



**Universidad
Tecnológica
del Perú**

**“IMPLEMETACION DE UN APLICATIVO PARA LA GESTION DE LA CLINICA DENTAL
PRINCE DENT”**



Curso:

Algoritmos y Estructura de Datos

Profesor:

ALVARADO HERRADA, EDGAR FRANCISCO

INTEGRANTES

BENITES QUISPE, ANDERSON FRANCK

CASAS CASTAÑEDA, JUAN DIEGO

RAFAEL JULCAMORO, JAKELI GIANNINA

SANTOS BORJA, ALDO MOISÉS

LIMA – 2023

ÍNDICE

Introducción

CAPÍTULO 1

Aspectos generales

Organigrama

Misión

Visión

Objetivos estratégicos

Problemática

Alternativas de solución

Solución elegida

CAPÍTULO 2

Estado del arte

Marco teórico

CAPÍTULO 3

Alcance

Requerimientos funcionales y no funcionales

Restricciones

Diagrama de clases

Prototipo o interfaces

CAPÍTULO 4

Diseño de la aplicación

Funcionalidades del aplicativo implementadas

Estructura de paquetes

Clases usadas en la aplicación

Código fuente

Bibliografía

Introducción:

Prince Dent es una clínica que ofrece diversos servicios relacionados con la salud bucodental. Actualmente, la empresa registra las citas de sus pacientes con una página web suscrita.

Sin embargo, dicha suscripción tiene un costo elevado y no cuenta con algunas funciones que podrían ayudar al usuario, lo que genera inconvenientes. Por ejemplo, en algunas ocasiones los pacientes sufren algún inconveniente y desean reagendar la cita, por lo que el usuario debe registrar otra cita en vez de postergar la que ya existía. Otro ejemplo sería que los especialistas solo están disponibles ciertos días en ciertas horas, lo que genera algunos problemas al momento de registrar las citas.

Es por ello, que la clínica desea implementar un aplicativo propio de gestión de citas, siendo este el principal objetivo de este trabajo.

CAPÍTULO 1

1.1 Aspectos generales:

En este proyecto, buscamos llevar a cabo una profunda transformación en la clínica dental "Prince Dent", mediante la incorporación de un nuevo software, un enfoque innovador que revolucionará la manera en que gestionamos nuestra clínica. El software que proponemos se trata de más que solo un software; representa una visión de modernización y eficiencia en cada aspecto de nuestra operación.

El núcleo del Software se basa en la agilización de procesos clave, como la realización de citas, la elección de doctores especializados y la implementación de un sistema de inventario integral. A través de la integración de tecnología avanzada, estamos comprometidos en brindar un servicio dental de alta calidad de manera más efectiva y cómoda para nuestros pacientes.

Este proyecto no solo es un impulso hacia la automatización, sino también un esfuerzo para aumentar la calidad y la atención personalizada que ofrecemos. Buscamos fortalecer nuestra relación con los pacientes y brindar un ambiente más acogedor en " Prince Dent ", al tiempo que optimizamos nuestros procesos internos.

1.2 Organigrama:

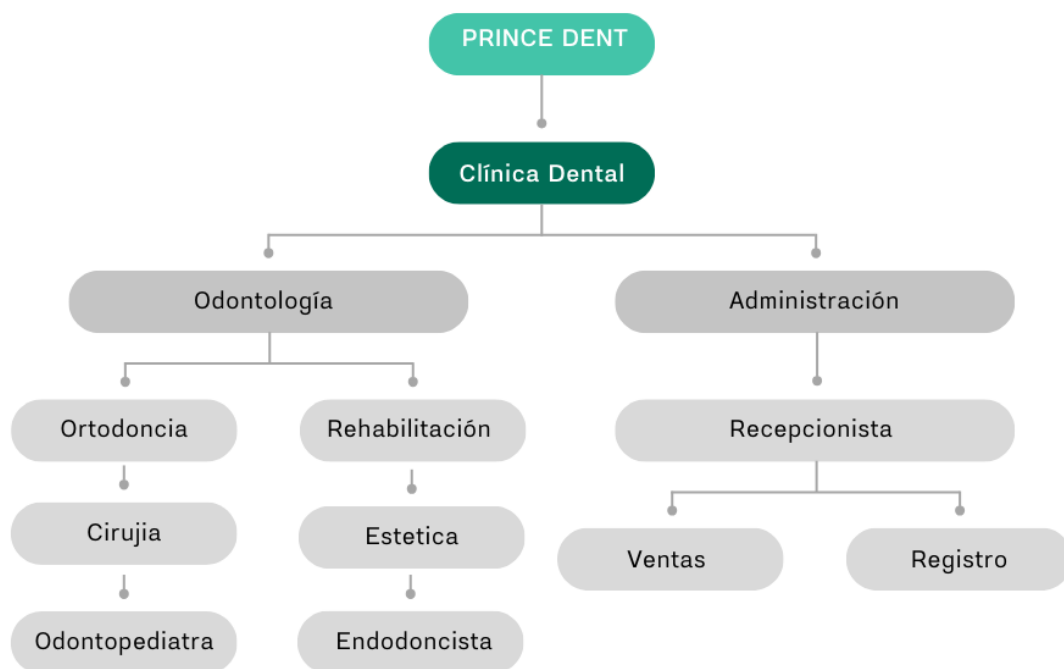


Figura 1. Organigrama de la empresa Prince Dent

Este organigrama refleja una clínica dental que separa claramente las responsabilidades clínicas como sería en servicio de odontología y el área de administración. La comunicación y coordinación efectivas entre ambas partes son fundamentales para proporcionar un servicio de alta calidad a los pacientes y garantizar el funcionamiento eficiente de la clínica. Por eso, se considera que podría ayudar a mejorar a un funcionamiento, mejorando la comunicación de estas áreas importantes.

1.3 Misión:

Somos una empresa odontológica que ofrece servicios por especialidades. Ofrecemos una eficiente atención integral de calidad para lograr el bienestar de nuestros pacientes.

1.4 Visión:

Ser una empresa líder en odontología en afán de superación y comprometida con el cambio. Satisfacer las expectativas de nuestros pacientes a través de la eficiencia y ética, con equidad, solidez y calidad de vida.

