Pi
agendarHorario(hora)
comprobarHorarios(hora)
agendarHorario(hora)
avisoHorarioAgendado()

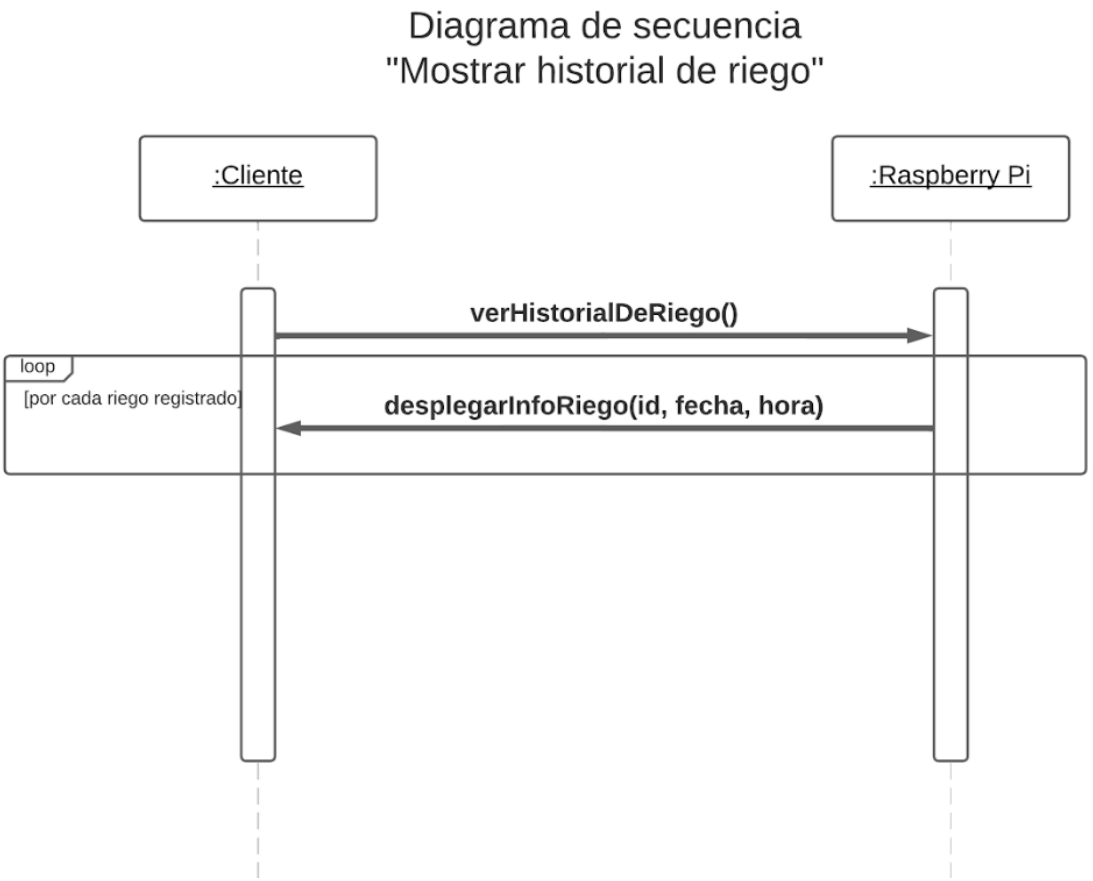
Caso de uso	Regar automáticamente
Descripción	Las plantas son regadas automáticamente y se le da aviso al cliente.
Actor	**Cliente, Raspberry Pi**
precondición	El cliente debe tener conexión a la Raspberry Pi por medio del sistema y tener riegos programados previamente
Flujo Principal	1. Cuando se recibe una señal de activación automática correspondiente al horario en el cual está agendado un riego automático en la Raspberry Pi, se activan los mecanismos de riego hasta que la humedad llegue al 100% 2. <<include>> Regar 3. El sistema notifica al cliente que su riego automático fue realizado
Postcondición	Las plantas se habrán regado de forma automática

Caso de uso	Mostrar condiciones del suelo
Descripción	El cliente puede revisar las condiciones del suelo en tiempo real
Actor	**Cliente, Raspberry Pi**
Flujo Principal	1. El cliente solicita visualizar estadísticas sobre el suelo en el cual se efectúan los regados, estos datos son recopilados por el sensor de humedad y sensor de conductividad. 2. El sistema muestra al cliente el porcentaje de humedad y porcentaje de conductividad del suelo

Diagrama de secuencia
"Mostrar condiciones de suelo"



Caso de uso	Mostrar historial de riegos
Descripción	El cliente puede ver los riegos que se han hecho con anterioridad.
Actor	**Cliente**
precondición	Se debe tener almacenada la información de los riegos.
Flujo Principal	1. El cliente puede ver el historial de riego de sus plantas. 2. Por cada riego efectuado se muestra el id, fecha y hora de realización.



