Descripción Arquitectónica

SOSEMADO: Software de Servicios de Mantenimiento a Domicilio

Desarrollo del Software 2020 - 2021 ETSIIT UGR

UNIVERSIDAD DE GRANADA

Ahmed El Moukhtari Koubaa
Iván Valero Rodríguez

Descripción del sistema

SOSEMADO es un Sistema Software para realizar tareas de Mantenimiento a Domicilio, donde un Cliente crea un trabajo, y ese trabajo se asigna a un Técnico.

Si este lo acepta, mandaría su presupuesto con un precio determinado. En caso contrario, se le reasigna a otro técnico, y así hasta que alguien lo elija o todos rechacen.

.

Para la selección del Técnico, se usan parámetros como la valoración de los clientes y su tipo de especialidad (albañilería, fontanería...)

Se habilitará un inicio de sesión y un registro para permitir el acceso privado a los trabajos por parte de técnicos y clientes del sistema.

Análisis de requisitos

Requisitos funcionales	Requisitos no funcionales
- **Dar alta cliente**: Se permitirá que un cliente se registre en nuestro sistema, para ello necesitará introducir algunos datos personales como: nombre, apellidos, domicilio, etc - **Cliente define trabajo**: un cliente previamente registrado es capaz de crear un trabajo, es decir, definirá un problema (relacionado con el mantenimiento de su hogar o relacionado) que constará de una descripción, una localización, el tipo (por ejemplo, es un problema de fontanería, carpintería, etc) - **Dar de baja al cliente**: un cliente que se registró previamente se podrá dar de baja, es decir, eliminar su registro de nuestro sistema. Esta es una operación delicada y que se realizará de manera más segura - **Modificar trabajo**: un trabajo previamente creado por un cliente podrá ser modificado, es decir,	- **Darse de alta es un proceso fácil y sencillo, toma menos de 5 minutos**: los clientes de nuestro sistema serán personas adultas que tengan un problema que necesita una solución rápida, no obstante, dichos clientes tienen que darse de alta antes de que puedan hacer uso de nuestros servicios. Para garantizar una experiencia de usuario satisfactoria el proceso de alta será lo más simple y claro posible. Podría basarse en un formulario donde se solicite, al menos, el nombre y la dirección de correo electrónico. - **Se mantendrá un registro de los trabajos:** los trabajos que se definan en el sistema deben quedar registrados en nuestro sistema, tanto si están pendientes por hacerse como si se tienen que hacer. Podríamos crear una tabla en nuestra base de datos dedicada íntegramente a los trabajos.

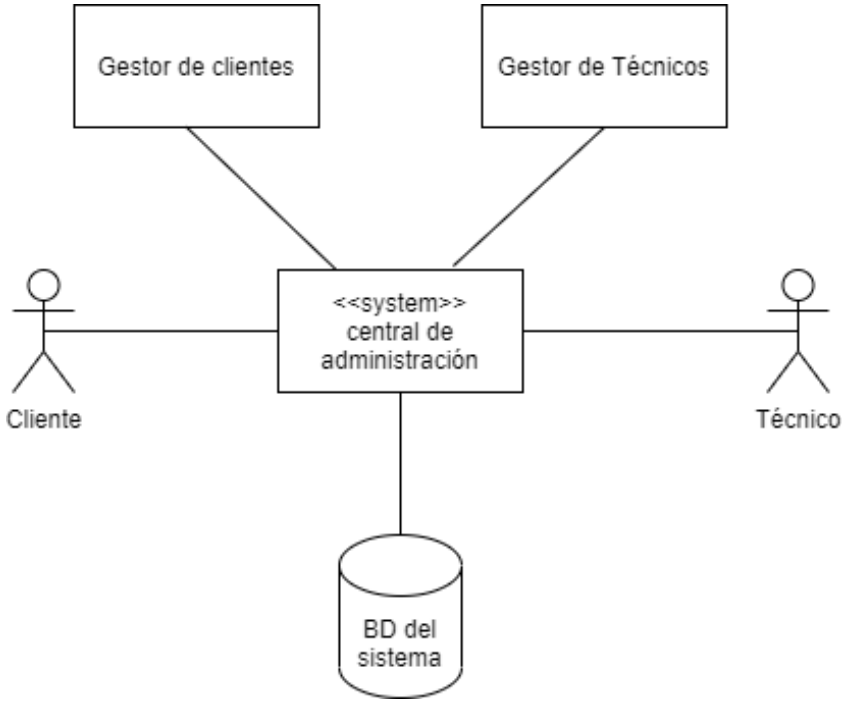
algunos de sus atributos cambiarán obteniendo nuevos valores - **Cancelar trabajo creado**: un trabajo que fue previamente creado por un cliente y enviado al sistema podrá ser cancelado (por el cliente o por los administradores del sistema) - **Asignar trabajo dinámicamente**: la función principal de nuestro sistema es su capacidad de asignar a un trabajo (definido por un cliente) un técnico capaz de realizarlo. Para eso hacemos uso de la asignación de trabajo dinámico, con esta función se busca en la base de datos quienes son los técnicos mejores valorados y más disponibles, entonces les ofrecerá un nuevo trabajo para que lo realicen, los técnicos deberán responder si desean realizar el trabajo o no y en caso afirmativo ofrecer un presupuesto.	- **La interfaz del cliente estará hecha en Ruby on Rails:** el uso de ruby on rails nos facilita la creación de una aplicación REST, además de ser un simple método de implementar el patrón MVC. Los clientes podrán gestionar así más claramente sus datos y trabajos. - **La base de datos usada será relacional:** por definición en nuestro sistema se establecen varias relaciones entre diferentes entidades, por ejemplo, los clientes definen trabajos, los técnicos los realizan, el sistema los asigna, los clientes podrían querer valorar a técnicos relacionados con los trabajos, etc. Parece entonces que el modelo que más nos convendría para nuestra base de datos sería el relacional. - **La base de datos será MySQL:** dicho software de gestión de base de datos responde estupendamente a las necesidades de nuestro sistema. Además, los desarrolladores se encuentran habituados a dicho software y lo manejan perfectamente.