1.5 Objetivos estratégicos:

Objetivo general: Desarrollar un aplicativo en Java que sirva para la gestión de la clínica dental “Prince Dent”.

1.6 Objetivos específicos:

- El aplicativo contará con un inicio de sesión donde el usuario insertará su usuario y su contraseña.
- El usuario podrá registrar a los pacientes insertando sus datos personales, así como actualizar la información o eliminar pacientes del registro.
- El usuario podrá registrar una cita a los pacientes que hayan sido registrados en el sistema.
- El usuario podrá buscar el historial de citas de cualquier paciente mediante su clave primaria.
- El usuario podrá registrar a los doctores insertando datos importantes para la empresa, además de poder actualizar la información y eliminar doctores del registro.
- El usuario podrá añadir nuevas especialidades si la empresa decide implementarlas en su local.

1.7 Problemática:

Unas de las problemáticas que podemos ver dentro de la clínica dental es el manejo de la gestión de historial clínico, gestión de citas y el de venta de productos. Todos estos aspectos se tratan a través de la recepcionista creando demoras en los procesos, que se podría optimizar con el software indicado. Para esto se pudo ver que, para registrar el historial clínico de cada paciente, el medico a cargo se tiene que acercar a la recepcionista para indicar los estudios que realizo al paciente. A la hora de realizar la venta de los productos, se visualizó que no manejan ningún tipo de control de inventario y por último la gestión de citas se maneja a través de la recepcionista y las redes sociales, algo que debería ser más automatizado para la toma de datos.

1.8 Alternativas de solución:

- Una de las alternativas, para esta problemática es la de alquilar un software de terceros, que algunas empresas ofrecen, el inconveniente de estos es que no está modelado de acuerdo con las necesidades de la clínica dental, y no deja que los procesos dentro de la clínica se desarrollen de manera correcta.
- Por otra parte, otra alternativa de solución es la desarrollar un aplicativo en Java que se acomode a las necesidades de la clínica dental, con esto poder ayudarle a automatizar procesos que necesite, asimismo poder llevar un control mejor sobre los requerimientos de "Prince Dent".

1.9 Solución elegida:

- La solución que se eligió es la de desarrollo del aplicativo, ya que es la que mejores ventajas ofrecen para la empresa. Con esta implementación se podrá tener un mayor control sobre la gestión de citas médicas y control de historias clínicas, adecuándose a lo que la empresa solicite y necesite dentro de los procesos que desarrolla.