5.1.3 Descripción de la arquitectura

Una vez consolidada la idea, es necesario establecer cómo será el proceso de conexión entre el usuario final y el modelo de negocio. En este caso se ha optado por una implementación usando el paradigma cliente-servidor, donde se encuentran las siguientes tecnologías:

Cliente:
El usuario usará su dispositivo móvil con sistema operativo Android. La aplicación del sistema está escrita en Kotlin, lenguaje de programación para la creación de aplicaciones nativas en Android.

Servidor:
El servidor está compuesto principalmente por la Raspberry Pi, sobre ella se ejecuta un software que funciona como intermediario entre el hardware y el usuario. Se usan las librerías de Python: Django y RPI.GPIO, para el backend como para la manipulación de entradas y salidas GPIO respectivamente

Ambos apartados tanto cliente como servidor deben estar conectados a una red Wi-Fi o en su defecto, una conexión Ethernet.

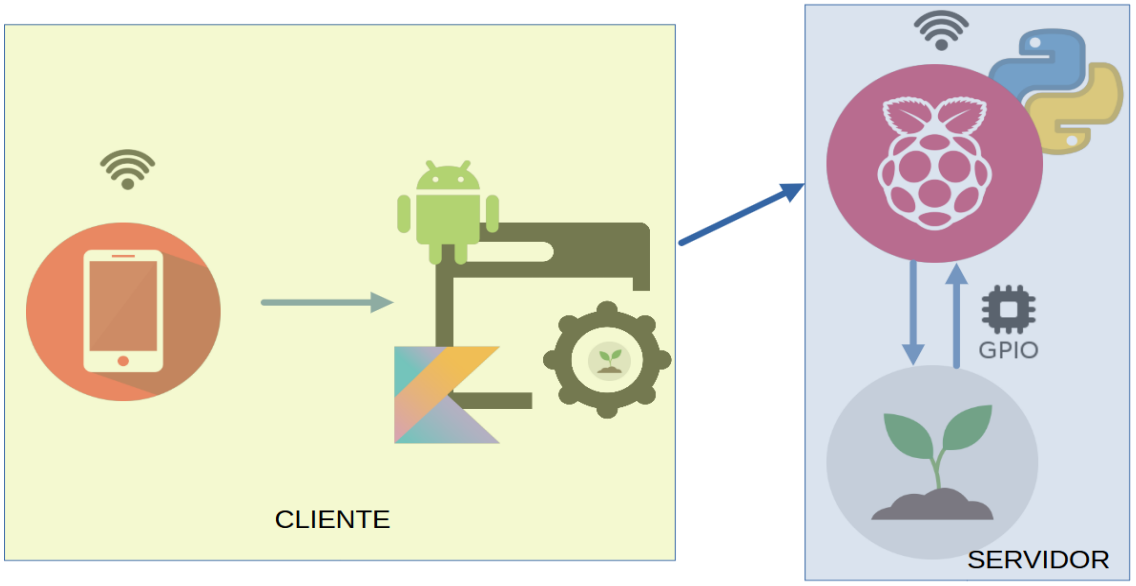
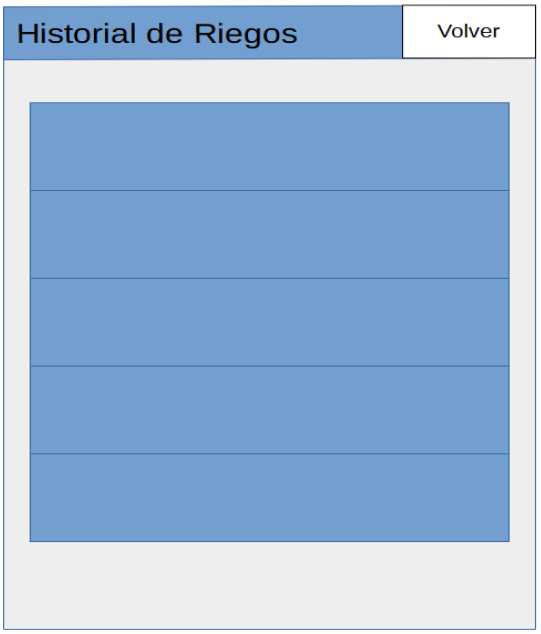


Figura 4: Descripción de la Arquitectura

6. Diseño de interfaces de Usuario

Debido al enfoque asistencial y destinado a personas pertenecientes a la tercera edad en su mayoría. Se ha optado por la realización de 3 vistas sencillas: Vista principal, Historial de Riego, y Ver Estadísticas

Figura 5: Vista Principal



Diseño de GUI:
Historial de Riego

Figura 6: Historial de Riego

Diseño de GUI:
Ver Estadísticas

Figura 7: Ver Estadísticas

7. Plan de Integración

El objetivo de este producto es lograr que el usuario frecuente la aplicación para conocer el estado actual del suelo de sus plantas, es por ello que la interfaz debe ser intuitiva y amigable, sin muchas opciones, sólo con las necesarias. Los sensores de entrada y salida (relé y sensor de agua) deben estar conectados a la Raspberry, la cual deberá estar encendida todo el tiempo. Es ideal que tanto la Raspberry como los circuitos integrados al GPIO estén aisladas y en un lugar seguro por el inminente peligro de cortocircuito provocado por el agua.

