IES El Rincón

Proyecto FullStack

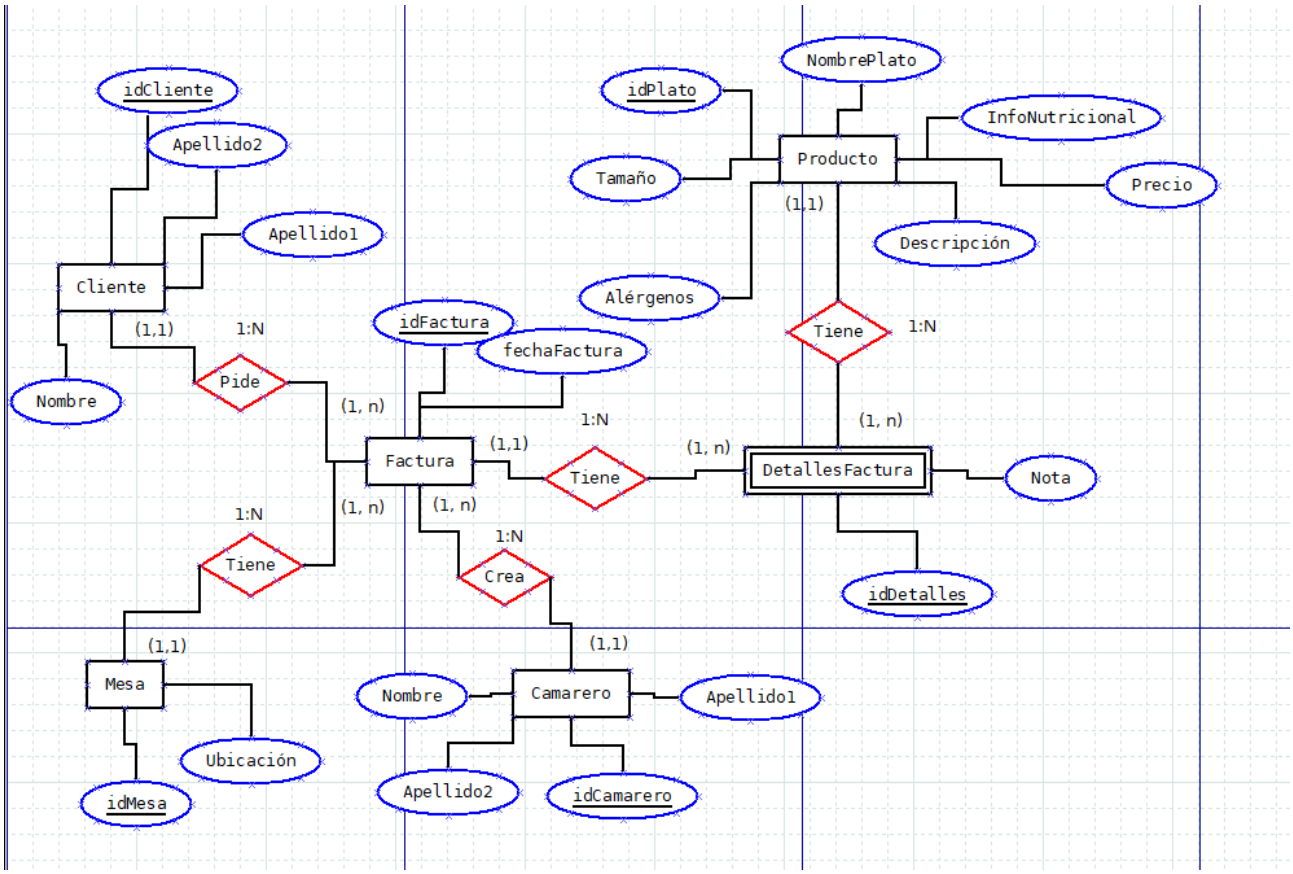
AUTOR

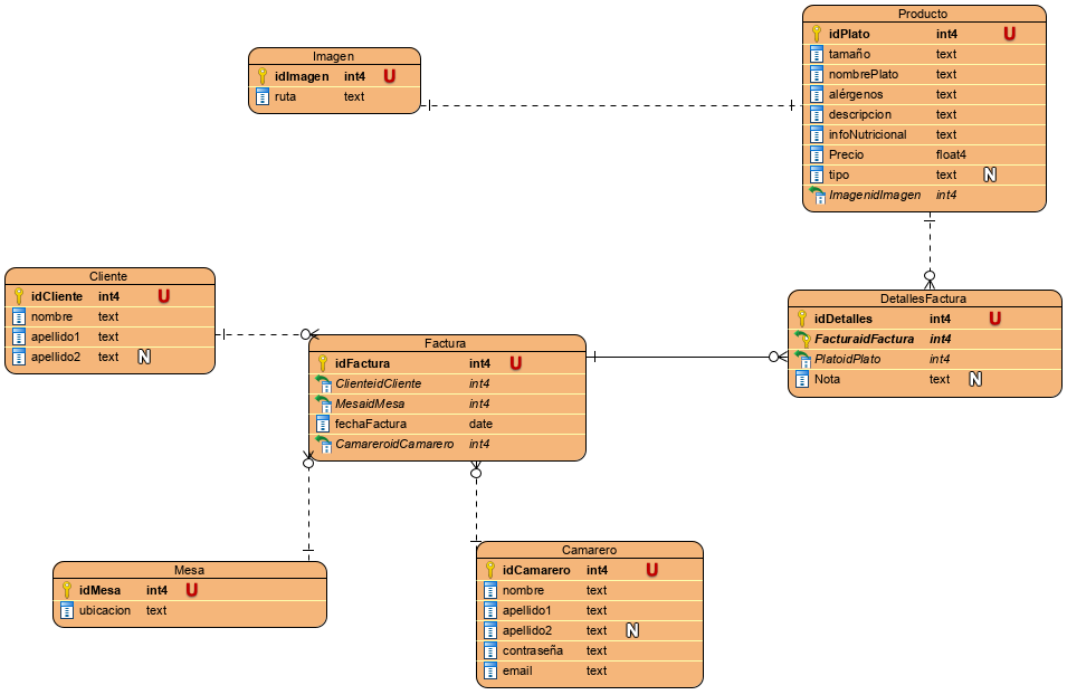
Roberto Carlos Ascanio Falcón

1. Pequeña descripción de la aplicación que se va desarrollar, diagrama E/R. que ilustre el diseño físico correspondiente y pila tecnológica a usar.