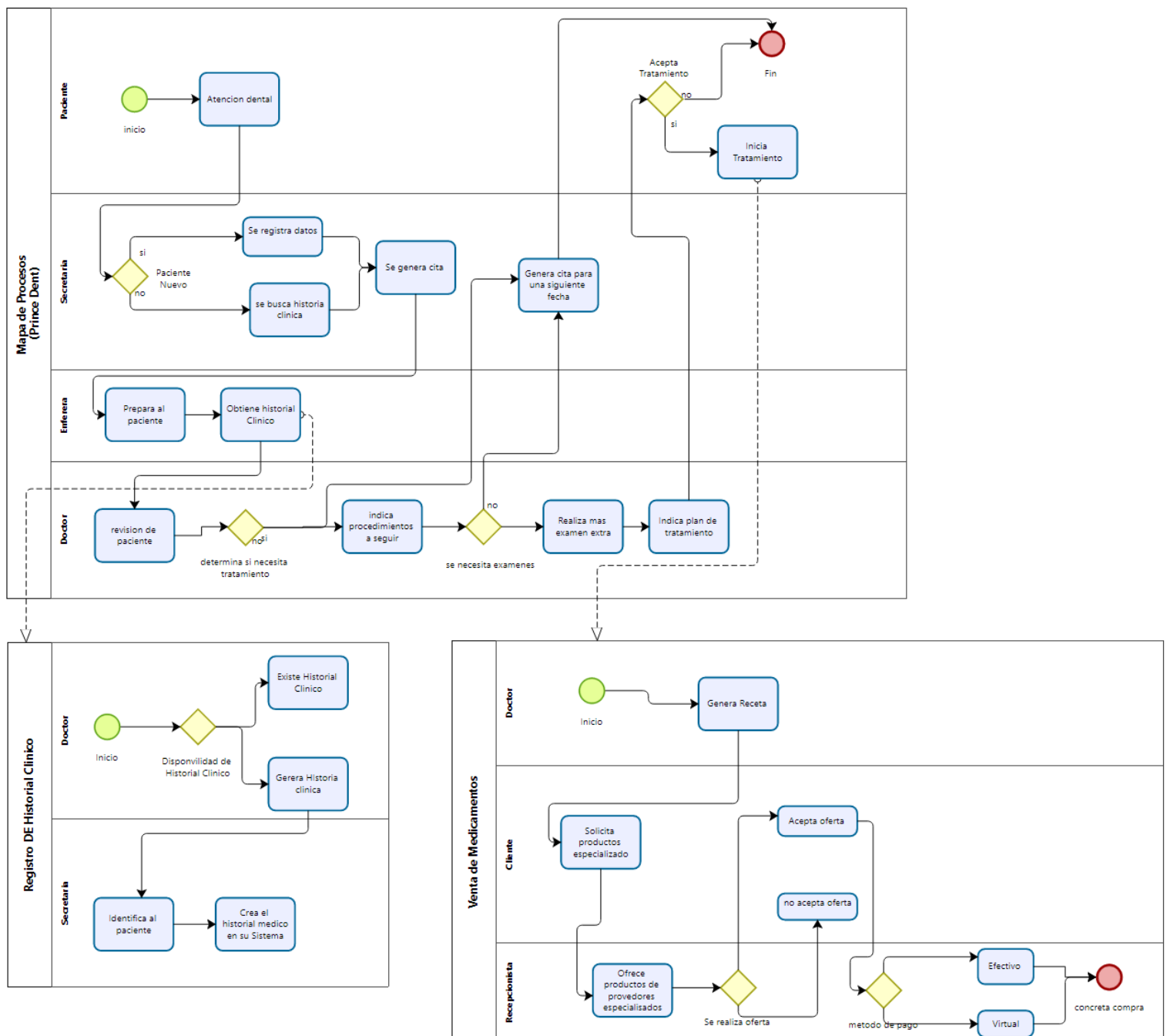
CAPÍTULO 2

2.1 Estado del arte:

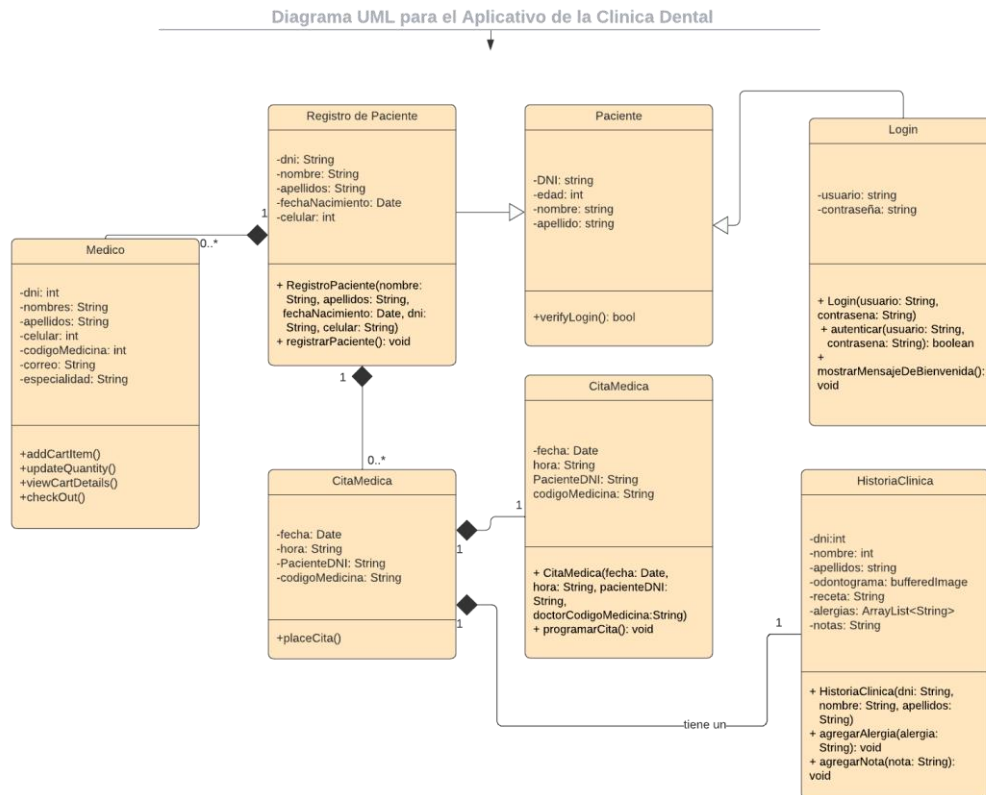
- En Ecuador, para ser exactos en la ciudad de Latacunga, se realizó un trabajo de investigación de un software para una clínica llamada " Dental Dent House" en el año 2022, el trabajo consistió de un sistema informático para la administración de la clínica para saber cómo son las historias clínicas odontológicas y el manejo de un odontograma para el seguimiento de cada paciente, por otro parte en Uruguay un grupo de universitarios de la Universidad " Laica Alfaro" de la ciudad de Manabi, realizaron una investigación que tuvo como objetivo implementar un Sistema SaaS para la optimización de Procesos del centro odontológico "Stetic Dental" tuvieron que realizar procedimientos cuantitativos y cualitativas para desarrollar el estudio.
- Y por último en Chile en la Universidad "Bio Bio" tuvieron que realizar un proyecto para su carrera de ing. De sistemas, su proyecto se llamó Sistemas de Fichas y gestión de abonos para la clínica Dental "Dentoart", la problemática de esta clínica era la aglomeración de fichas dentales y el orden de abonos realizados por los pacientes por lo cual tuvieron que utilizar la metodología iterativa incrementar junto con el

Framework Yii2, el cual se basa en el patrón de la arquitectura de software, entre otros lenguajes de programas y funciones de sistemas.

2.2 Diagrama de Procesos:



2.3 Diagrama UML:



2.3 Base Datos

En el esquema presentado anterior se muestra 5 entidades en la base de datos que se usara para el software de “Prince Dent”. La entidad relación es cita, de ahí se relaciona con las otras entidades Login, Paciente y doctor, vemos también que Especialidad está relacionado con Doctor. Esta base de datos nos ayuda a entender los datos necesarios para el manejo de la clínica.