8. Modelo de Implementación

El desarrollo de la aplicación es producido una sola vez, luego puede ser instalado en varios dispositivos móviles. Sin embargo, para implementar el proyecto se debe contar con los implementos necesarios para el cliente. Esto significa que la mayoría de los costos serán asociados al hardware. El enfoque del proyecto y del producto permite al cliente final elegir cómo será implementado su sistema de cañerías y la disposición del suelo a utilizar (macetas con suelo compartido, una sola planta u otras variantes).

9. Módulos Implementados

Dada la arquitectura del modelo de negocio, es posible dedicar las siguientes herramientas/lenguajes/librerías a cada módulo:

Aplicación Móvil	Kotlin (Jetpack Compose)
Módulo HTTP para Android	Kotlin (RetroFit y OkHTTP)
API de envío de datos	Python (Django)
Servidor	Python (Django)

Tabla 5: Módulos Implementados

9.1 Aplicación Móvil

El desarrollo de aplicaciones móviles por parte de Android Studio ha sido más accesible gracias a la aparición de Jetpack Compose, un framework de enfoque declarativo que optimiza el desarrollo de vistas complejas.

Una parte fundamental del framework es la clase MainActivity, dentro de la cual pueden colgarse los demás componentes de la aplicación. Para el caso de múltiples vistas, se usa el artefacto Navigator Host, el cual permite la navegación por medio de distintos botones.