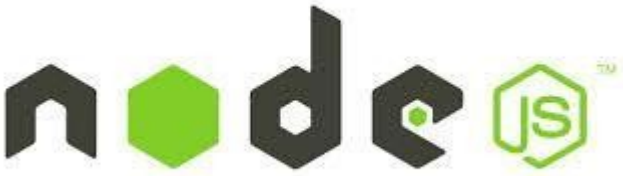
La aplicación FullStack que voy a desarrolar será para una empresa de restauración. Se usará en hardware como tablets, o una aplicación a la que los clientes puedan tener acceso. Con ella podrán hacer los pedidos directamente desde sus mesas, ahorrandonos el papeleo de las notas de un camarero, y el informe del pedido llega directamente a la cocina, con la información del cliente y la mesa que ocupa. Además, en todo momento el cliente podrá ver el menú, y en cada plato será capaz de ver información extra, como el precio, nombre, una descripción de este, podrá elegir el tamaño, etc. Podrá elegir varios productos para luego confirmar su pedido. Además, será capaz de eliminar productos de su pedido antes de confirmar este, y será capaz de modificar los productos también.

Diagrama ER de la aplicación:

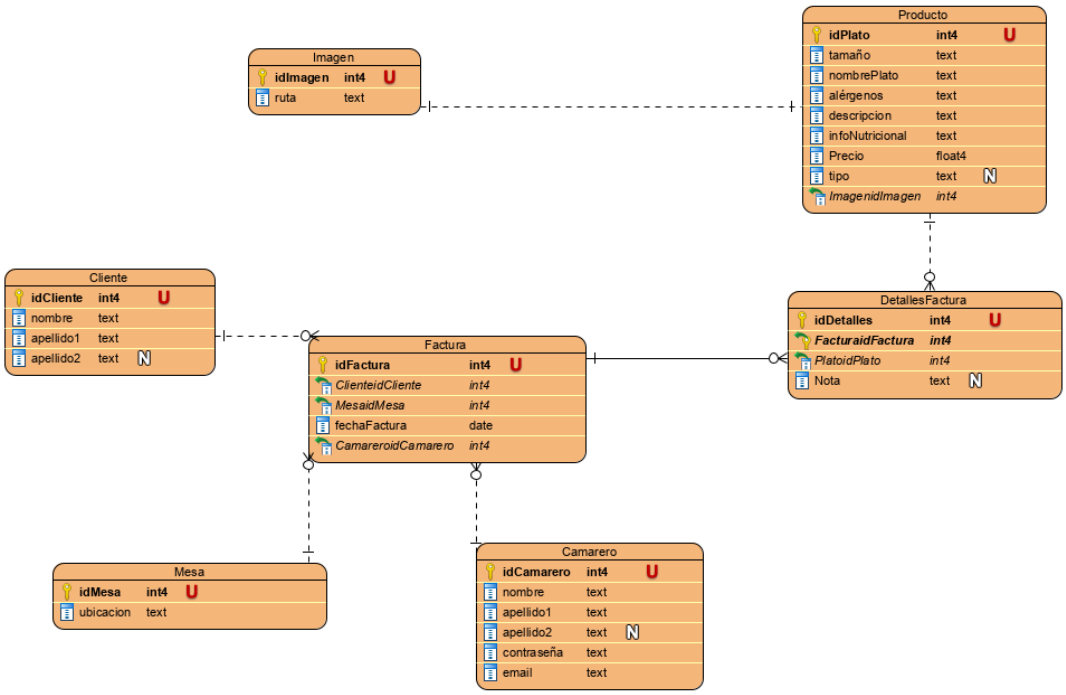


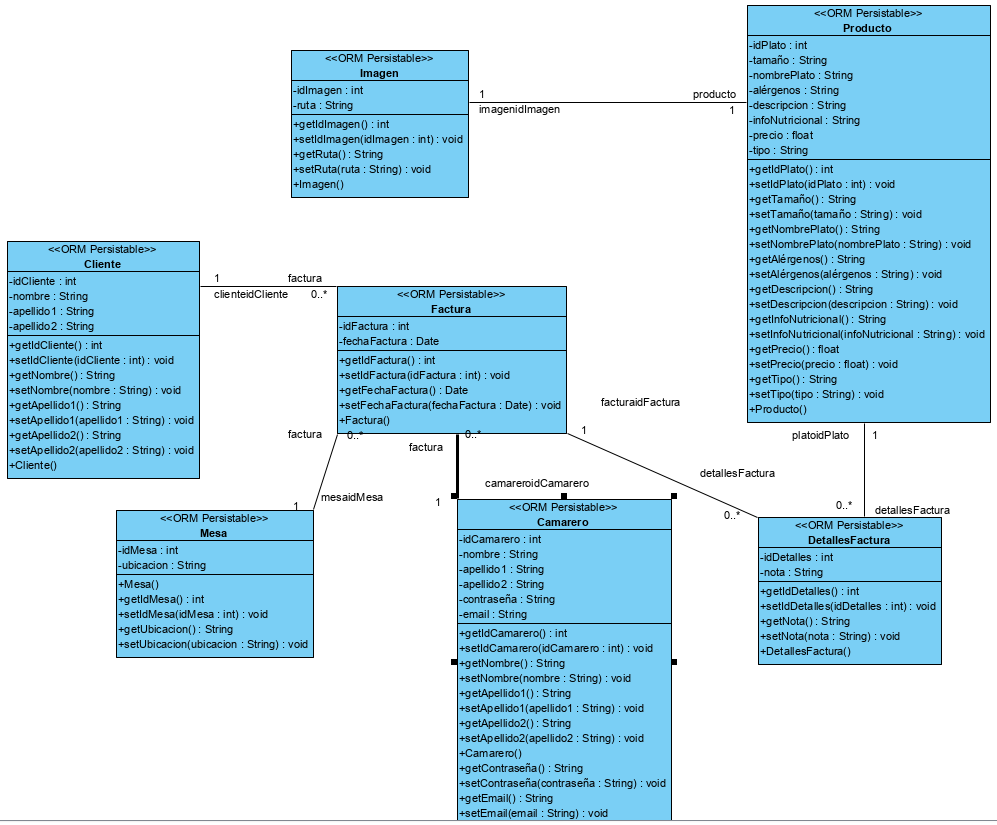


Pila tecnólogica que se va a usar en el proyecto:
Para la base de datos usaremos PostgreSQL, y para el mapeado ORM Sequelize. Además, para el servidor usaremos Node.js e Ionic para la aplicación



2. Modelo de datos representando las clases gráficamente con UML.





3. Descripción de cada tabla/entidad del modelo de datos, relaciones y de cada uno de sus atributos. Ilustrar el diseño físico con el diagrama de las tablas y relaciones según el SGBD

Empezamos describiendo nuestras tablas, sus relaciones y cada uno de sus atributos:

-**Tabla Clientes:** En esta tabla almacenaremos los clientes del restaurante, con los datos básicos de ellos como sus atributos/columnas. En este caso serán:

- **idCliente:** Almacenará un identificador único para cada cliente, en forma de int. No puede tener un valor nulo. Es la **clave** primaria de la tabla.
- **Nombre:** Como indica, almacenará el nombre del cliente del que guardamos los datos. No puede ser nulo, es un campo requerido y es tipo varchar.
- **Apellido1:** Almacena el primer apellido del cliente. No será nulo, dado que se requerirá al menos en la información que un apellido sea obligatorio y es tipo varchar.
- **Apellido2:** Almacenará el segundo apellido del cliente. El segundo apellido será opcional, por lo que sí podrá ser nulo y es tipo varchar.

La tabla Clientes tendrá una relación de cardinalidad **1:N** con la Tabla Factura, pasándole el idCliente como clave foránea a esta tabla. La participación será (1,1) por parte de cliente, (1,n) por parte de Factura.

-**Tabla Mesa:** En esta tabla almacenaremos información sobre las mesas del restaurante, que nos ayudará luego a saber ubicarlas y saber donde está el cliente. Sus atributos son:

- **idMesa**: Almacenará un identificador único para cada mesa, en forma de int. Equivalentes al número de esa mesa como en un restaurante. No puede ser nulo. Es la **clave** primaria de la tabla.
- **ubicación:** Guarda la información respecto a donde se encuentra esa mesa en el restaurante para que sea más fácil encontrarla a la hora de llevar la comida. No puede ser nulo y es tipo varchar.

La tabla Mesa estará relacionada con Factura, con una cardinalidad **1:N**, le pasa idMesa como clave foránea. La participación será (1,1) por parte de Mesa, (1,n) por parte de Factura.

-**Tabla Camarero:** Esta tabla almacena la información básica de los camareros empleados por el restaurante, para así tener a la hora de hacer la factura el id del camarero que atendió al cliente. Los atributos de esta tabla son:

- **idCamarero**: Almacenará un identificador único para cada camarero, en forma de int. No puede ser nulo. Además, es la **clave primaria** de la tabla.
- **Nombre:** Como indica, almacenará el nombre del camarero del que guardamos los datos. No puede ser nulo, es un campo requerido y es tipo varchar.

- **Apellido1:** Almacena el primer apellido del camarero. No será nulo, dado que se requerirá al menos en la información que un apellido sea obligatorio y es tipo varchar.
- **Apellido2:** Almacenará el segundo apellido del camarero. El segundo apellido será opcional, por lo que sí podrá ser nulo y es tipo varchar.
- **Email:** El email de login del camarero. Será tipo text y no es nulo
- **Password:** La contraseña del camarero para el login. Tipo text y no es nulo

Como hemos explicado en la descripción de la tabla, la relación que tiene la tabla Camarero será con la tabla Factura, pasándole a esta el idCamarero debido a la cardinalidad de la relación, que es **1:N**. La participación será (1,1) por parte de Camarero, (1,n) por parte de Factura.

-Tabla Factura: En esta tabla recogeremos la información de las facturas/pedidos de los clientes. Los atributos que vamos a guardar en la tabla serán:

- **idFactura:** Será un identificador único para cada factura, en forma de int. No puede ser nulo. Es la **clave** primaria de la tabla.
- **FechaFactura:** Guardará la fecha en la que se realizó esa factura/pedido. No puede ser nulo, es de tipo Date.
- **Atributos foráneos que vienen de las entidades dichas anteriormente, es decir, idCliente, idMesa e idCamarero (todos tipo int).**

La tabla tiene 4 relaciones. 3 de ellas ya las hemos nombrado en los apartados anteriores, son las relaciones 1:N entre Mesa, Camarero y Cliente. La otra es una relación de cardinalidad **1:N** entre la tabla Factura y la tabla DetallesFactura, pasándole a DetallesFactura el atributo idFactura como clave foránea. La participación es (1,1) por parte de Factura y (1,n) por parte de DetallesFactura.