Los interesados y sus inquietudes

Partes interesadas	Intereses
Cliente (customer)	- Resolver su problema lo antes posible Encontrar un técnico de calidad - Una oferta competitiva

Técnico (realizadores de trabajos)	- Conseguir nuevos clientes - Tener presencia en el mundo digital - Beneficio económico - Asignación automática de trabajos
Administradores de empresa	- Beneficio económico (por publicidad) - Captar usuarios - Expandir la marca - Hacer contratos con agencias privadas - Expandirse ofreciendo nuevos servicios
Otras empresas del sector	- Expandirse a través de nuestra marca - Llegar a nuevos usuarios - Nuevas tecnologías para publicidad
Organizaciones públicas	- Beneficio económico (impuestos) - Colaboraciones con empresas (a través de nuestro sistema)

Diagrama de arquitectura del sistema



Podemos ver en nuestro diagrama que con nuestro sistema van a interactuar dos actores principalmente, clientes y técnicos.

Disponemos de un “gestor de clientes”, el cual se encargará de gestionar toda la información y las acciones relacionadas con los clientes, lo mismo ocurre con los técnicos.

Nuestro sistema dispone de una base de datos en la que se almacenará toda la información que se genere en el sistema y se llevarán a cabo los registros necesarios.

Por último, nuestro sistema funciona como administrador general, se encarga de enlazar los intereses de los clientes con los de los técnicos mediante los gestores correspondientes y hacer uso de la base de datos para almacenar y extraer los datos oportunos.

Criterios de calidad a partir de las distintas perspectivas

Seguridad

Una de las grandes inquietudes del sistema por parte, sobre todo, de los clientes, es la protección de la Información. De forma paralela, por la Ley Orgánica de Protección de Datos, se pide que se trate de proteger la información personal de los Clientes y Técnicos, ya que si la información se filtrara se podría llevar a vulnerar sus derechos (y causaría problemas legales).

Por ello,

- La contraseña se encriptará usando un algoritmo de hash como MD5, SHA… (si bien no debería dar perjuicio con respecto a la aplicación para técnicos)
- Debería darse la información justa y necesaria para realizar el Trabajo, o mostrar la información relevante de contacto.

Desempeño

El desempeño del sistema implica que se permite realizar las operaciones con relativa rapidez, ya que a los usuarios, tanto clientes como técnicos, no les interesa esperar. Por ello, no debería tardarse más de 10 segundos en las operaciones más complejas, y sobre todo, cumplir con lo comentado en los requisitos no funcionales.

Disponibilidad

Otro de los grandes puntos del sistema a tener en cuenta de cara a realizar criterios de calidad es la disponibilidad, es decir, cómo tolerará los errores y cómo dejar que el sistema esté funcionando el mayor tiempo posible. Por ello:

- El sistema web se desplegará en el servidor ofrecido para realizar las prácticas.
- Se trabajará lo máximo posible en local antes de subirlo al servidor.