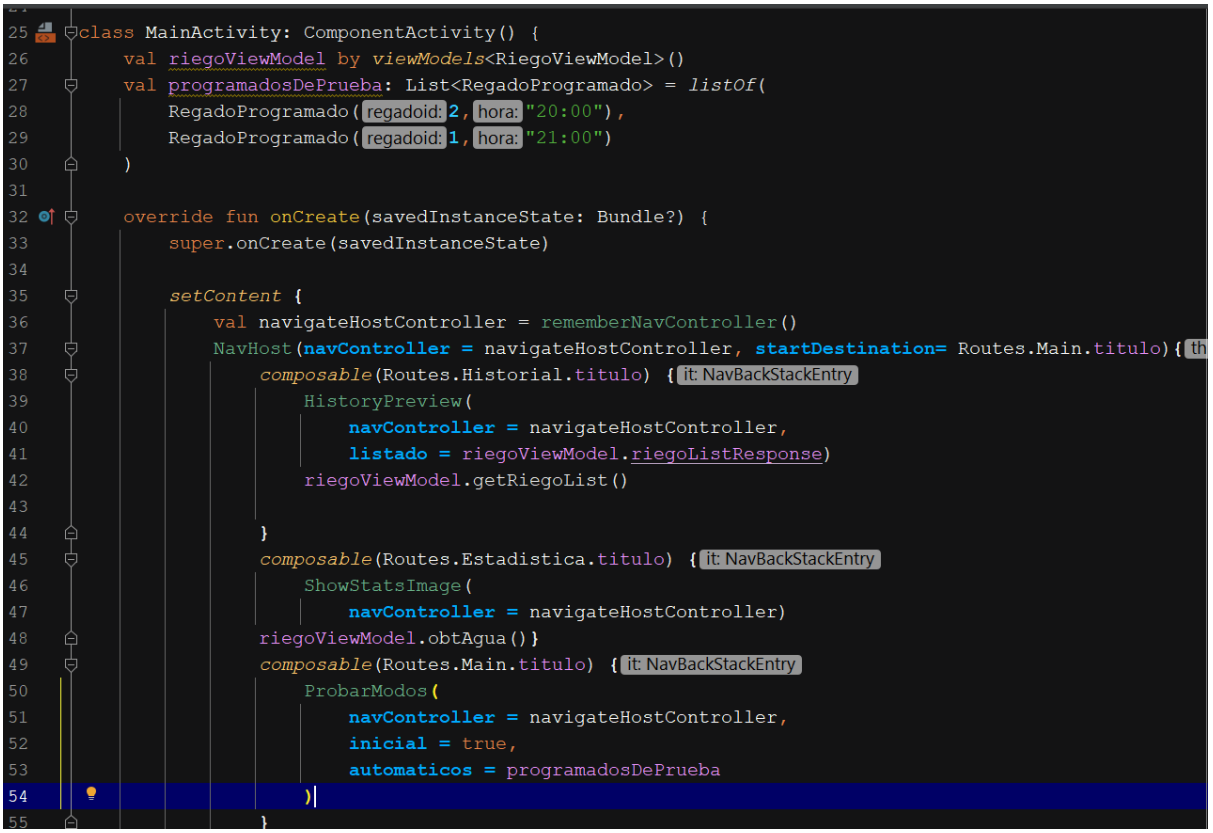


Figura 8: Class Main dentro de la aplicación

Se puede observar que existen las opciones Historial, Estadística y Probar Modos, este último cambia entre modo manual y automático con una variable booleana guardada en el caché.

```
286
287  @Composable
288  fun ProbarModos(
289      navController: NavHostController,
290      inicial: Boolean,
291      automaticos: List<RegadoProgramado>
292  ) {
293      var manual by rememberSaveable { mutableStateOf(switcheable.get()) }
294
295      if (manual) {
296          Column { this: ColumnScope
297              ModoManual(navController = navController)
298              Button(onClick = { manual = switcheable.switch() }) { this: RowScope
299                  Text(text = "Cambiar a modo automático")
300              }
301
302          }
303
304
305      } else {
306          Column { this: ColumnScope
307              ModoAutomatico(navController = navController, automaticos = automaticos)
308              Button(onClick = { manual = switcheable.switch() }) { this: RowScope
309                  Text(text = "Cambiar a modo manual")
310              }
311          }
312      }
313  }
314
315
```

Figura 9: Cambiar Modos manual o automático.

```
186  @Composable
187  fun ModoManual(navController: NavHostController) {
188      BotaniDripTheme {
189          Column(modifier = Modifier.fillMaxWidth()) { this: ColumnScope
190              ShowTitle(title = "Modo Manual")
191              Spacer(modifier = Modifier.padding(20.dp))
192
193              Box(modifier = Modifier.fillMaxWidth()) { this: BoxScope
194                  Text(
195                      text = "Bienvenido a BotaniDrip",
196                      fontSize = 30.sp,
197                      fontWeight = FontWeight.Bold,
198                      modifier = Modifier.align(alignment = Alignment.Center)
199                  )
200              }
201              Spacer(modifier = Modifier.padding(20.dp))
202              Box(modifier = Modifier.fillMaxWidth()) { this: BoxScope
203                  BotonHistorial(
204                      mod = Modifier.align(alignment = Alignment.Center),
205                      nav = navController
206                  )
207              }
208              Spacer(modifier = Modifier.padding(20.dp))
209              Box(modifier = Modifier.fillMaxWidth()) { this: BoxScope
210                  BotonEstadistica(
211                      mod = Modifier.align(alignment = Alignment.Center),
212                      nav = navController
213                  )
214              }
215              Spacer(modifier = Modifier.padding(40.dp))
216              Box(
```