-Tabla DetallesFactura: En esta tabla recogemos los detalles de cada factura, lo que será cada pedido/producto individualmente. Esta tabla es **débil**, dado que depende de la tabla Factura para existir. Los atributos que encontraremos aquí son:

- **idDetalles:** Almacenará un identificador único para cada “detalle”, en forma de int. No puede ser nulo. Es la **clave** primaria de la tabla, en conjunto con el que nombraremos más adelante, **idFactura** (que proviene como clave foránea de la tabla Factura)
- **Nota:** Guarda los detalles que apuntará el cliente sobre el producto, como que quiere quitarle ingredientes al producto escogido, etc. Puede ser nulo y es tipo varchar.
- **IdFactura:** Clave foránea proveniente de la relación 1:N que tiene la tabla con la tabla Factura, es tipo int. Es también **clave primaria**, junto a **idDetalles.**
- **IdPlato:** Clave foránea que proviene de la relación que tiene con la tabla **Producto** (que mencionaremos más adelante). No puede ser nulo y tipo int.

Esta tabla tiene 2 relaciones. Una ya hablamos de ella previamente, que es con la Tabla Factura, cardinalidad **1:N.** La segunda relación es con la Tabla Producto, y también tiene cardinalidad **1:N.** La participación sería (1,1) por parte de la Tabla Producto, y (1,n) por la Tabla DetallesFactura.

-Tabla Producto: Última tabla de nuestra base de datos. En esta recogeremos los varios datos que necesitamos de los distintos productos que venderá nuestro restaurante. Estos atributos serán:

- **idPlato:** Almacenará un identificador único para cada producto, en forma de int. No puede ser nulo. Es la **clave** primaria de la tabla.

- **Tamaño:** Guardará el “tamaño” (grande, mediano, pequeño) del que será nuestro producto. No puede ser nulo y es tipo varchar.
- **NombrePlato:** Guarda el nombre del tipo de producto que se vende. No puede ser nulo y es tipo varchar.
- **Alérgenos:** Guarda información sobre los alérgenos que tiene el producto en concreto. No puede ser nulo y será tipo varchar.
- **Descripcion:** Guarda una descripción del producto en sí. No puede ser nulo y será tipo varchar.
- **InfoNutricional:** Guarda la información nutricional del producto en cuestión. No puede ser nulo y será tipo varchar.
- **Precio:** Guarda el precio del producto en cuestión. No puede ser nulo y será tipo varchar.
- **Idimagen:** Guarda la imagen correspondiente a ese producto. No es nulo.

La relación que tiene esta tabla es con la tabla DetallesFactura y con la tabla Imagen. Tiene una cardinalidad de **1:N**. La participación es **(1,1)** por parte de Producto y de **(1,n)** por parte de DetallesFactura. Con imagen es cardinalidad **1:N**, (1,n) por parte de producto y (1,1) por parte de Imagen

-Tabla Imagen: Última tabla del proyecto. Recoge las rutas de las imagenes que usaremos para nuestros productos. Sus atributos son:

- **idimagen:** Almacena un identificador único para cada imagen en forma de int. No puede ser nulo y es la clave primaria
- **ruta:** Almacena la ruta de la imagen que usa (debe estar en assets/images)

4. Codificación del ORM que gestiona el CRUD de los datos.

App.js

src > JS app.js > ...

```
import express, { json } from 'express';
import morgan from 'morgan';

//Importing routes
import clientsRoutes from './routes/clientes';
import tablesRoutes from './routes/mesas';
import waiterRoutes from './routes/camareros';
import productsRoutes from './routes/productos';
import detailRoutes from './routes/detalles';
import billRoutes from './routes/facturas';
import imageRoutes from './routes/imagenes';

//initialization
const app = express();

//Middlewares
app.use(morgan('dev'));
app.use(json());
var cors = require('cors');
app.use(cors());

//routes
app.use('/api/clientes',clientsRoutes);
app.use('/api/mesas',tablesRoutes);
app.use('/api/camareros', waiterRoutes);
app.use('/api/productos', productsRoutes);
app.use('/api/facturas', billRoutes);
app.use('/api/detalles', detailRoutes);
app.use('/api/imagenes', imageRoutes)

export default app;
```

index.js

```
import app from './app';
import '@babel/polyfill';

async function main(){
    await app.listen(4000);
    console.log('Server on port 4000');
}

main();
```

package.json

CONTROLADORES:

camarero

```
import Camareros from '../models/camareros'

export async function createWaiter(req, res) {
    const { nombre, apellido1, apellido2, contraseña, email } = req.body;
    try {
        let newCamarero = await Camareros.create({
            nombre: nombre,
            apellido1: apellido1,
            apellido2: apellido2,
            contraseña: contraseña,
            email: email
        }, {
            fields: ['nombre', 'apellido1', 'apellido2', 'contraseña', 'email']
        });
        if (newCamarero) {
            return res.json({
                message: 'Waiter registered',
                data: newCamarero
            });
        }

    } catch (error) {
        console.log(error);
        res.status(500).json({
            message: 'Something went wrong, waiter not registered',
            data: {}
        });
    }

}

export async function getWaiters(req, res) {
    try {
        const camareros = await Camareros.findAll();
        res.json({
            data: camareros
        });
    } catch (error) {

    }
```

```
export async function getOneWaiter(req, res) {
    try {
        const { id } = req.params;
        const camarero = await Camareros.findOne({
            where: {
                idcamarero: id
            }
        });
        res.json({
            data: camarero
        })
    } catch (error) {

    }
}

export async function deleteOneWaiter(req, res) {
    try {
        const { id } = req.params;
        const deletedWaiterRows = await Camareros.destroy({
            where: {
                idcamarero: id
            }
        });
        res.json({
            message: 'Deleted client',
            count: deletedWaiterRows
        })
    } catch (error) {

    }
}
```

```
export async function updateOneWaiter(req, res) {
    try {
        const { id } = req.params;
        const { nombre, apellido1, apellido2, contraseña, email } = req.body;

        const camarero = await Camareros.findAll({
            attributes: ['idcamarero','nombre', 'apellido1', 'apellido2', 'contraseña', 'email'],
            where: {
                idcamarero: id
            }
        });

        if(camarero.length > 0){
            camarero.forEach(async camarero =>{
                await camarero.update({
                    nombre: nombre,
                    apellido1: apellido1,
                    apellido2: apellido2,
                    contraseña: contraseña,
                    email: email
                });
            })
        }
        return res.json({
            message: 'Waiter updated!',
            data: camarero
        })
    } catch (error) {

    }
}
```