Evolución

Si bien el proyecto no daría para hacer una mayor evolución, se ha pensado en sistemas alternativos de asignación de técnicos a trabajos, como la subasta por el menor precio. En parte, también se trata del sistema que se ha ido ideando durante las anteriores prácticas.

Integridad

Es importante para nuestro proyecto porque nuestros datos deben ser correctamente guardados y accedidos, sobre todo las credenciales, las cuales en parte se cifran para evitar que en el peor de los casos no pudiesen ser usadas por actores ajenos a nuestro sistema o bien llegar a comprender el contenido de estas.

Diagrama de clases de análisis

A raíz de todo lo expuesto anteriormente, se ha decidido crear el siguiente diagrama de clases. Implementamos el patrón MVC por lo que nos encontramos con un 3 bloques principalmente, el modelo, la vista y el controlador.

En el modelo tenemos todas las estructuras de datos necesarias para almacenar y gestionar la información de los clientes, técnicos y trabajos. Cabe destacar que también disponemos de una clase “BaseDatos”, aportará abstracción en cuanto al tipo de base de datos, las consultas, la lógica de la misma, etc. Dicha clase implementará los métodos necesarios para extraer e introducir datos a la “BD”.

El controlador hace uso del modelo descrito anteriormente bajo demanda de la vista. Será capaz de invocar a “BaseDatos” y obtener estructuras de datos con la información necesaria para procesarla o mostrarla por pantalla.

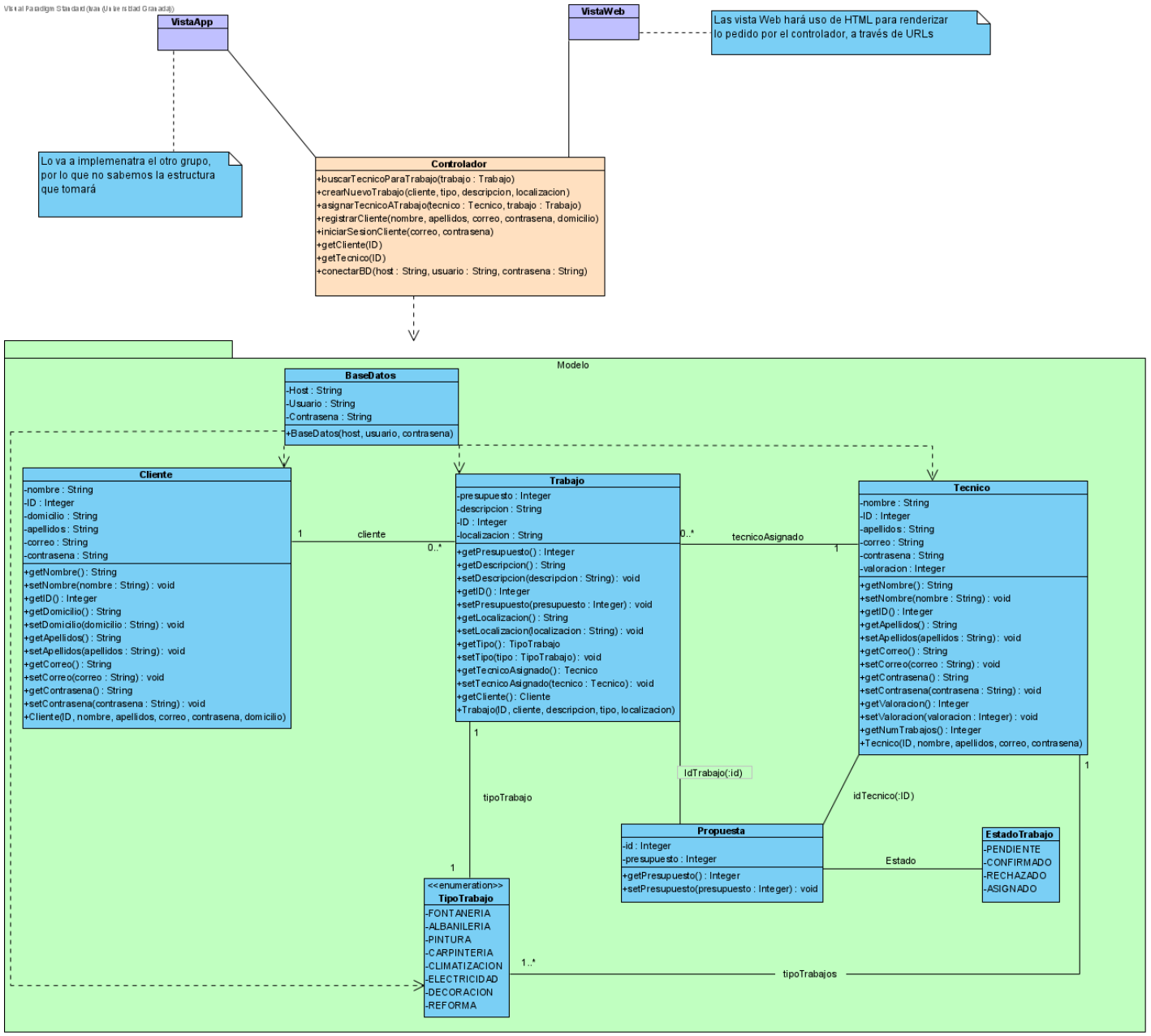
Las vistas nos permiten interactuar con el usuario del sistema, de manera que son un objeto meramente gráfico sustentado por el controlador, las vistas harán peticiones/invocaciones al controlador y este se encargará del resto (por ejemplo, un formulario consta del botón “registrarse”, al pulsarlo el controlador comienza el registro de usuario, para ello invoca a la base de datos y añade los nuevos datos del usuario e informa finalmente a la vista).