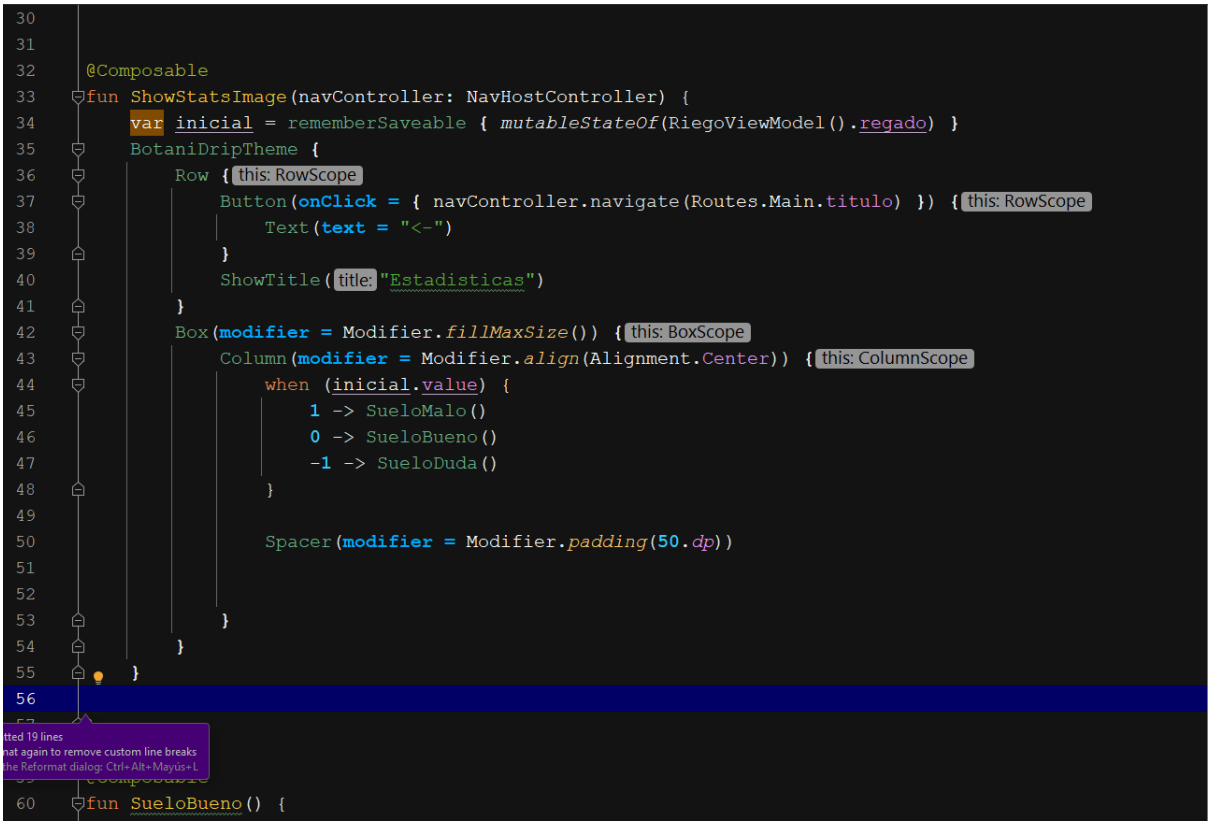
Figura 10: Código de Modo Manual

```
285
286
287  @Composable
288  fun ProbarModos(
289      navController: NavHostController,
290      inicial: Boolean,
291      automaticos: List<RegadoProgramado>
292  ) {
293      var manual by rememberSaveable { mutableStateOf(switcheable.get()) }
294
295      if (manual) {
296          Column { this: ColumnScope
297              ModoManual(navController = navController)
298              Button(onClick = { manual = switcheable.switch() }) { this: RowScope
299                  Text(text = "Obtener Ip")
300              }
301
302          }
303
304
305      } else {
306          Column { this: ColumnScope
307              ModoAutomatico(navController = navController, automaticos = automaticos)
308              Button(onClick = { manual = switcheable.switch() }) { this: RowScope
309                  Text(text = "Obtener Ip")
310              }
311          }
312      }
313  }
314
```

Figura 11: Selector de Vistas de los Modos Manual o Automático

```
108
109
110  @Composable
111  fun HistoryPreview(navController: NavHostController, listado: List<RiegoPrueba>) {
112      Column { this: ColumnScope
113          BotaniDripTheme {
114
115              Row { this: RowScope
116                  Button(onClick = { navController.navigate(Routes.Main.titulo) }) { this: RowScope
117                      Text(text = "<-")
118                  }
119                  ShowTitle( title: "Historial de Riego")
120              }
121              Surface(
122                  modifier = Modifier.fillMaxWidth(),
123                  color = MaterialTheme.colorScheme.background
124              ) {
125                  Box(modifier = Modifier.padding(5.dp)) { this: BoxScope
126                      ShowRiegoHistory(riegos = listado)
127
128                  }
129
130              }
131
132          }
133      }
134
135  }
```

Figura 12: Vista del Historial de Riegos



```
30
31
32  @Composable
33  fun ShowStatsImage(navController: NavHostController) {
34      var inicial = rememberSaveable { mutableStateOf(RiegoViewModel().regado) }
35      BotaniDripTheme {
36          Row { this: RowScope
37              Button(onClick = { navController.navigate(Routes.Main.titulo) }) { this: RowScope
38                  Text(text = "<-")
39              }
40              ShowTitle( title: "Estadisticas")
41          }
42          Box(modifier = Modifier.fillMaxSize()) { this: BoxScope
43              Column(modifier = Modifier.align(Alignment.Center)) { this: ColumnScope
44                  when (inicial.value) {
45                      1 -> SueloMalo()
46                      0 -> SueloBueno()
47                      -1 -> SueloDuda()
48                  }
49
50                  Spacer(modifier = Modifier.padding(50.dp))
51
52
53              }
54          }
55      }
56
tted 19 lines
nat again to remove custom line breaks
he Reformat dialog: Ctrl+Alt+Mayús+L
60  fun SueloBueno() {
```