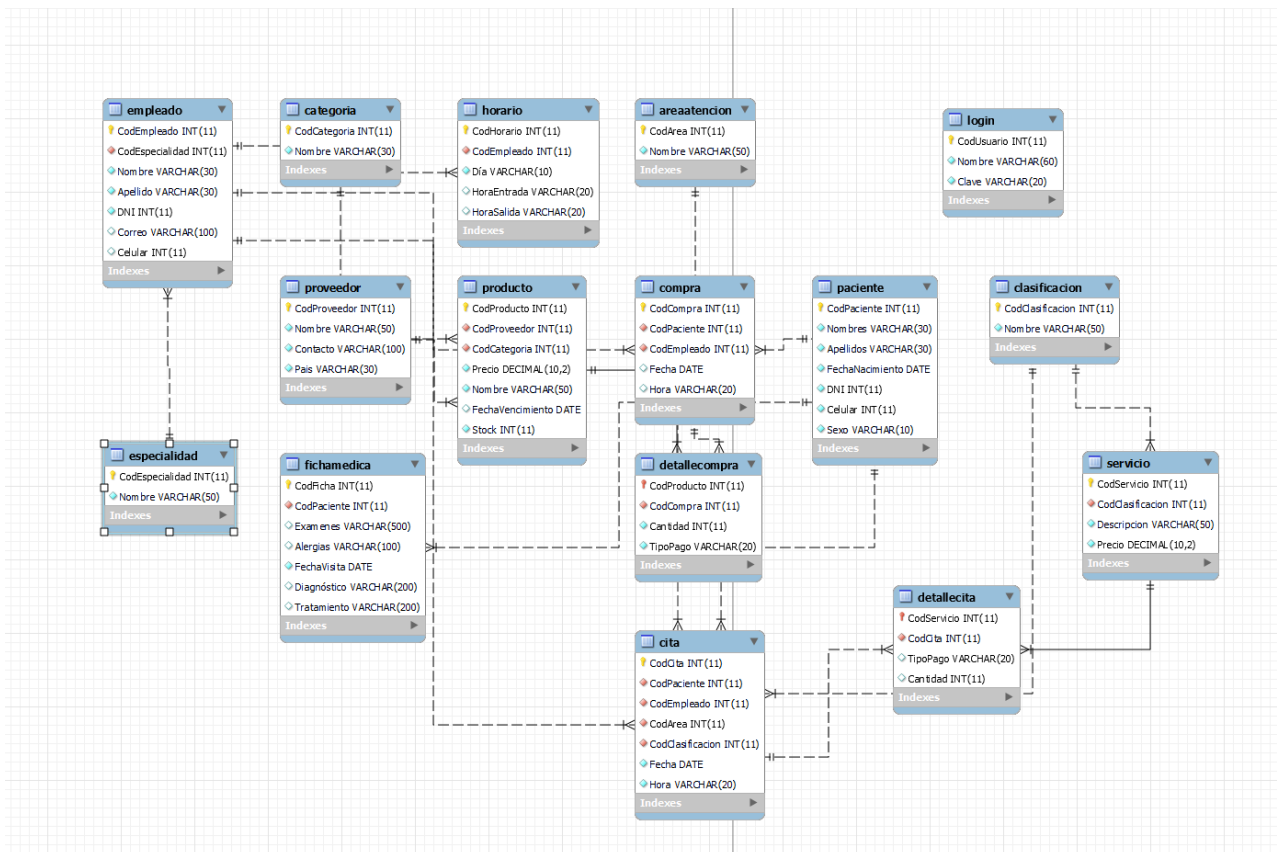


Figura 2. Base de Datos para el software Prince Dent

2.4 Referencias Funcionales:

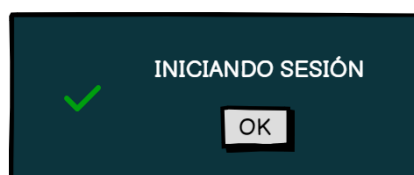
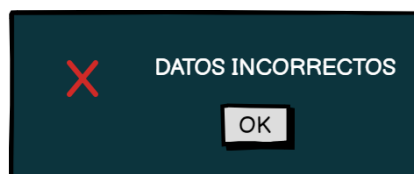
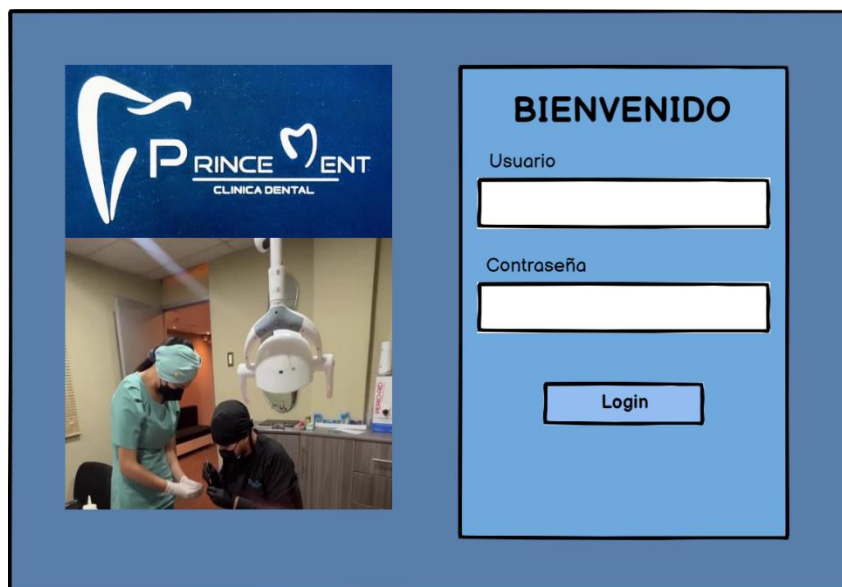
- ❖ La aplicación contará con la función de Login, para que los usuarios autorizados puedan utilizar el aplicativo
- ❖ La aplicación brinda la función de búsqueda de paciente con el cual se pueda hallar todos sus datos registrados
- ❖ La aplicación brinda la creación de usuarios para el uso del aplicativo
- ❖ La aplicación puede registrar el historial clínico de cada paciente que se registre en Prince Dent
- ❖ El aplicativo brinda la opción de la gestión de citas para los pacientes
- ❖ El aplicativo proporciona a cada paciente un apartado para sus recetas medicas
- ❖ El aplicativo permite visualizar el historial de citas de cada paciente
- ❖ El aplicativo proporciona un área para el pago
- ❖ El aplicativo maneja un registro de empleados
- ❖ El aplicativo maneja un registro de pagos por cada paciente

- ❖ El aplicativo ofrece la opción de tratamientos en el cual se agrega para cada paciente

2.5 Referencias no Funcionales:


- ❖ El aplicativo esta desarrollado en el IDE NetBeans el cual es de código libre
- ❖ El aplicativo contara una base de datos desarrollada en MySQL
- ❖ El aplicativo se está realizando de manera cooperativa a través de la herramienta Git
- ❖ El aplicativo tiene bocetos desarrollados en Balsamiq
- ❖ El aplicativo se gestiona el trabajo a través de la herramienta Trello
- ❖ El aplicativo

2.6 Prototipo:



PacienteHistorialCitaEspecialidadesDoctor

Nuevo PacienteGestionar Paciente



Nuevo Paciente

Nombre

Apellido

Fecha de Nacimiento


DNI

Celular

Guardar

PacienteHistorialCitaEspecialidadesDoctor

Nuevo PacienteGestionar Paciente



Gestionar Paciente

Q search

NombreApellidoFecha N.DNICelular

Actualizar

Eliminar


Nombre

DNI

Apellido

Celular

Fecha de Nacimiento



Paciente

Historial

Cita

Especialidades

Doctor

Historial - Paciente



Historial de Paciente

Nombre

Apellido

Fecha N.