Si bien se usaría una BD para ello, algunas de esas clases se convertirán en tablas de la Base de Datos. Generalmente eso sería el modelo, quitando la clase BaseDatos para su conexión.

Expliquemos un poco esas clases. Los enumerados son al final tablas con claves únicas de forma que pudiéramos asegurar que los valores son los que se piden en cada momento, o en su defecto, una condición.

Por otra parte tenemos un controlador que se encarga de conectar las vistas con los modelos, si bien en Rails se puede ver que ya generan tales cosas por nosotros.

Las vistas por lo general solo renderizan HTML, por lo que aparte del renderizado y autocompletado por parte de los controladores, no ofrecen mucha más funcionalidad.

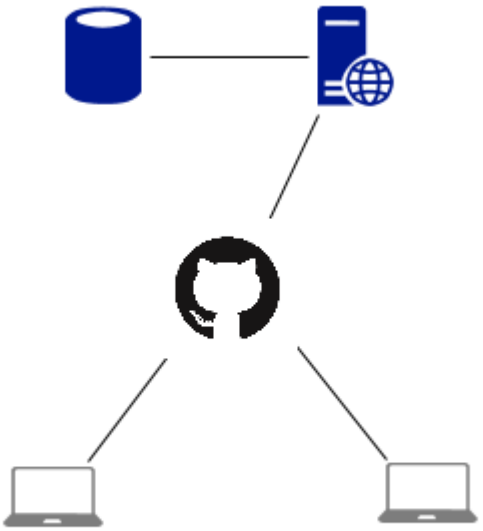

Visual Paradigm Standard (Ivan (Universidad Granada))
VistaApp
VistaWeb
Las vista Web hará uso de HTML para renderizar lo pedido por el controlador, a través de URLs
Lo va a implemenatra el otro grupo, por lo que no sabemos la estructura que tomará
Controlador
+buscarTecnicoParaTrabajo(trabajo : Trabajo)
+crearNuevoTrabajo(cliente, tipo, descripcion, localizacion)
+asignarTecnicoATrabajo(tecnico : Tecnico, trabajo : Trabajo)
+registrarCliente(nombre, apellidos, correo, contrasena, domicilio)
+iniciarSesionCliente(correo, contrasena)
+getCliente(ID)
+getTecnico(ID)
+conectarBD(host : String, usuario : String, contrasena : String)
Modelo
BaseDatos
-Host : String
-Usuario : String
-Contrasena : String
+BaseDatos(host, usuario, contrasena)
Cliente
-nombre : String
-ID : Integer
-domicilio : String
-apellidos : String
-correo : String
-contrasena : String
+getNombre() : String
+setNombre(nombre : String) : void
+getID() : Integer
+getDomicilio() : String
+setDomicilio(domicilio : String) : void
+getApellidos() : String
+setApellidos(apellidos : String) : void
+getCorreo() : String
+setCorreo(correo : String) : void
+getContrasena() : String
+setContrasena(contrasena : String) : void
+Cliente(ID, nombre, apellidos, correo, contrasena, domicilio)
1
cliente
0..*
Trabajo
-presupuesto : Integer
-descripcion : String
-ID : Integer
-localizacion : String
+getPresupuesto() : Integer
+getDescripcion() : String
+setDescripcion(descripcion : String) : void
+getID() : Integer
+setPresupuesto(presupuesto : Integer) : void
+getLocalizacion() : String
+setLocalizacion(localizacion : String) : void
+getTipo() : TipoTrabajo
+setTipo(tipo : TipoTrabajo) : void
+getTecnicoAsignado() : Tecnico
+setTecnicoAsignado(tecnico : Tecnico) : void
+getCliente() : Cliente
+Trabajo(ID, cliente, descripcion, tipo, localizacion)
0..*
tecnicoAsignado
1
Tecnico
-nombre : String
-ID : Integer
-apellidos : String
-correo : String
-contrasena : String
-valoracion : Integer
+getNombre() : String
+setNombre(nombre : String) : void
+getID() : Integer
+getApellidos() : String
+setApellidos(apellidos : String) : void
+getCorreo() : String
+setCorreo(correo : String) : void
+getContrasena() : String
+setContrasena(contrasena : String) : void
+getValoracion() : Integer
+setValoracion(valoracion : Integer) : void
+getNumTrabajos() : Integer
+Tecnico(ID, nombre, apellidos, correo, contrasena)
1
1
tipoTrabajo
IdTrabajo(:id)
idTecnico(:ID)
Propuesta
-id : Integer
-presupuesto : Integer
+getPresupuesto() : Integer
+setPresupuesto(presupuesto : Integer) : void
Estado
EstadoTrabajo
-PENDIENTE
-CONFIRMADO
-RECHAZADO
-ASIGNADO
1
<<enumeration>>
TipoTrabajo
-FONTANERIA
-ALBANILERIA
-PINTURA
-CARPINTERIA
-CLIMATIZACION
-ELECTRICIDAD
-DECORACION
-REFORMA
1..*
tipoTrabajos