Figura 13: Vista de Estadísticas

9.2 Módulo HTTP para Android

Para conocer y recibir los datos provenientes del servidor. Kotlin dispone de la librería Retrofit y OkHTTP para Android, la cual permite que la aplicación pueda realizar peticiones básicas del protocolo HTTP por medio de funciones. Esto es posible gracias a la codificación de una interfaz API Service, que contiene los métodos a implementar por parte de un modelo (ViewModel). La API es construida gracias a un objeto compañero (característico de Kotlin), el cual tiene un atributo para guardar la ip del servidor.

```
6
7   interface ApiService {
8
9       @GET("/riegos/")
10      suspend fun getRiegos(): List<RiegoPrueba>
11
12      @GET("/programados/")
13      suspend fun getRegadosProgramados(): List<RegadoProgramado>
14
15      @GET("/rm1/")
16      suspend fun regarManual(): Int
17
18      @GET("/ra1/")
19      suspend fun regarAutomatico(): Int
20
21      @GET("/ws1/")
22      suspend fun obtenerEstadoAgua(): Int
23
24
25      companion object {
26          var apiService: ApiService? = null
27          var ip: String = "http://192.168.70.158:8080"
28
29
30          fun getInstance(): ApiService {
31              if (apiService == null) {
32                  apiService = Retrofit.Builder() Retrofit.Builder
33                      .baseUrl(ip) Retrofit.Builder
34                      .addConverterFactory(GsonConverterFactory.create())
35                      .build().create(ApiService::class.java)
36              }
```

Figura 14: API Service que contiene métodos para realizar peticiones.

Un ViewModel es el encargado de implementar los métodos de una interfaz API Service, en este caso, el viewmodel es llamado RiegoViewModel, maneja los campos para controlar el listado de riegos ya hechos, el estado del agua y un código de comprobación de regado exitoso.

```
11
12  class RiegoViewModel: ViewModel() {
13      var riegoListResponse: List<RiegoPrueba> by mutableStateOf(listOf())
14      var regadosProgramados: List<RegadoProgramado> by mutableStateOf(listOf())
15      var regado: Int by mutableStateOf( value: -1)
16      var hayAgua: Boolean by mutableStateOf( value: false)
17      var errorMessage: String by mutableStateOf( value: "")
18
19      fun getRiegoList(){
20          viewModelScope.launch { this: CoroutineScope
21              val apiService = ApiService.getInstance()
22              try {
23                  riegoListResponse = apiService.getRiegos()
24              }
25              catch (e: Exception){
26                  errorMessage = e.message.toString()
27              }
28          }
29      }
30      fun getRegadosProgramadosList(){
31          viewModelScope.launch { this: CoroutineScope
32              val apiService = ApiService.getInstance()
33              try {
34                  regadosProgramados = apiService.getRegadosProgramados()
35              }
36              catch (e: Exception){
37                  errorMessage = e.message.toString()
38              }
39          }
40      }
41      fun obtAgua() {
```

Figura 15: RiegoViewModel con atributos y algunos de sus métodos

Gracias a estos dos artefactos (ViewModel y Api Service) es posible asignar a variables los valores obtenidos de una petición HTTP. Cabe recalcar que para obtener una actualización de los datos es necesario realizar otra llamada al servidor, el cual está siempre escuchando y esperando peticiones.

9.3 API de Envío de Datos

Los datos guardados en el servidor son la información de todos los riegos hechos, guardando su id único, fecha , hora y si ha sido por acción manual o automática.