Las funciones son iguales para el resto de controladores, pero con los datos de cada tabla. Excepto especificaciones.

```
  1   import Clientes from '../models/clientes'
  2
  3 > export async function createClient(req, res) {···
 28   }
 29
 30 > export async function getClients(req, res) {···
 39   }
 40
 41 > export async function getOneClient(req, res) {···
 55   }
 56
 57 > export async function deleteOneClient(req, res) {···
 72   }
 73
 74 > export async function updateOneClient(req, res) {···
102   }
```

src > controllers > JS detalles.controller.js > ...

```
  1   import Detalles from '../models/detalles'
  2
  3 > export async function createDetails(req, res) {···
 28   }
 29
 30 > export async function getDetails(req, res) {···
 39   }
 40
 41 > export async function getOneDetail(req, res) {···
 59   }
 60
 61 > export async function deleteOneDetail(req, res) {···
 76   }
 77
 78 > export async function updateOneDetail(req, res) {···
106   }
107
108 > export async function getDetailsByBill(req, res) {···
115   }
116
117 > export async function getDetailsByProduct(req, res) {···
124   }
```

src > controllers > JS facturas.controller.js > getBillsByClient

```
  1   import Facturas from '../models/facturas'
  2
  3 > export async function createBill(req, res) {···
 29   }
 30
 31 > export async function getBills(req, res) {···
 42   }
 43
 44 > export async function getOneBill(req, res) {···
 59   }
 60
 61 > export async function deleteOneBill(req, res) {···
 76   }
 77
 78 > export async function updateOneBill(req, res) {···
107   }
108
109 > export async function getBillsByClient(req, res) {···
116   }
117
118 > export async function getBillsByTable(req, res) {···
125   }
126
127 > export async function getBillsByWaiter(req, res) {···
134   }
```

src > controllers > JS imagen.controller.js > ...

```
  1   import Imagenes from '../models/imagenes'
  2
  3 > export async function createImage(req, res) {···
 26   }
 27
 28 > export async function getImages(req, res) {···
 37   }
 38
 39 > export async function getOneImage(req, res) {···
 53   }
 54
 55 > export async function deleteOneImage(req, res) {···
 70   }
 71
 72 > export async function updateOneImage(req, res) {···
 98   }
```

src > controllers > JS mesas.controller.js > ...

```
  1   import Mesas from '../models/mesas'
  2
  3 > export async function createTable(req, res) {···
 26   }
 27
 28 > export async function getTables(req, res) {···
 37   }
 38
 39 > export async function getOneTable(req, res) {···
 53   }
 54
 55 > export async function deleteOneTable(req, res) {···
 70   }
 71
 72 > export async function updateOneTable(req, res) {···
 98   }
```

src > controllers > JS producto.controller.js > ...

```
  1   import Productos from '../models/productos'
  2
  3 > export async function createProduct(req, res) {···
 33   }
 34
 35 > export async function getProductByType(req, res){···
 49   }
 50
 51 > export async function getProducts(req, res) {···
 60   }
 61
 62 > export async function getOneProduct(req, res) {···
 76   }
 77
 78 > export async function deleteOneProduct(req, res) {···
 93   }
 94
 95 > export async function updateOneProduct(req, res) {···
128   }
```

Funciones específicas:
Detalles por factura y detalles por producto

```
108  export async function getDetailsByBill(req, res) {
109      const { facturaidfactura } = req.params;
110      const detalles = await Detalles.findAll({
111          attributes: ['iddetalles', 'facturaidfactura', 'platoidplato', 'nota'],
112          where: { facturaidfactura }
113      });
114      res.json({ detalles });
115  }
116
117  export async function getDetailsByProduct(req, res) {
118      const { platoidplato } = req.params;
119      const detalles = await Detalles.findAll({
120          attributes: ['iddetalles', 'facturaidfactura', 'platoidplato', 'nota'],
121          where: { platoidplato }
122      });
123      res.json({ detalles });
124  }
```

Facturas por cliente, por mesa, y por camareros

```
109  export async function getBillsByClient(req, res) {
110      const { clienteidcliente } = req.params;
111      const facturas = await Facturas.findAll({
112          attributes: ['idfactura', 'clienteidcliente', 'mesaidmesa', 'fechafactura', 'camareroidcamarero'],
113          where: { clienteidcliente }
114      });
115      res.json({ facturas });
116  }
117
118  export async function getBillsByTable(req, res) {
119      const { mesaidmesa } = req.params;
120      const facturas = await Facturas.findAll({
121          attributes: ['idfactura', 'clienteidcliente', 'mesaidmesa', 'fechafactura', 'camareroidcamarero'],
122          where: { mesaidmesa }
123      });
124      res.json({ facturas });
125  }
126
127  export async function getBillsByWaiter(req, res) {
128      const { camareroidcamarero } = req.params;
129      const facturas = await Facturas.findAll({
130          attributes: ['idfactura', 'clienteidcliente', 'mesaidmesa', 'fechafactura', 'camareroidcamarero'],
131          where: { camareroidcamarero }
132      });
133      res.json({ facturas });
134  }
```