El desarrollo de la implementación

Entorno de desarrollo

Como se ha mencionado anteriormente, en nuestro grupo hemos realizado la aplicación para clientes en Ruby on Rails. Para poder empezar a trabajar, necesitamos poder tener una copia de lo que se había creado en Clados, por lo que se hizo una copia del sistema original en un repositorio de GitHub privado.
Tras clonar el repositorio en los PCs hemos visto que la versión de Ruby que usaba este servidor era más actualizado del que tenían en los repositorios de paquetes de Ubuntu 20.04, por lo que tuvimos que usar la utilidad rbenv para instalar tal versión y posteriormente hacer un bundle install de todas las gemas que componían el proyecto en el servidor.

Así pues, teníamos una infraestructura que se resume en el siguiente diagrama:



De esa forma, como los dos tenemos el mismo entorno que el servidor, todo cambio que se haga en nuestro código se sube al repositorio para tratar posibles colisiones por trabajar en el mismo fichero, o para sincronizar los cambios entre ambos ordenadores para el desarrollo de la práctica. Como la copia local permitía el acceso a la base de datos, ahorrábamos tiempo de hacer pruebas en el servidor y evitábamos posibles fallos y actualizaciones constantes.

Una vez la funcionalidad estaba acorde a lo que se pensaba, se subía al servidor por si hubiera que alterar algún detalle, y ya dejarlo funcional.

Sobre la implementación

Para este sistema hemos creado el modelo y controlador de cada elemento que conforma el sistema:

- Clientes
- Técnicos
- Trabajos
- Propuestas

Del desarrollo realizado en Ruby on Rails podemos destacar la facilidad para crear nuevos elementos usando la orden generate scaffold, al principio nos costó entender la sintaxis y el modo de funcionamiento de este comando pero una vez comprendido se hizo más ameno. Otra facilidad reside en la conexión con la base de datos, desde RoR hemos podido tener cierta abstracción al respecto puesto que las estructuras de datos creadas ya enlazaban y ejecutaban consultas en la base de datos mediante métodos que nos ofrecen.

Si bien nuestro sistema parece un tanto sencillo, se le ha dado más hincapié en la API, ya que sirve de contacto entre lo que se hace en nuestra de clientes y la vista de Técnicos. por ello ha sido muy frecuente el intercambio de información entre ambos subgrupos para que se le pudiera facilitar lo que se necesitara, manteniendo lo que se daba a través de tal API, ya que de no hacerlo estaríamos vulnerando la seguridad. Era algo interesante que comentar ya que era algo que al momento de planificar no llegamos a pensar.

Entre las dificultades generales que hemos encontrado, podríamos destacar el tener que hacer uso de comandos get, put, delete, etc los cuales requieren de una ruta. Las rutas y cómo estas se configuran han supuesto una dificultad añadida al principio del desarrollo.