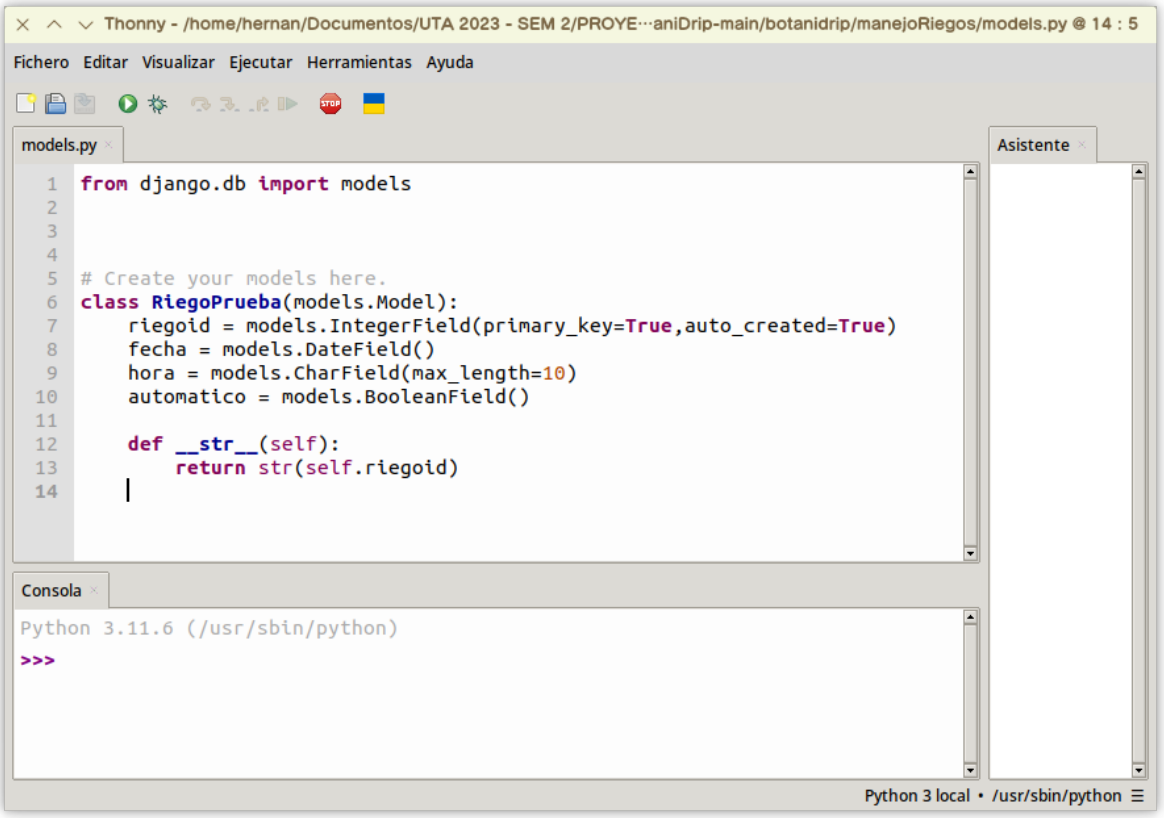


Figura 16: RiegoPrueba Model.

Para enviar los datos del servidor desde una base de datos a un cliente final es necesario convertir el formato de tablas y relacional de los datos, a uno que la mayoría de tecnologías puedan entender. Para ello, se ha realizado un conversor sencillo que transforma la información de una base de datos al formato estándar JSON, siendo este reconocible para la API Service de la Aplicación. Finalmente, la salida es puesta a disposición de un objeto HttpResponse, el cual puede ser procesado por quien realiza la petición.

```
30
31
32  # Create your views here.
33  def Pruebas(request):
34      return HttpResponse('AGREGADO')
35
36  def getRiegos(request):
37      salida = "["
38      for r in RiegoPrueba.objects.all():
39          salida+="{"
40          salida+=f"\"riegoid\":{r.riegoid}, \"fecha\": \"{r.fecha}\", \"hora\": \"{r.hora}\", \"automatico\": {int(r.automatico)}"
41          salida+="},"
42      salida = salida[:-1] + "]"
43      return HttpResponse(salida)
44
45
```

Figura 17: Envío de datos en formato JSON

9.4 Servidor

El servidor está alojado en la Raspberry Pi, y está encargado de recibir la información de los sensores conectados directamente al GPIO de la máquina. Se han usado un total de 4 pines:

Número de Pin	Nombre	Funcionalidad
4	5V PWR	5V de potencia
6	GND	Pin de Tierra
40	GPIO 21	Pin de bomba (output)
37	GPIO 26	Pin de Sensor (input)

Tabla 6: Numeración, Nombre y Funcionalidad de pines utilizados

Además, es necesario programar los encendidos y apagados de cada pin, para ello se han implementado las funciones de iniciado de los sensores:

modules.py

```
import RPi.GPIO as GPIO
import time



SENSOR_AGUA=26
RELAY=21
CONDICION = False
SEGUNDOS = 3


def init_sensor():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(SENSOR_AGUA,GPIO.IN)


def init_components():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(SENSOR_AGUA, GPIO.IN)
    GPIO.setup(RELAY, GPIO.OUT)

```

Asistente

Figura 18 : Implementación de pines e inicializador de componentes

Se han desarrollado las funciones de regado hasta húmedo y regado con tiempo, por ahora dentro de la aplicación sólo se utiliza la primera mencionada.

Figura 19: Funciones de regado

Para responder y accionar las funciones de regado, el servidor debe crear un hilo asíncrono, el cual finalizará en cuanto termine el regado. Esto ha sido implementado de esa forma con el fin de que la respuesta al cliente sea justo luego de dar la orden. El servidor está encargado de terminar el hilo, deshabilitando la bomba de agua en cuanto lo estime conveniente.

```
45
46  def accionarRegado(request):
47      try:
48          hilo = th.Thread(name="riiego",target=regadoManual)
49
50          hilo.start()
51          r = RiegoPrueba(getNewID(),getDate(),getHour(),False)
52          r.save()
53
54      except:
55          return HttpResponse(-1)
56
57      return HttpResponse(1)
58
59
```

Figura 20: Response al cliente con creación de hilo

Para obtener el estado actual del agua encapsulado en una respuesta HTTP se ha creado una función de iguales características a la ya mencionada, pero con el fin de retornar el estado del agua en el instante que se realice la petición. Esta función es necesaria para los casos en que el cliente desee saber el estado del suelo dentro de la aplicación.

```
59
60  def obtEstadoAgua(request):
61      try:
62          hilo = th.Thread(name="compruena",target=comprobarHumedad)
63          hilo.start()
64      except:
65          return HttpResponse(-1)
66      return HttpResponse(GPIO.input(SENSOR_AGUA))
67
68
```

Figura 21: Función para obtener el estado del suelo

Para un correcto funcionamiento de las funciones de este módulo, fue necesaria la creación de métodos auxiliares como por ejemplo, la creación de ID únicas para los regados y obtener la fecha y hora actual del sistema.