Productos por tipo

```
35  export async function getProductByType(req, res){
36      try {
37          const {type} = req.params;
38          const productos = await Productos.findAll({
39                  attributes: ['idplato', 'tamaño', 'nombreplato', 'alérgenos', 'descripcion', 'infonutricional', 'precio', 'tipo', '
40                  where: {tipo : type }
41
42          });
43          res.json({
44              data: productos
45          })
46      } catch (error) {
47          console.log(error)
48      }
49  }
```

CONEXION A LA BASE DE DATOS

src > database > JS database.js > [e] sequelize

```
1   import Sequelize from 'sequelize';
2
3   export const sequelize = new Sequelize(
4       'postgres',
5       'postgres',
6       'elrincon1920',
7       {
8           host: 'localhost',
9           dialect: 'postgres',
10          pool:{
11              max: 5,
12              require: 30000,
13              idle: 10000
14          },
15          logging: false
16      }
17  )
```

MODELOS

Camareros

src > models > JS camareros.js > ...

```
import Sequelize from 'sequelize';
import { sequelize } from'../database/database';
import Facturas from './facturas';

const Camareros = sequelize.define('camarero',{
    idcamarero: {
        type: Sequelize.INTEGER,
        primaryKey: true
    },
    nombre: {
        type: Sequelize.TEXT
    },
    apellido1: {
        type: Sequelize.TEXT
    },
    apellido2: {
        type: Sequelize.TEXT
    },
    contraseña: {
        type: Sequelize.TEXT
    },
    email: {
        type: Sequelize.TEXT
    }
}, {
    timestamps: false,
    freezeTableName: true
});

Camareros.hasMany(Facturas, {foreignKey: 'camareroidcamareo', sourceKey:'idcamarero'});
Facturas.belongsTo(Camareros, {foreignKey: 'camareroidcamareo', sourceKey:'idcamarero'})
export default Camareros;
```

Clientes

src > models > JS clientes.js > ...

```
import Sequelize from 'sequelize';
import { sequelize } from'../database/database';
import Facturas from './facturas';

const Clientes = sequelize.define('cliente',{
    idcliente: {
        type: Sequelize.INTEGER,
        primaryKey: true
    },
    nombre: {
        type: Sequelize.TEXT
    },
    apellido1: {
        type: Sequelize.TEXT
    },
    apellido2: {
        type: Sequelize.TEXT
    }
}, {
    timestamps: false,
    freezeTableName: true
});

Clientes.hasMany(Facturas, {foreignKey: 'clienteidcliente', sourceKey:'idcliente'});
Facturas.belongsTo(Clientes, {foreignKey: 'clienteidcliente', sourceKey:'idcliente'})
export default Clientes;
```

Detalles

src > models > JS detalles.js > ...

```
import Sequelize from 'sequelize';
import { sequelize } from'../database/database';

const Detalles = sequelize.define('detallesfactura',{
    iddetalles: {
        type: Sequelize.INTEGER,
        primaryKey: true
    },
    facturaidfactura: {
        type: Sequelize.INTEGER,
        primaryKey: true
    },
    platoidplato: {
        type: Sequelize.INTEGER
    },
    nota: {
        type: Sequelize.TEXT
    }
}, {
    timestamps: false,
    freezeTableName: true
});

export default Detalles;
```

Facturas

src > models > JS facturas.js > ...

```
import Sequelize from 'sequelize';
import { sequelize } from'../database/database';
import Detalles from './detalles';

const Facturas = sequelize.define('factura',{
    idfactura: {
        type: Sequelize.INTEGER,
        primaryKey: true
    },
    clienteidcliente: {
        type: Sequelize.INTEGER
    },
    mesaidmesa: {
        type: Sequelize.INTEGER
    },
    fechafactura: {
        type: Sequelize.DATE
    },
    camareroidcamarero: {
        type: Sequelize.INTEGER
    }
}, {
    timestamps: false,
    freezeTableName: true
});
Facturas.hasMany(Detalles, {foreignKey: 'facturaidfactura', sourceKey:'idfactura'});
Detalles.belongsTo(Facturas, {foreignKey: 'facturaidfactura', sourceKey:'idfactura'});
export default Facturas;
```

Imagenes

src > models > JS imagenes.js > ...

```
import Sequelize from 'sequelize';
import { sequelize } from'../database/database';
import Productos from './productos';

const Imagenes = sequelize.define('imagen',{
    idimagen: {
        type: Sequelize.INTEGER,
        primaryKey: true
    },
    ruta: {
        type: Sequelize.TEXT
    }
}, {
    timestamps: false,
    freezeTableName: true
});

Imagenes.hasMany(Productos, {foreignKey: 'imagenidimagen', sourceKey:'idimagen'});
Productos.belongsTo(Imagenes, {foreignKey: 'imagenidimagen', sourceKey:'idimagen'})
export default Imagenes;
```

Mesas

src > models > JS mesas.js > ...

```
import Sequelize from 'sequelize';
import { sequelize } from'../database/database';

import Facturas from './facturas';

const Mesas = sequelize.define('mesa',{
    idmesa: {
        type: Sequelize.INTEGER,
        primaryKey: true
    },
    ubicacion: {
        type: Sequelize.TEXT
    }
}, {
    timestamps: false,
    freezeTableName: true
});

Mesas.hasMany(Facturas, { foreignKey: 'mesaidmesa', sourceKey: 'idmesa'});
Facturas.belongsTo(Mesas, { foreignKey: 'mesaidmesa', sourceKey: 'idmesa'});
export default Mesas;
```