Además de la funcionalidad, se ha querido dotar de cierto estilo y responsividad para que, a pesar de una página web, sea agradable a varios tamaños de pantalla. Sin embargo, como no era nuestra prioridad, posiblemente haya pantallas donde no esté muy bien las posiciones.

Otro de los puntos más problemáticos es la transmisión de feedback, ya que hay veces que al querer enviar lo que ha fallado, no sale esas respuestas, así que no se da mucho feedback al usuario en caso de error.

Pruebas del sistema

Pruebas de Unidad

En las pruebas de unidad hemos comprobado el correcto funcionamiento del código de nuestra app, hemos aplicado la prueba de unidad a un método de todas las clases del proyecto, excepto la clase presupuesto que ha recibido dos pruebas de unidad.

Para la clase Tecnico hemos probado el constructor de este, comprobando que los valores introducidos se igualan correctamente a los atributos del mismo, el test del constructor ha sido realizado al igual que este en el resto de clases(presupuesto, trabajo y propuesta).

También hemos realizado una prueba a un método del main (`checkusuario`), el cual comprueba si existe el correo del usuario en la lista de los técnicos registrados en el sistema.

Por último la clase Presupuesto ha recibido un test probando el método total, el cual observamos si devuelve una cantidad correcta acorde a los atributos de esta clase.

Ejemplo de que los 6 tests pasan:

```
PS C:\Users\Ivan\Desktop\sosemadofinal> flutter test .\test\unidad_test.dart
Running "flutter pub get" in sosemadofinal...                       11,7s
00:12 +6: All tests passed!
```

Pruebas de Widgets

En los tests de widgets comprobamos el correcto funcionamiento de los algunos widgets de la app, hemos realizado 3 tests para nuestros diferentes widgets.

Para nuestro widget `PerfilTecnico`(), hemos realizado dos tests, el primero hemos comprobado que aparecen los textos correspondientes a los atributos del técnico identificado y el otro test hemos comprobado que aparezcan los 4 botones de funcionalidades del perfil (Modificar Tecnico, Eliminar Cuenta,Calculadora Presupuesto y Consultar Trabajos) que nos llevan a los otros dos widgets.

El último test ha sido para nuestro widget `RutaCalculadora`() donde hemos comprobado que todos los `TextFormField` se rellenan con los valores introducidos correctamente.

Foto de los 3 test pasados

```
PS C:\Users\Ivan\Desktop\sosemadofinal> flutter test .\test\widget_test.dart
00:03 +3: All tests passed!
```

Pruebas de Integración

Hemos realizado un ejemplo del funcionamiento de una parte de la aplicación.
Esta consiste en acceder a la calculadora de presupuestos.

El test consiste en iniciar sesión y obtener un presupuesto gracias a la calculadora de presupuestos, para ello:

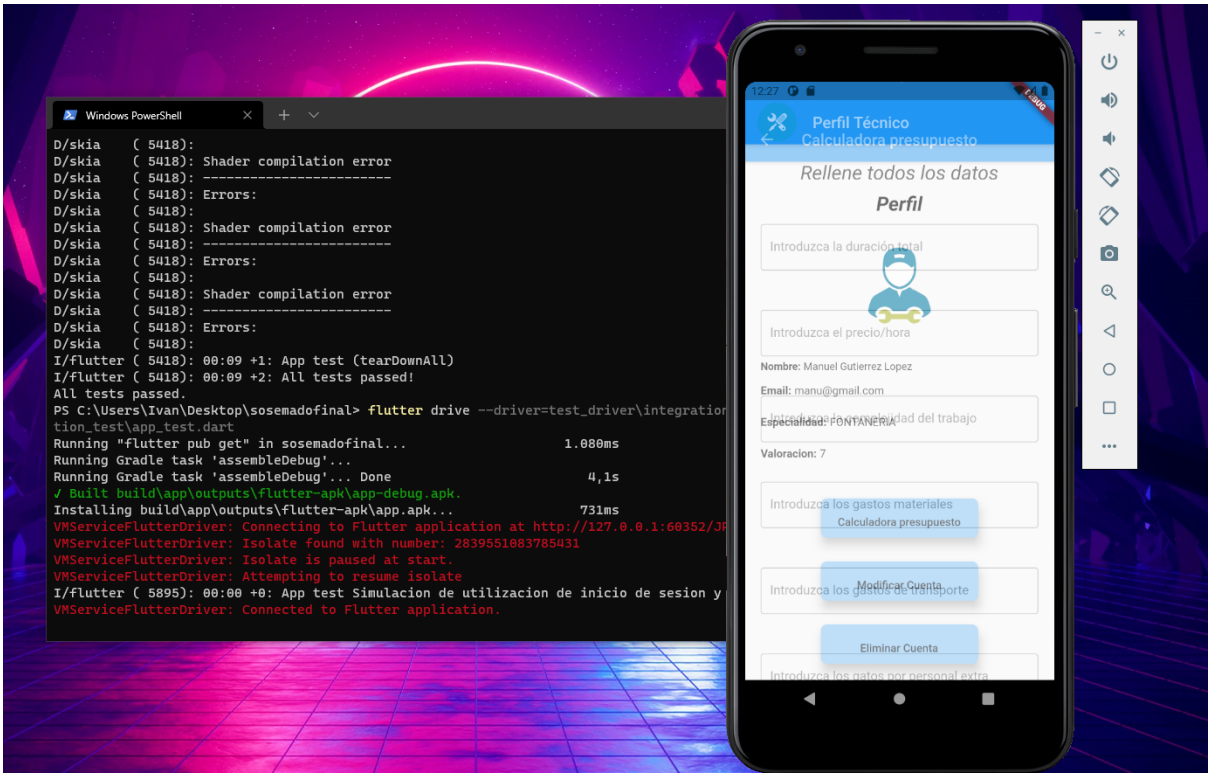
Primero: realizamos el login, comprobamos que existan los campos de inicio sesión y el botón, una vez comprobados, rellenamos con un usuario del sistema, y pulsamos el botón de iniciar sesión, pasamos a la página de `PerfilTecnico`.