DNI

Celular



Paciente

Historial

Cita

Especialidades

Doctor

Cita - Paciente



Cita de Paciente

Nombre

Apellido

Fecha de Nacimiento

DNI

Celular

Tipo Consulta

▼

Doctores

▼

Fecha



Hora

▼

SEPTEMBER 2023

S

M

T

W

T

F

S

27

28

29

30

31

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

1

2

3

4

5

6

7



Registrar

Paciente


Historial

Cita

Especialidades

Doctor

Nueva Especialidad



Tipo de Especilidad

Descripción de la consulta

Guardar

Paciente

Historial

Cita

Especialidades

Doctor

Registro Doctor

Gestión Doctor



Registro de Doctores

Nombre	<input type="text"/>
Apellido	<input type="text"/>
DNI	<input type="text"/>
Correo	<input type="text"/>
Celular	<input type="text"/>
Especialidad	<input type="text"/>

Guardar

Paciente

Historial


Cita

Especialidades

Doctor

Registro Doctor

Gestión Doctor



Gestionar datos de Doctores

Q search

Nombre

Apellido

DNI

Correo

Celular

Especialidad

Actualizar

Eliminar

Nombre


Apellido

Correo

DNI

Celular

Especialidad



3 Desarrollo:

3.1 Package Modelo:

Empleado:

```
public class Empleado {
    //Atributos
    private int codEmpleado;
    private int codEspecialidad;
    private String nombre;
    private String apellido;
    private int dni;
    private String correo;
    private int celular;

    public Empleado() {
        this.codEmpleado = 0;
        this.codEspecialidad = 0;
        this.nombre = "";
        this.apellido = "";
        this.dni = 0;
        this.correo = "";
        this.celular = 0;
    }

    public Empleado(int codEmpleado, int codEspecialidad, String nombre, String apellido, int dni,
        this.codEmpleado = codEmpleado;
        this.codEspecialidad = codEspecialidad;
        this.nombre = nombre;
        this.apellido = apellido;
        this.dni = dni;
        this.correo = correo;
        this.celular = celular;
    }

    public int getCodEmpleado() {
        return codEmpleado;
    }

    public void setCodEmpleado(int codEmpleado) {
        this.codEmpleado = codEmpleado;
    }

    public int getCodEspecialidad() {
        return codEspecialidad;
    }

    public void setCodEspecialidad(int codEspecialidad) {
        this.codEspecialidad = codEspecialidad;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getApellido() {
        return apellido;
    }

    public void setApellido(String apellido) {
        this.apellido = apellido;
    }

    public int getDni() {
        return dni;
    }
}
```

Este código Java define una clase llamada "Empleado" que representa a un empleado en un sistema. La clase tiene los siguientes atributos:

- codEmpleado: Un entero que representa el código del empleado.
- codEspecialidad: Un entero que representa el código de la especialidad del empleado.

- nombre: Una cadena de texto que almacena el nombre del empleado.
- apellido: Una cadena de texto que almacena el apellido del empleado.
- DNI: Un entero que almacena el número de identificación del empleado.
- correo: Una cadena de texto que almacena la dirección de correo del empleado.
- celular: Un entero que almacena el número de teléfono celular del empleado.

La clase tiene dos constructores, uno sin argumentos que inicializa todos los atributos en sus valores predeterminados y otro que acepta argumentos para inicializar los atributos con valores específicos. Además, proporciona métodos "get" y "set" para acceder y modificar los valores de estos atributos. En resumen, esta clase modela la información de un empleado en un sistema

Especialidad:

```
public class Especialidad {
    private int codEspecialidad;
    private String nombre;

    public Especialidad () {
        this.codEspecialidad = 0;
        this.nombre = "";
    }

    public Especialidad(int codEspecialidad, String nombre) {
        this.codEspecialidad = codEspecialidad;
        this.nombre = nombre;
    }

    public int getCodEspecialidad() {
        return codEspecialidad;
    }

    public void setCodEspecialidad(int codEspecialidad) {
        this.codEspecialidad = codEspecialidad;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}
```

Este código Java define una clase llamada "Especialidad" que representa una especialidad en el contexto de un sistema. La clase tiene los siguientes atributos:

- codEspecialidad: Un entero que representa el código de la especialidad.
- nombre: Una cadena de texto que almacena el nombre de la especialidad.

La clase tiene dos constructores, uno sin argumentos que inicializa ambos atributos en sus valores predeterminados y otro que acepta argumentos para inicializar los atributos con valores específicos. Además, proporciona métodos "get" y "set" para acceder y modificar los valores

de estos atributos. En resumen, esta clase modela la información de una especialidad en un sistema.

Ficha Medica:

```
public class FichaMedica {  
  
    //Atributos  
    private int codFicha;  
    private int codPaciente;  
    private String examenes;  
    private String alergias;  
    private String ficha_visita;  
    private String diagnostico;  
    private String tratamiento;  
  
    public FichaMedica() {  
        this.codFicha = 0;  
        this.codPaciente = 0;  
        this.examenes = "";  
        this.alergias = "";  
        this.ficha_visita = "";  
        this.diagnostico = "";  
        this.tratamiento = "";  
    }  
  
    public FichaMedica(int codFicha, int codPaciente, String examenes, String alergias, String ficha_visita, String diagnostico, String tratamiento) {  
        this.codFicha = codFicha;  
        this.codPaciente = codPaciente;  
        this.examenes = examenes;  
        this.alergias = alergias;  
        this.ficha_visita = ficha_visita;  
        this.diagnostico = diagnostico;  
        this.tratamiento = tratamiento;  
    }  
}
```

Este código Java define una clase llamada "FichaMedica" que representa la información de una ficha médica en el contexto de un sistema de gestión de registros de pacientes. La clase tiene los siguientes atributos:

- **codFicha:** Un entero que representa el código de la ficha médica.
- **codPaciente:** Un entero que representa el código del paciente al que pertenece la ficha médica.
- **examenes:** Una cadena de texto que almacena información sobre los exámenes médicos realizados.
- **alergias:** Una cadena de texto que almacena información sobre las alergias del paciente.
- **ficha_visita:** Una cadena de texto que almacena detalles de la visita médica.
- **diagnostico:** Una cadena de texto que almacena el diagnóstico médico.

- **tratamiento:** Una cadena de texto que almacena información sobre el tratamiento médico.