Productos

src > models > JS productos.js > [o] Productos > infonutricional

```
import Sequelize from 'sequelize';
import { sequelize } from'../database/database';
import Detalles from './detalles';

const Productos = sequelize.define('producto',{
    idplato: {
        type: Sequelize.INTEGER,
        primaryKey: true
    },
    tamaño: {
        type: Sequelize.TEXT
    },
    nombreplato: {
        type: Sequelize.TEXT
    },
    alérgenos: {
        type: Sequelize.TEXT
    },
    descripcion: {
        type: Sequelize.TEXT
    },
    infonutricional: {
        type: Sequelize.TEXT
    },
    precio: {
        type: Sequelize.REAL
    },
    tipo: {
        type: Sequelize.TEXT
    },
    imagenidimagen: {
        type: Sequelize.INTEGER
    }
}, {
    timestamps: false,
    freezeTableName: true
});

Productos.hasMany(Detalles, {foreignKey: 'platoidplato', sourceKey:'idplato'});
Detalles.belongsTo(Productos, {foreignKey: 'platoidplato', sourceKey:'idplato'});
export default Productos;
```

RUTAS

camareros

```
import { Router} from 'express';
import { getWaiters, createWaiter, getOneWaiter, deleteOneWaiter, updateOneWaiter } from '../controllers/camarero.controller';
const router = Router();


// /api/camareros/
router.get('/', getWaiters);
router.post('/', createWaiter);

// /api/camareros/:id
router.get('/:id', getOneWaiter);
router.delete('/:id', deleteOneWaiter);
router.put('/:id', updateOneWaiter);

export default router;
```

clientes

src > routes > JS clientes.js > ...

```
import { Router} from 'express';
import { createClient, getClients, getOneClient, deleteOneClient, updateOneClient } from '../controllers/clientes.controller'
const router = Router();

// /api/clientes/
router.get('/', getClients);
router.post('/', createClient);

// /api/clientes/:id
router.get('/:id', getOneClient);
router.delete('/:id', deleteOneClient);
router.put('/:id', updateOneClient);

export default router;
```

detalles

src > routes > JS detalles.js > ...

```
import { Router} from 'express';
import { createDetails, getDetails, getOneDetail, deleteOneDetail, updateOneDetail, getDetailsByBill, getDetailsByProduct } from '../controllers/detalles.controller';
const router = Router();

//api/detalles
router.post('/', createDetails);
router.get('/', getDetails);
//api/detalles/:id
router.get('/:id', getOneDetail);
router.delete('/:id', deleteOneDetail);
router.put('/:id', updateOneDetail);

//api/detalles/facturas/:idfactura
router.get('/facturas/:facturaidfactura', getDetailsByBill);
//api/detalles/productos/:idproducto
router.get('/productos/:platoidplato', getDetailsByProduct);

export default router;
```

facturas

src > routes > JS facturas.js > ...

```
import { Router} from 'express';
import { getBillsByClient, getBillsByTable, getBillsByWaiter, createBill, getBills, getOneBill, deleteOneBill, updateOneBill } from '../controllers/facturas.controller
const router = Router();

//api/facturas
router.post('/', createBill);
router.get('/', getBills);

//api/facturas/:id
router.get('/:id', getOneBill);
router.delete('/:id', deleteOneBill);
router.put('/:id', updateOneBill);

//api/facturas/clientes/:clienteidcliente
router.get('/clientes/:clienteidcliente', getBillsByClient);
//api/facturas/mesas/:mesaidmesa
router.get('/mesas/:mesaidmesa', getBillsByTable);
//api/facturas/camareros/:camareroidcamarero
router.get('/camareros/:camareroidcamarero', getBillsByWaiter);

export default router;
```

imagenes

src > routes > JS imagenes.js > ...

```
import { Router} from 'express';
import { getOneImage, deleteOneImage, updateOneImage, createImage, getImages } from '../controllers/imagen.controller';
const router = Router();

//api/imagenes
router.post('/', createImage);
router.get('/', getImages);

//api/imagenes/:id
router.get('/:id', getOneImage);
router.delete('/:id', deleteOneImage);
router.put('/:id', updateOneImage);

export default router;
```

mesas

src > routes > JS mesas.js > ...

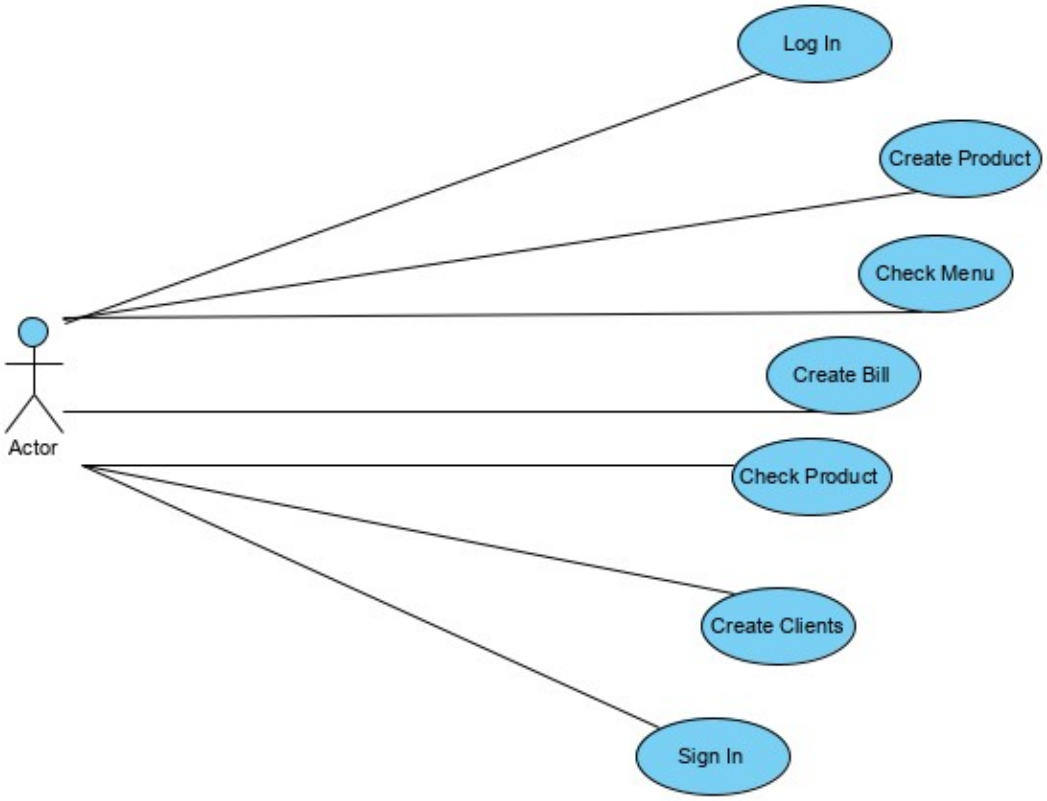
```
import { Router} from 'express';
import { getTables, createTable, getOneTable, deleteOneTable, updateOneTable } from '../controllers/mesas.controller';
const router = Router();

// /api/mesas
router.get('/', getTables);
router.post('/', createTable);

// /api/mesas/:id
router.get('/:id', getOneTable);
router.delete('/:id', deleteOneTable);
router.put('/:id', updateOneTable);

export default router;
```