Segundo: buscamos el botón de la calculadora y lo pulsamos, y nos lleva a la ruta Calculadora donde realizaremos la ayuda para un presupuesto introducido.

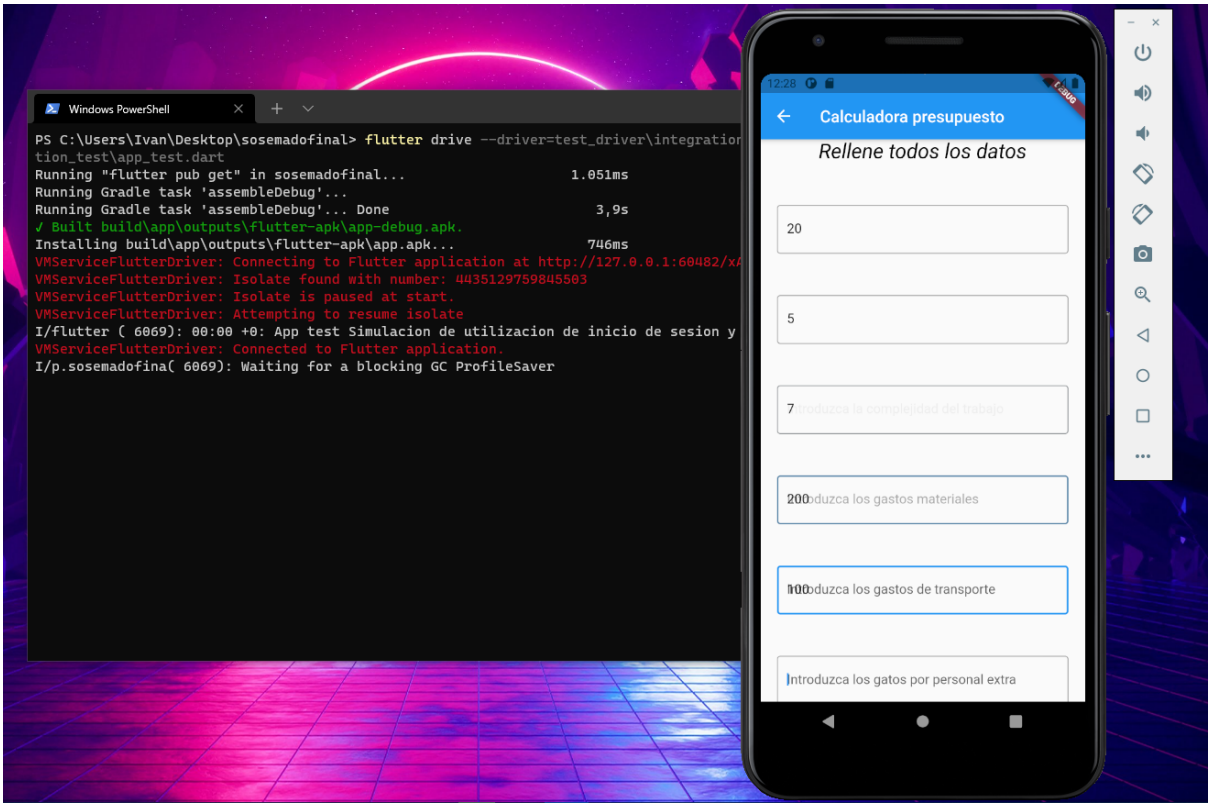
Tercero: una vez en la ruta Calculadora, comprobamos que están todos los campos y el botón de calcular el presupuesto. Una vez comprobados, rellenamos los campos con valores y pulsamos el botón.