La clase tiene dos constructores, uno sin argumentos que inicializa todos los atributos en sus valores predeterminados y otro que acepta argumentos para inicializar los atributos con valores específicos. Además, proporciona métodos "get" y "set" para acceder y modificar los valores de estos atributos. En resumen, esta clase modela la información de una ficha médica en un sistema de gestión de pacientes.

Paciente:

```
public class Paciente {
    //Atributos
    private int codPaciente;
    private String nombre;
    private String apellido;
    private String fecha_nacimiento;
    private int dni;
    private int celular;
    private String sexo;

    public Paciente() {
        this.codPaciente = 0;
        this.nombre = "";
        this.apellido = "";
        this.fecha_nacimiento = "";
        this.dni = 0;
        this.celular = 0;
        this.sexo = "";
    }

    public Paciente(int codPaciente, String nombre, String apellido, String fecha_nacimiento, int dni, int celular, String sexo) {
        this.codPaciente = codPaciente;
        this.nombre = nombre;
        this.apellido = apellido;
        this.fecha_nacimiento = fecha_nacimiento;
        this.dni = dni;
        this.celular = celular;
        this.sexo = sexo;
    }

    public int getCodPaciente() {
```

Este código Java define una clase llamada "Paciente" que representa la información de un paciente en el contexto de un sistema de gestión de registros médicos. La clase tiene los siguientes atributos:

- **codPaciente:** Un entero que representa el código del paciente.
- **nombre:** Una cadena de texto que almacena el nombre del paciente.
- **apellido:** Una cadena de texto que almacena el apellido del paciente.

- **fecha_nacimiento:** Una cadena de texto que almacena la fecha de nacimiento del paciente.
- **dni:** Un entero que almacena el número de identificación del paciente.
- **celular:** Un entero que almacena el número de teléfono celular del paciente.
- **sexo:** Una cadena de texto que almacena el género o sexo del paciente.

La clase tiene dos constructores, uno sin argumentos que inicializa todos los atributos en sus valores predeterminados y otro que acepta argumentos para inicializar los atributos con valores específicos. Además, proporciona métodos "get" y "set" para acceder y modificar los valores de estos atributos. En resumen, esta clase modela la información de un paciente en un sistema de gestión de registros médicos.

3.2 Package Controlador:

C. Empleado:

```
public class Controlador_Empleado {

    /*Metodo para guardar Empleado*/
    public boolean guardar(Empleado objeto) {
        boolean respuesta = false;
        Connection cn = conexion.conectar();
        try {
            PreparedStatement consulta = cn.prepareStatement("insert into Empleado values(?, ?, ?, ?, ?, ?)");
            consulta.setInt(1, 0); //codEmpleado
            consulta.setInt(2, 0); //codEspecialidad
            consulta.setString(3, objeto.getNombre());
            consulta.setString(4, objeto.getApellido());
            consulta.setInt(5, objeto.getDni());
            consulta.setString(6, objeto.getCorreo());
            consulta.setInt(7, objeto.getCelular());
            if (consulta.executeUpdate() > 0) {
                respuesta = true;
            }
            cn.close();
        } catch (SQLException e) {
            System.out.println("Error al guardar Empleado: " + e);
        }
        return respuesta;
    }

    /** metodo para actualizar un Empleado */
    public boolean actualizar(Empleado objeto, int codEmpleado) {
        boolean respuesta = false;
        Connection cn = conexion.conectar();
        try {
            PreparedStatement consulta = cn.prepareStatement("update Empleado set nombre=?, apellido = ?, dni=?, correo=?, celular=? where codEmpleado=?");
            consulta.setString(1, objeto.getNombre());
            consulta.setString(2, objeto.getApellido());
            consulta.setInt(3, objeto.getDni());
            consulta.setInt(4, objeto.getCorreo());
            consulta.setString(5, objeto.getCorreo());
            consulta.setInt(6, objeto.getCelular());
            consulta.setInt(7, codEmpleado);

            if (consulta.executeUpdate() > 0) {
                respuesta = true;
            }
            cn.close();
        } catch (SQLException e) {
            System.out.println("Error al actualizar Empleado: " + e);
        }
        return respuesta;
    }

    /**
     * *****
     * metodo para eliminar un Empleado
     */
}
```

Este código Java es un controlador llamado "Controlador_Empleado" que se utiliza para interactuar con registros de empleados en una base de datos. El controlador proporciona métodos para realizar operaciones como guardar, actualizar, eliminar y buscar empleados. Aquí hay un resumen de los métodos y sus funciones:

- **guardar (Empleado objeto):** Este método se utiliza para guardar un nuevo registro de empleado en la base de datos.
- **actualizar (Empleado objeto, int codEmpleado):** Actualiza la información de un empleado existente en la base de datos utilizando su código de empleado.
- **eliminar (int dni):** Elimina un empleado de la base de datos en función de su número de identificación (DNI).
- **existeEmpleado(String dni):** Verifica si un empleado con un número de identificación (DNI) específico existe en la base de datos.
- **buscarPorEspecialidad(String especialidad):** Busca empleados por su especialidad y devuelve una lista de empleados que coinciden con la especialidad proporcionada. Este método realiza una consulta compleja que involucra la tabla "Empleado" y "Especialidad" para obtener información adicional sobre la especialidad de los empleados.

El código también maneja excepciones de SQL y asegura que los recursos, como las conexiones a la base de datos, se cierren adecuadamente en el bloque finally para evitar problemas de fuga de recursos.

