UNIVERSIDAD
DE LA FRONTERA

RESERV-APP

Integrantes:
Javier Alcalde Vivas
Joaquín Faundez Concha
Christian Gajardo Contreras.

Asignatura:
Programación Orientada a Objetos

Profesor:
Samuel Eduardo Sepulveda Cuevas.

UNIVERSIDAD DE LA FRONTERA

ÍNDICE:

1. Introducción

En el contexto actual, el avance tecnológico crece cada vez más en todos los rubros, el turismo no es la excepción. Frente a gigantes plataformas para el arriendo de habitaciones, cabañas, etc. acaparan gran parte del mercado, pero ¿Qué hay de las pymes y pequeñas empresas del rubro?, claramente están en desventaja.

Es por esto que se crea la necesidad de un sistema que permita gestionar y llevar un registro de las reservas y clientes de emprendimientos más pequeños, y es aquí cuando nace Reserv-App.

En el siguiente documento se presentará la segunda versión de R.A., el cual deja de trabajar con programación estructurada, y se basa en la Programación Orientada a Objetos. También se abordarán los principales cambios realizados con respecto a la versión 1, definición de las clases identificadas, modelo UML y GUIS.

2. Contexto y problemática

La poca visualización que presentan los pequeños emprendimientos o “pymes” de zonas poco conocidas, es una problemática que afecta directamente a los dueños de negocios en el rubro del turismo, específicamente en el arriendo de hospedajes. Reserv-App (R.A.) es un software destinado a una pyme de cabañas (dos cabañas, siendo posible agregar más posteriormente) ubicadas en la región de la Araucanía, Chile. La problemática que se busca abordar con R.A. es dar accesibilidad a estas cabañas que no se encuentran en las grandes plataformas tales como AIRBNB o Booking, dándole al usuario la posibilidad de reservar estas cabañas a través de la aplicación, puesto a que en principio estas se reservan hablando directamente con los dueños de la pyme. De esta manera la reserva es más directa.

2.1 Objetivo

El objetivo de R.A. es poder simplificar la tarea de arrendar una cabaña, pero específicamente para los clientes de una pyme ubicada en la región de la Araucanía. Esto se logrará creando una app móvil basada en java, que permita la reserva directa de las cabañas por parte de los clientes, y la visualización de la disponibilidad de dicha cabaña.

2.2 Solución

La aplicación en su versión inicial contará con la interacción por consola con el usuario, esté al ingresar a la app deberá ingresar sus datos personales (nombre, numero de telefono, contraseña) para poder ingresar, hecho esto, los datos se almacenarán y el usuario podrá ver información sobre las cabañas y la disponibilidad de estas.

2.3 Cliente

R.A. es un software hecho a medida para un cualquier cliente que quiera arrendar una cabaña en la IX región, Araucanía, Chile. Esto hace que el producto sea único, y se adapte al contexto descrito anteriormente.

3. Diferencias Entre Versiones:

Se decidió que la reserva de las cabañas no será por 12 horas, por lo que el checkout se debe hacer manualmente, ya que esto sería de una dificultad mayor para nuestro proyecto, de manera que tenemos una aplicación enfocada a las 2 cabañas pertenecientes a la familia de nuestro compañero, la cual permite ver disponibilidad, arrendar las mismas y desocuparlas cuando sea necesario.. También ya se comienza a trabajar con el manejo de archivos de tipo JSON. A su vez las cabañas tendrán o no un arrendatario asociado.

UNIVERSIDAD DE LA FRONTERA

4. Descripción de Clases:

- ## Cabaña

  Atributos:

  - private int id
  - private String nombre
  - private int habitaciones
  - private int baños
  - private boolean isOcupada
  - private Cliente arrendatario

  Métodos:

  - public Cabaña(int id, String nombre, int habitaciones, int baños, boolean isOcupada, Cliente arrendatario)
  - public Cabaña(int id, String nombre, int habitaciones, int baños)
  - public int getId()
  - public Cliente getArrendatario()
  - public void setIsOcupada(boolean isOcupada)
  - public void reservarCabaña(Cliente usr)
  - public void setArrendatario(Cliente Arrendatario)
  - public void mostrarCabaña()
  - public void checkOutCabaña()
  - public JSONObject cabañaToJson()

- ## Cliente

  Atributos:

  - private String usuario
  - private String contraseña
  - private int celular

Métodos:

- public Cliente(String usuario, String contraseña, int celular)
- public String getUsuario()
- public String getContraseña()
- public JSONObject clienteToJson()

- Menú

Métodos:

- public void menuPrincipal()
- public void menuBienvenida()
- public void opcionesMenúBienvenida()
- public void opcionesMenúPrincipal()

- GestorDeCabañas

Atributos:

- private ArrayList<Cabaña> listaCabañas

Métodos:

- public GestorDeCabañas()
- public ArrayList<Cabaña> getListaCabañas()
- private Cabaña instanciarCabañaJson(JSONObject archivoCabaña)
- private ArrayList<Cabaña> setListaCabaña(ArrayList<JSONObject> cabañas)
- public void menuReservarCabaña(Cliente usr)
- public void menuCheckOutCabaña(Cliente usr)
- public void mostrarCabañasExistentes()
- public void mostrarCabañasReservadas(Cliente usr)
- private int lecturaInt()
- public void registrarCabañasEnArchivoJson()

- GestorDeClientes

  Atributos:

  - private ArrayList<Cliente> listaClientes

  Métodos:

  - public GestorDeClientes()
  - public ArrayList<Cliente> getListaClientes()
  - private Cliente instanciarClienteJson(JSONObject archivoCliente)
  - private ArrayList<Cliente> setListaClientes(ArrayList<JSONObject> clientes)
  - public void singUP()
  - public Cliente loginUsuario()
  - private boolean validarUsuario(String usuario, String contraseña)
  - public int obtenerPosicionUsuario(String Usuario)
  - private String lecturaString()
  - private int lecturaInt()
  - public void registrarClientesEnArchivoJson()

- Launcher

  Método: public static void main(String args[])

- GestorDeArchivos

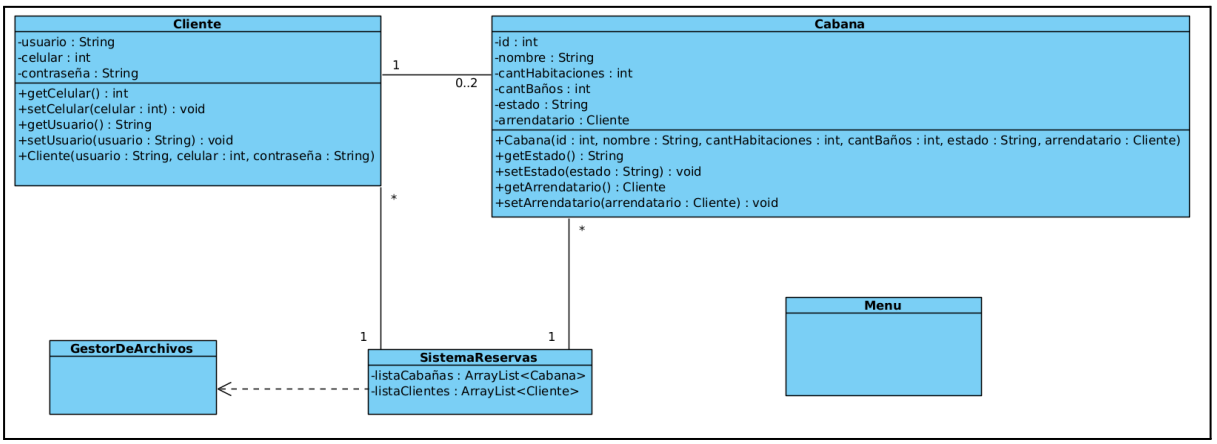
  Métodos:

- private void crearCarpeta
- private void eliminarArchivo
- private File[] listaArchivos
- private String getExtension
- private Boolean esArchivoJson

- public Boolean archivoJsonExiste
- public void escribirArchivoJSON
- private JSONObject leerArchivoJson
- private ArrayList<JSONObject> listaArchivosJson
- public Boolean usuarioExiste
- public ArrayList<JSONObject> listaJsonCliente
- public ArrayList<JSONObject> listaJsonCabañas