Una de las cosas a destacar durante la prueba era que, según indicaciones de los compañeros, hay que esperar un segundo para que pueda conectarse a la BD, por lo que decidimos, desde que se ejecuta el test, dar 5 segundos de gracia para evitar problemas por ello.
Al final, los resultados de las pruebas son positivas.

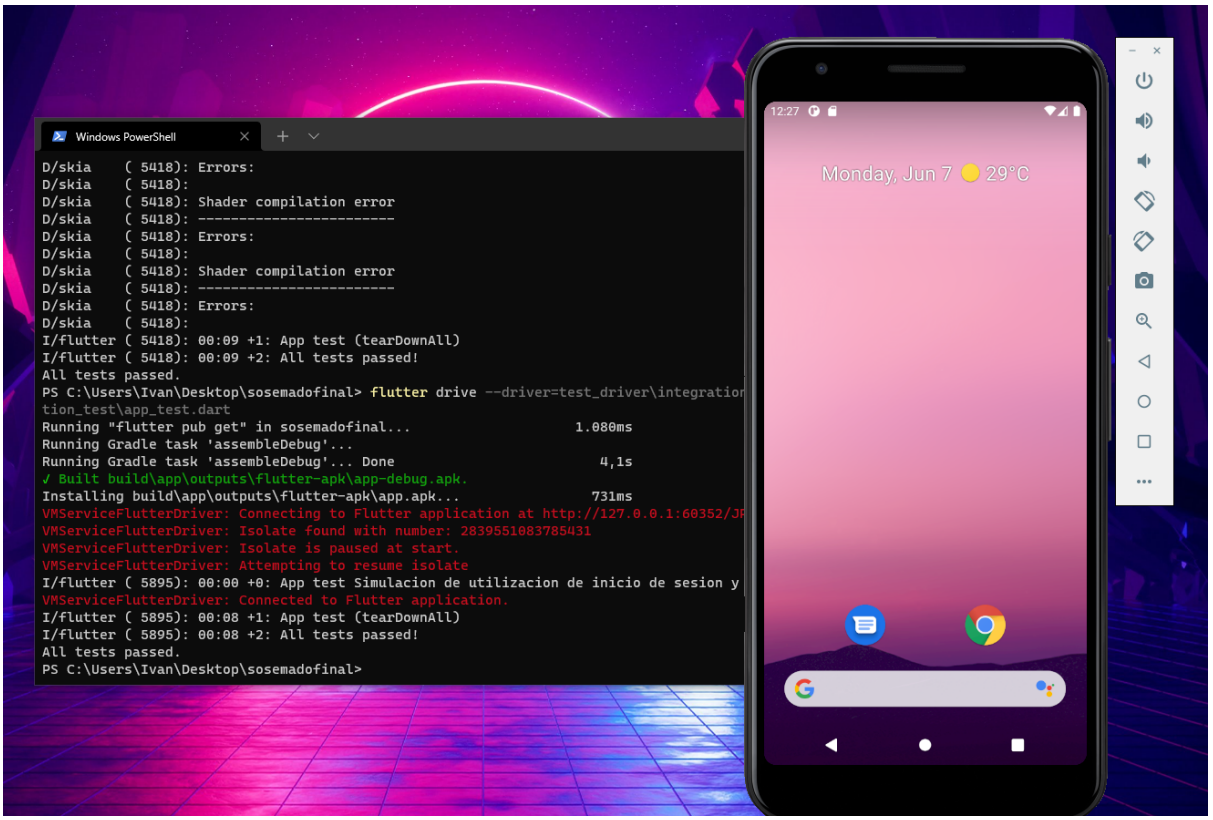
Adjuntamos capturas del proceso.



Captura en medio de la ejecución. Nótese que por la velocidad puede darse que la ejecución tarde del orden de segundos



Prueba de relleno de la calculadora



Captura final con la terminal diciendo que se completaron las pruebas.