C. Especialidad:

```

public class Controlador_Especialidad {

    /**
     * *****
     * metodo para guardar una nueva Especialidad
     * *****
     */
    public boolean guardar(Especialidad objeto) {
        boolean respuesta = false;
        Connection cn = Conexion.conexion.conectar();
        try {

            PreparedStatement consulta = cn.prepareStatement("insert into Especialidad values(?,?)");
            consulta.setInt(1, objeto.getId());
            consulta.setString(2, objeto.getNombre());

            if (consulta.executeUpdate() > 0) {
                respuesta = true;
            }

            cn.close();

        } catch (SQLException e) {
            System.out.println("Error al guardar Especialidad: " + e);
        }

        return respuesta;
    }

    /**
     * *****
     * metodo para consultar si la Especialidad registrado ya existe
     * *****
     */
    public boolean existeEspecialidad(String Nombre) {
        boolean respuesta = false;
        String sql = "select Nombres from Especialidad where Nombres = '" + Nombre + "'";
        Statement st;

        try {
            Connection cn = Conexion.conectar();
            st = cn.createStatement();
            ResultSet rs = st.executeQuery(sql);
            while (rs.next()) {
                respuesta = true;
            }

        } catch (SQLException e) {
            System.out.println("Error al consultar Especialidad: " + e);
        }

        return respuesta;
    }

    /**
     * *****
     * metodo para actualizar una nueva Especialidad
     * *****
     */
    public boolean actualizar(Especialidad objeto, int codEspecialidad) {
        boolean respuesta = false;
        Connection cn = Conexion.conexion.conectar();
        try {

            PreparedStatement consulta = cn.prepareStatement("update Especialidad set nombre=? where id=?");
            consulta.setString(1, objeto.getNombre());
            consulta.setInt(2, objeto.getId());

        } catch (SQLException e) {
            System.out.println("Error al actualizar Especialidad: " + e);
        }

        return respuesta;
    }
}

```

Este código Java es un controlador llamado "Controlador_Especialidad" que se utiliza para interactuar con registros de especialidades en una base de datos. El controlador proporciona métodos para realizar operaciones como guardar, actualizar, eliminar y verificar si una especialidad existe. Aquí hay un resumen de los métodos y sus funciones:

- **guardar(Especialidad objeto):** Este método se utiliza para guardar una nueva especialidad en la base de datos.
- **existeEspecialidad(String Nombre):** Verifica si una especialidad con un nombre específico ya existe en la base de datos.
- **actualizar(Especialidad objeto, int codEspecialidad):** Actualiza la información de una especialidad existente en la base de datos utilizando su código de especialidad.
- **eliminar(int codEspecialidad):** Elimina una especialidad de la base de datos en función de su código de especialidad.

El código maneja excepciones de SQL y asegura que los recursos, como las conexiones a la base de datos, se cierren adecuadamente en el bloque finally para evitar problemas de fuga de recursos.

C. Ficha Medica:

```

public boolean guardar(FichaMedica objeto) {
    boolean respuesta = false;
    Connection cn = conexion.conectar();
    try {
        PreparedStatement consulta = cn.prepareStatement("insert into FichaMedica values(?, ?, ?, ?, ?, ?, ?)");
        consulta.setInt(1, 0); //cod
        consulta.setInt(2, objeto.getCodPaciente());
        consulta.setString(3, objeto.getExamenes());
        consulta.setString(4, objeto.getAlergias());
        consulta.setString(5, objeto.getFicha_visita());
        consulta.setString(6, objeto.getDiagnostico());
        consulta.setString(7, objeto.getTratamiento());
        if (consulta.executeUpdate() > 0) {
            respuesta = true;
        }
        cn.close();
    } catch (SQLException e) {
        System.out.println("Error al guardar cliente: " + e);
    }
    return respuesta;
}

public boolean existeFichaMedica(String dni) {
    boolean respuesta = false;
    String sql = "select DNI from FichaMedica where DNI = '" + dni + "'";
    Statement st;
    try {
        Connection cn = conexion.conectar();
        st = cn.createStatement();
        ResultSet rs = st.executeQuery(sql);
        while (rs.next()) {
            respuesta = true;
        }
    } catch (SQLException e) {
        System.out.println("Error al consultar FichaMedica: " + e);
    }
    return respuesta;
}

public boolean existeFichaMedica(int dni) {
    boolean respuesta = false;
    String sql = "select DNI from FichaMedica where DNI = '" + dni + "'";
    Statement st;
    try {
        Connection cn = conexion.conectar();
        st = cn.createStatement();
        ResultSet rs = st.executeQuery(sql);
        while (rs.next()) {
            respuesta = true;
        }
    } catch (SQLException e) {
        System.out.println("Error al consultar FichaMedica: " + e);
    }
    return respuesta;
}
}

```

Este código Java es un controlador llamado "Controlador_FichaMedica" que se utiliza para interactuar con registros de fichas médicas en una base de datos. El controlador proporciona métodos para realizar operaciones como guardar una nueva ficha médica y verificar si una ficha

médica ya existe en función del número de identificación del paciente. Aquí hay un resumen de los métodos y sus funciones:

- **guardar(FichaMedica objeto):** Este método se utiliza para guardar una nueva ficha médica en la base de datos. Prepara una consulta SQL para insertar los detalles de la ficha médica, como los exámenes, alergias, diagnóstico, etc., y luego ejecuta la consulta en la base de datos.
- **existeFichaMedica(String dni):** Verifica si existe una ficha médica en la base de datos asociada a un paciente con un número de identificación (DNI) específico. Realiza una consulta SQL para buscar un DNI en la tabla de fichas médicas.
- **existeFichaMedica(int dni):** Similar al método anterior, verifica la existencia de una ficha médica en función de un número de identificación (DNI) proporcionado como un valor entero.

El código maneja excepciones de SQL y asegura que los recursos, como las conexiones a la base de datos, se cierren adecuadamente en el bloque finally para evitar problemas de fuga de recursos.