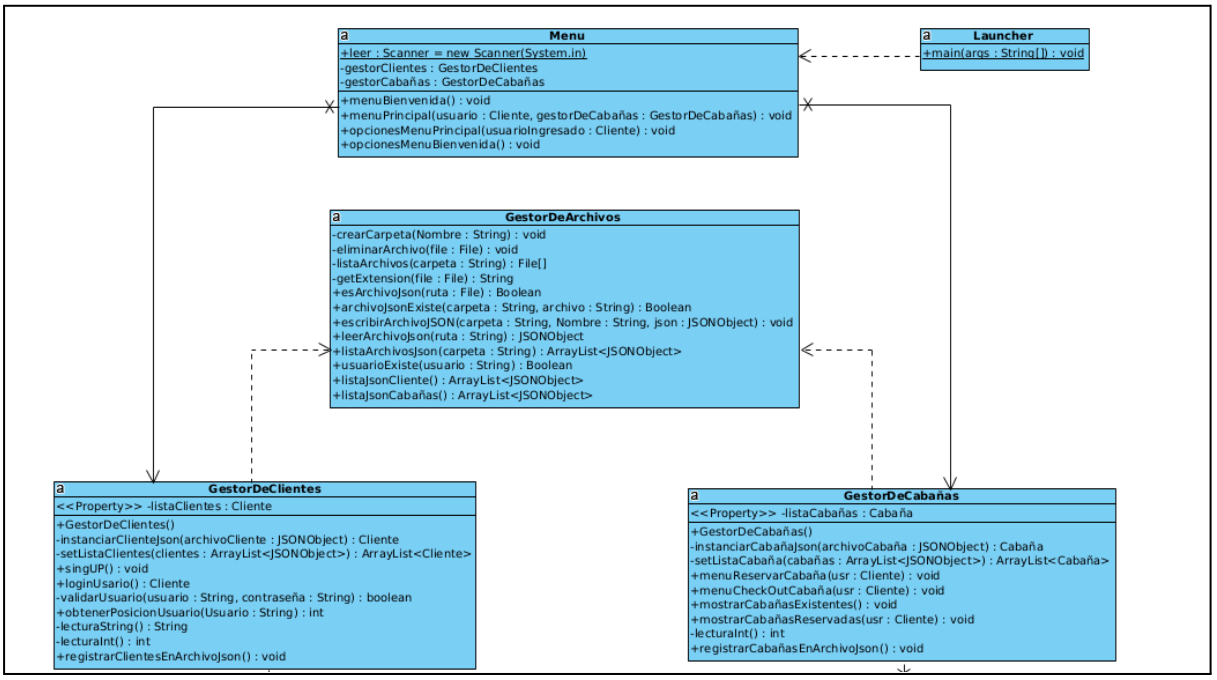
5. Modelo UML

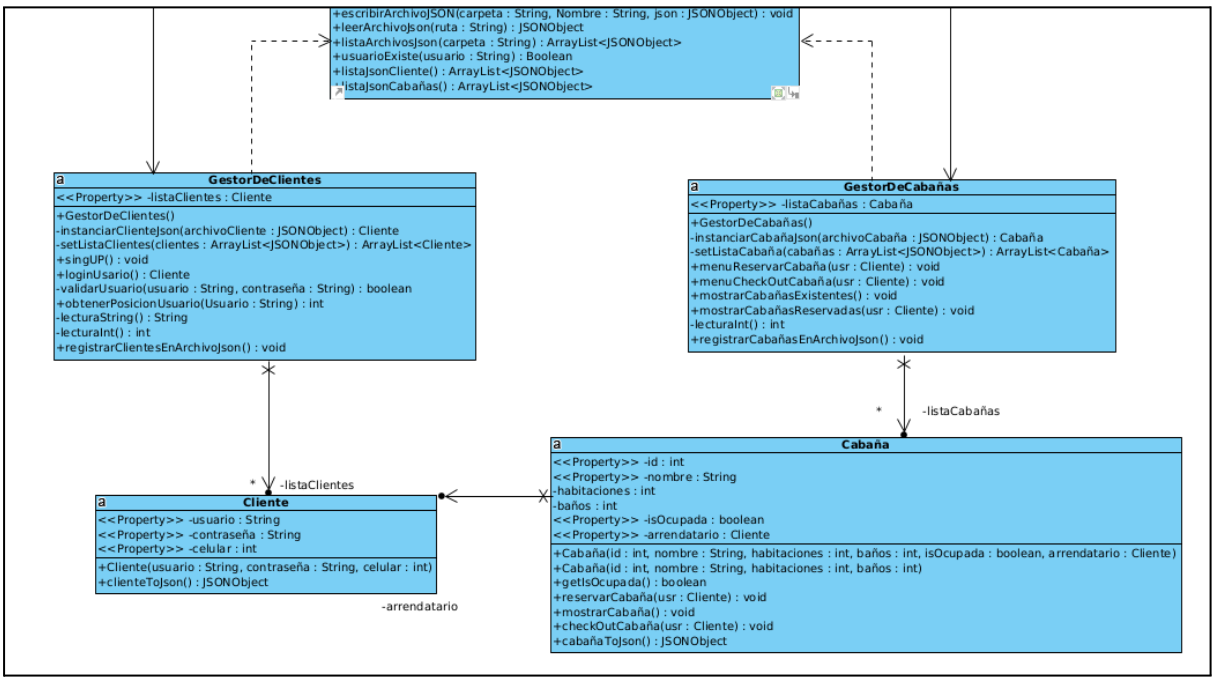
Primera iteración:



UNIVERSIDAD DE LA FRONTERA

Última interacción:





6. Manejo de Datos:

El tipo de archivo que fue utilizado para el almacenamiento de datos es el de .json, un formato que nos permite almacenar objetos json, tipo de datos ampliamente utilizado en la informática y bases de datos.

Los datos almacenados es la información de todos los atributos de los objetos previamente creados durante la ejecución del programa, pertenecientes a las clases cliente y cabaña.

En el caso de cliente se almacena: Nombre de usuario. (String)
Número celular. (int)
Contraseña vinculada. (String)

En el caso de cabaña se almacena:
ID. (Int)
Nombre de la cabaña. (String)
N° de habitaciones. (int)
N° de baños. (int)
El estado en el que se encuentra (Boolean)
Nombre del usuario asignado, en caso de existir. (String)

7. Interfaces

Menú bienvenida:

Esta es la ventana que se ejecuta al iniciar el programa, su objetivo es ser la primera impresión del usuario, mostrando la imagen principal de nuestra app y de igual manera las dos primeras opciones: iniciar sesión y registrar usuario nuevo.

LogIn:

Ventana que nos permite acceder con nuestro usuario previamente creado, de manera de acceder a una versión personalizada de las múltiples funciones de nuestra app.

SingUp:

Ventana que nos permite crear nuevos usuarios, permitiendo ingresar los datos relevantes del usuario (nombre usuario, celular y contraseña). Siendo el caso que solamente si el usuario no existía antes, cree un nuevo objeto cliente con su respectivo .json

Cabañas disponibles:

Ventana que muestra todas las cabañas existentes al usuario ya ingresado.

Reserva de cabaña:

Ventana que permite realizar una reserva de la cabaña seleccionada, modificando sus atributos a "isOcupada: true" y asignándole como atributo el objeto de la clase cliente respectivo, solo en el caso que este disponible.

Mis arriendos:

Ventana que permite ver la características de mis cabañas anteriormente arrendadas y de igual manera dar la opción de realizar un checkOut.

8. Conclusión

En base a la versión anterior, se logró cambiar de la programación estructurada a la POO, haciendo uso de Visual Paradigm para lograr modelar el diagrama de clases, el cual facilitó la tarea inicial. También se logró implementar un sistema de almacenamiento en formato .json, permitiendo respaldar la información de cualquier cambio realizado dentro del sistema, ahora bien todavía quedan desafíos.

El desarrollo de una interfaz gráfica funcional (GUI) es el mayor reto que se deberá trabajar e investigar, para lograr su efectiva implementación y que logre responder a todas nuestras expectativas como desarrolladores, seguido con eso es asegurar que nuestro programa no se caiga bajo ninguna circunstancia en situaciones límites, y logrando una comunicación amigable con el usuario que le permita entender qué ocurre.

UNIVERSIDAD DE LA FRONTERA

9. Links

Github:
https://github.com/jfaundez07/Proyecto_POO.git

10. Bibliografía

JSON. (2006-2023). JSON is a light-weight, language independent, data interchange format. Maven Central. Recuperado 09/09/2023:
https://mvnrepository.com/artifact/org.json/json
JUnit 5 User Guide. (2023). Recuperado 09/09/2023
https://junit.org/junit5/docs/current/user-guide/
Oracle. (2023). Lesson: Exceptions (The Java™ Tutorials > Essential Java Classes). Recuperado 09/09/2023
https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html