Figura 22: Métodos auxiliares

10. Comparativa de Requerimientos

Una buena forma de medir el desempeño del grupo es comparar lo propuesto en los requerimientos con el producto final, la siguiente tabla lo describe:

Requerimiento	Cumplido	No Cumplido	Alternativa
Conexión entre usuario y raspberry por medio de Wifi	X		
Cliente debe encontrarse en el dispositivo móvil del usuario	X		
El servidor debe estar alojado en la Raspberry Pi	X		
Efectuar riegos automáticamente por medio de accionamiento manual	X		
Efectuar riegos automáticamente previamente programados por el usuario		X	No ha sido implementado por requerir conocimientos más elevados de BroadCasting y manejo de alarmas
Recopilar y desplegar estadísticas sobre las condiciones del suelo	X		
Notificar al usuario cuando se haya efectuado un riego	X		
Almacenar información de Riegos efectuados	X		
Visualizar listado de Riegos efectuados	X		

Tabla 7: Comparativa de Requerimientos

Analizando la tabla, es destacable decir que el regado por accionamiento manual es mucho más necesario para el prototipo final que el automático, ya que este último estaba enfocado en casos donde el actor principal no era perteneciente al público objetivo.

11. Trabajo Futuro

Dado el resultado de la comparativa, evidentemente el objetivo principal será dar por cumplido el apartado de la funcionalidad de riego programado por alarma, esto involucra aprender y adquirir conocimientos sobre el manejo de alarmas y el funcionamiento del modo de operar de los sistemas operativos móviles. Además, queda pendiente mejorar la seguridad de conexión entre el cliente y el servidor, siendo el formato de la conexión HTTPS y no HTTP como lo es actualmente.

Como desafío, también está la inquietud de realizar una implementación de la aplicación en dispositivos con sistema operativo IOS, ya que un porcentaje no menor de adultos mayores usan estas tecnologías, quedando fuera del producto final por el momento.

12. Conclusiones

Se ha optado por brindar una solución concisa e implementable a un problema específico. Se espera que el reconocimiento de posibles riesgos y factores de riesgos acompañado de sus respectivas acciones remediales permitan que frente a cualquier problema presentado, el grupo de trabajo será capaz de enfrentarlo, aplicando así principios de ingeniería en el desarrollo de cada miembro como estudiante, y con ello, preparándose para el mundo laboral.

La descripción de la arquitectura del producto es de vital importancia para la fase de implementación, ya que traza las directrices a seguir para cumplir a cabalidad con la planeación del grupo.

La aplicación móvil está implementada casi en su totalidad, cumpliendo con las funcionalidades básicas necesarias para un uso regular. Se logró dar con una interfaz simple. Por otra parte, el sistema de sensores y hardware fue implementado con éxito y en su totalidad, siendo flexible a las preferencias del usuario final. Es por esto último que Botani Drip tiene la capacidad de ser implementado a pequeña escala (un solo sensor de agua) y de la misma manera a gran escala (múltiples sensores).

Además de la implementación, la mayoría de lo propuesto en la Carta Gantt ha sido cumplido con un pequeño desfase de una semana.

Como observación personal, la experiencia y los conocimientos adquiridos durante la creación del prototipo y producto final han sido muy enriquecedoras para el grupo.

13. Referencias (formato IEEE)

[1] Teleasistencia. (s. f.). Beneficios de la jardinería para personas mayores. Recuperado de https://teleasistencia.es/es/blog/ocio-en-la-tercera-edad/beneficios-de-la-jardineria-para-personas-mayores

[2] Raspberry Pi. (s. f.). Raspberry Pi 4. Recuperado de https://raspberrypi.cl/raspberry-pi-4/

[3] MCI Electronics. (s. f.). Raspberry Pi 4 Modelo B en el mercado. Recuperado de https://mcielectronics.cl/shop/product/raspberry-pi-4-modelo-b-8gb-ram-raspberry-pi-28296

[4] Android Developers. (s. f.). Android Studio (Jetpack Compose). Recuperado de https://developer.android.com/jetpack/compose?hl=es-419

[5] Repositorio de Github. Python y Kotlin Recuperado de

https://github.com/TLRP-45/BotaniDrip.git