C. Paciente:


```

public class Controlador_Paciente {

    /** metodo para guardar un paciente */
    public boolean guardar(Paciente objeto) {
        boolean respuesta = false;
        Connection cn = conexion.conectar();
        try {
            PreparedStatement consulta = cn.prepareStatement("insert into Paciente values(?,?,?,?",
            consulta.setInt(1, 0); //cod
            consulta.setString(2, objeto.getNombre());
            consulta.setString(3, objeto.getApellido());
            consulta.setString(4, objeto.getFecha_nacimiento());
            consulta.setInt(5, objeto.getDni());
            consulta.setInt(6, objeto.getCelular());
            consulta.setString(7, objeto.getSexo());
            if (consulta.executeUpdate() > 0) {
                respuesta = true;
            }
            cn.close();
        } catch (SQLException e) {
            System.out.println("Error al guardar Paciente: " + e);
        }
        return respuesta;
    }

    /** metodo para actualizar un paciente */

    public boolean actualizar(Paciente objeto, int codPaciente) {
        boolean respuesta = false;
        Connection cn = conexion.conectar();
        try {
            PreparedStatement consulta = cn.prepareStatement("update Paciente set nombre=?, apellido =
            consulta.setString(1, objeto.getNombre());
            consulta.setString(2, objeto.getApellido());
            consulta.setString(3, objeto.getFecha_nacimiento());
            consulta.setInt(4, objeto.getDni());
            consulta.setInt(5, objeto.getCelular());
            consulta.setString(6, objeto.getSexo());

            if (consulta.executeUpdate() > 0) {
                respuesta = true;
            }
            cn.close();
        } catch (SQLException e) {
            System.out.println("Error al actualizar Paciente: " + e);
        }
        return respuesta;
    }

    /**
     * *****
     * metodo para eliminar un cliente
     * *****
     */
    public boolean eliminar(int codPaciente) {
        boolean respuesta = false;
        Connection cn = conexion.conectar();
        try {
            PreparedStatement consulta = cn.prepareStatement(
                "delete from Paciente where codPaciente =" + codPaciente + " ");
            consulta.executeUpdate();

            if (consulta.executeUpdate() > 0) {
                respuesta = true;
            }
        }
    }
}

```

Este código Java es un controlador llamado "Controlador_Paciente" que se utiliza para interactuar con registros de pacientes en una base de datos. El controlador proporciona métodos para realizar operaciones como guardar un nuevo paciente, actualizar la información de un paciente, eliminar un paciente y verificar si un paciente existe. Además, también proporciona un método para buscar a un paciente por su número de identificación (DNI). Aquí hay un resumen de los métodos y sus funciones:

- **guardar(Paciente objeto):** Este método se utiliza para guardar un nuevo registro de paciente en la base de datos. Prepara una consulta SQL para insertar los detalles del paciente, como el nombre, apellido, fecha de nacimiento, DNI, celular y sexo, y luego ejecuta la consulta en la base de datos.
- **actualizar(Paciente objeto, int codPaciente):** Actualiza la información de un paciente existente en la base de datos utilizando su código de paciente. Prepara una consulta SQL para actualizar los detalles del paciente y luego ejecuta la consulta en la base de datos.
- **eliminar(int codPaciente):** Elimina un paciente de la base de datos en función de su código de paciente. Prepara una consulta SQL de eliminación y la ejecuta en la base de datos.
- **existePaciente(String dni):** Verifica si existe un paciente en la base de datos asociado a un número de identificación (DNI) específico. Realiza una consulta SQL para buscar un DNI en la tabla de pacientes.
- **buscarPorDNI(int dni):** Busca un paciente en la base de datos por su número de identificación (DNI) y devuelve un objeto Paciente que contiene la información del paciente encontrado.

El código maneja excepciones de SQL y asegura que los recursos, como las conexiones a la base de datos, se cierren adecuadamente en el bloque finally para evitar problemas de fuga de recursos.

Bibliografía:

Anexo:

Prince Dent

Av. Puente Piedra 611 - Lima

Tel: 984051931 - 017634940

Ticket número: 00001335

Fecha: 1/07/2023 Hora: 11:54

Moneda: Soles

Cliente: JACKELIN RAFAEL

Cant	Item	Precio
	CONTROL ORTODONCIA	
1	(E)	160.000
Total pagado:		S/ 160.000

Tipo de pago: Efectivo

Cajero: Dr. Jason Principe

Gracias



Prince Dent
Av. Puente Piedra 611 , Puente Piedra, Lima
Tel: 984051931 - 017634940

Paciente: JACKELIN RAFAEL

Impreso el 10 de Junio 2022 a las 04:41

Servicio	Cant.	Precio U.	Dcto	Subtotal	Pagado	Por pagar	Coment
SELLANTES	2	S/ 40.00	0.00 %	80.00	80.00	0.00	pieza
RESTAURACION SIMPLE	1	S/ 60.00	0.00 %	60.00	60.00	0.00	piez
DESTARTRAJE + PROFILAXIS (1 CITA)	1	S/ 80.00	0.00 %	80.00	80.00	0.00	
				Total:	S/	220.00	
				Pagado:	S/	220.00	
				Por pagar:	S/	0.00	



Prince Dent
Av. Puente Piedra 611 , Puente Piedra, Lima
Tel: 984051931 - 017634940

Paciente: JACKELIN RAFAEL

Impreso el 13 de Agosto 2022 a las 11:02

Servicio	Cant.	Precio U.	Dcto	Subtotal	Pagado	Por pagar	Comentar
CONTROL ORTODONCIA (E)	1	S/ 160.00	0.00 %	160.00	160.00	0.00	
				Total:	S/	160.00	
				Pagado:	S/	160.00	
				Por pagar:	S/	0.00	



Prince Dent
Av. Puente Piedra 611 , Puente Piedra, Lima
Tel: 984051931 - 017634940

Paciente: JACKELIN RAFAEL

Impreso el 12 de Julio 2022 a las 03:31

Servicio	Cant.	Precio U.	Dcto	Subtotal	Pagado	Por pagar	Comentar
CONTROL ORTODONCIA (E)	1	S/ 160.00	0.00 %	160.00	160.00	0.00	
							Total: S/ 160.00
							Pagado: S/ 160.00
							Por pagar: S/ 0.00

Conclusiones:

- Un programa de software puede ayudar a la automatización de procesos clave, como la gestión de citas, el historial clínico y el inventario; por lo que es esencial para aumentar la eficiencia.
- La integración de diferentes funciones en una plataforma única facilita la gestión y el acceso a la información.
- El programa no solo beneficia al personal de la clínica simplificando sus tareas, sino que también mejora la atención y la satisfacción de los pacientes, lo que es fundamental para el éxito a largo plazo de la clínica dental.



Herramientas Utilizadas:

Balsamiq:

[New Wireframe 7 copy | Your First Project | Clinica dental | Balsamiq Cloud](#)

GitHub:

<https://github.com/77ALDO77/ProyectoAED.git>

Trello:

<https://trello.com/invite/b/UiMRV4l8/ATTIae4e7744fd0f0653ad91cdd39e6baca125CB89>

[FB/desarrollo-de-aplicativo](#)