productos

src > routes > JS productos.js > ...

```
import { Router} from 'express';
import { getProducts, createProduct, getOneProduct, deleteOneProduct, updateOneProduct, getProductByType } from '../controllers/producto.controller';
const router = Router();

// /api/productos
router.get('/', getProducts);
router.post('/', createProduct);

// /api/productos/:id
router.get('/:id', getOneProduct);
router.delete('/:id', deleteOneProduct);
router.put('/:id', updateOneProduct);


// /api/productos/tipo/:type
router.get('/tipo/:type', getProductByType);

export default router;
```

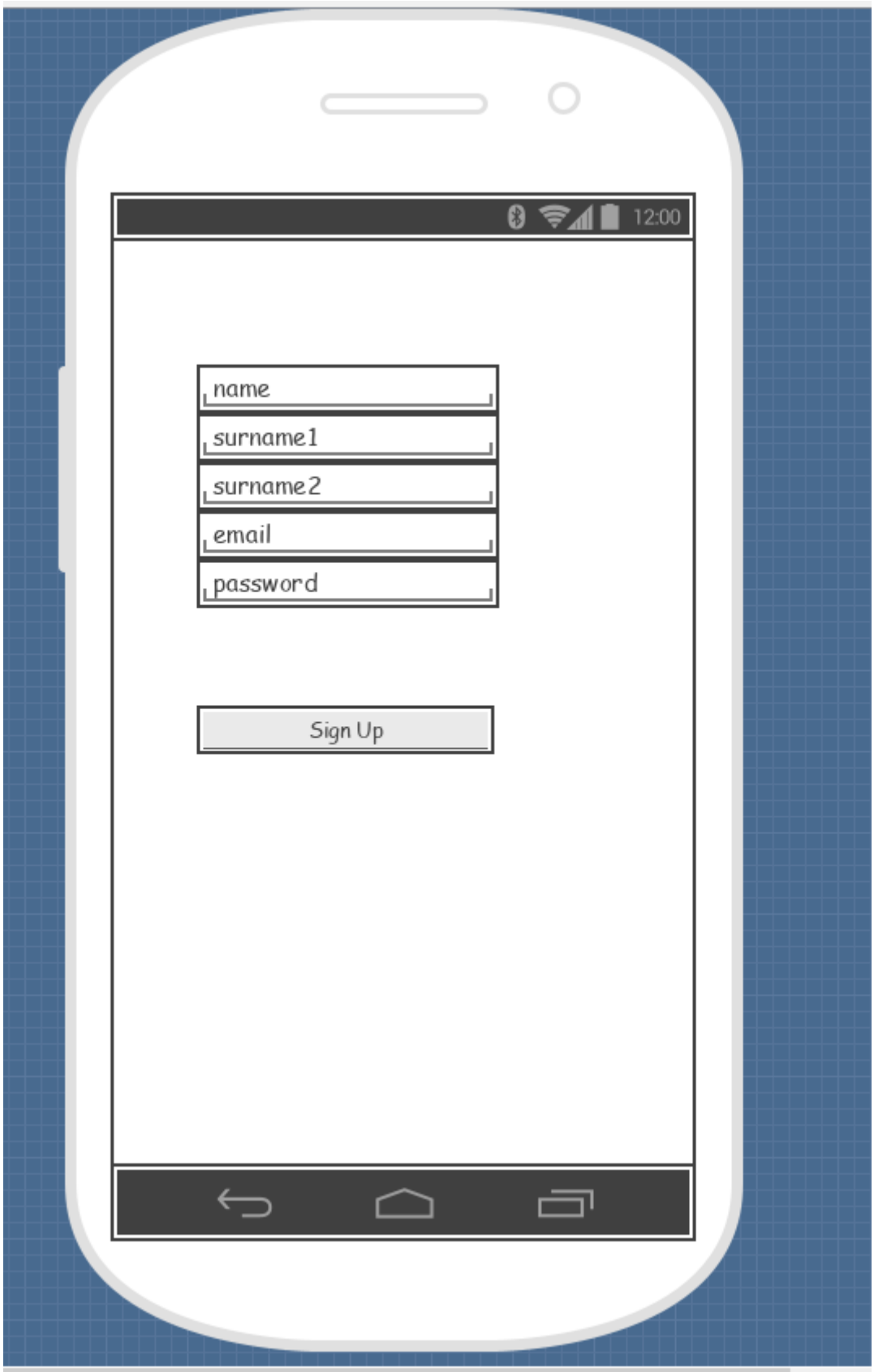
5. Descripción de la app mediante casos de uso



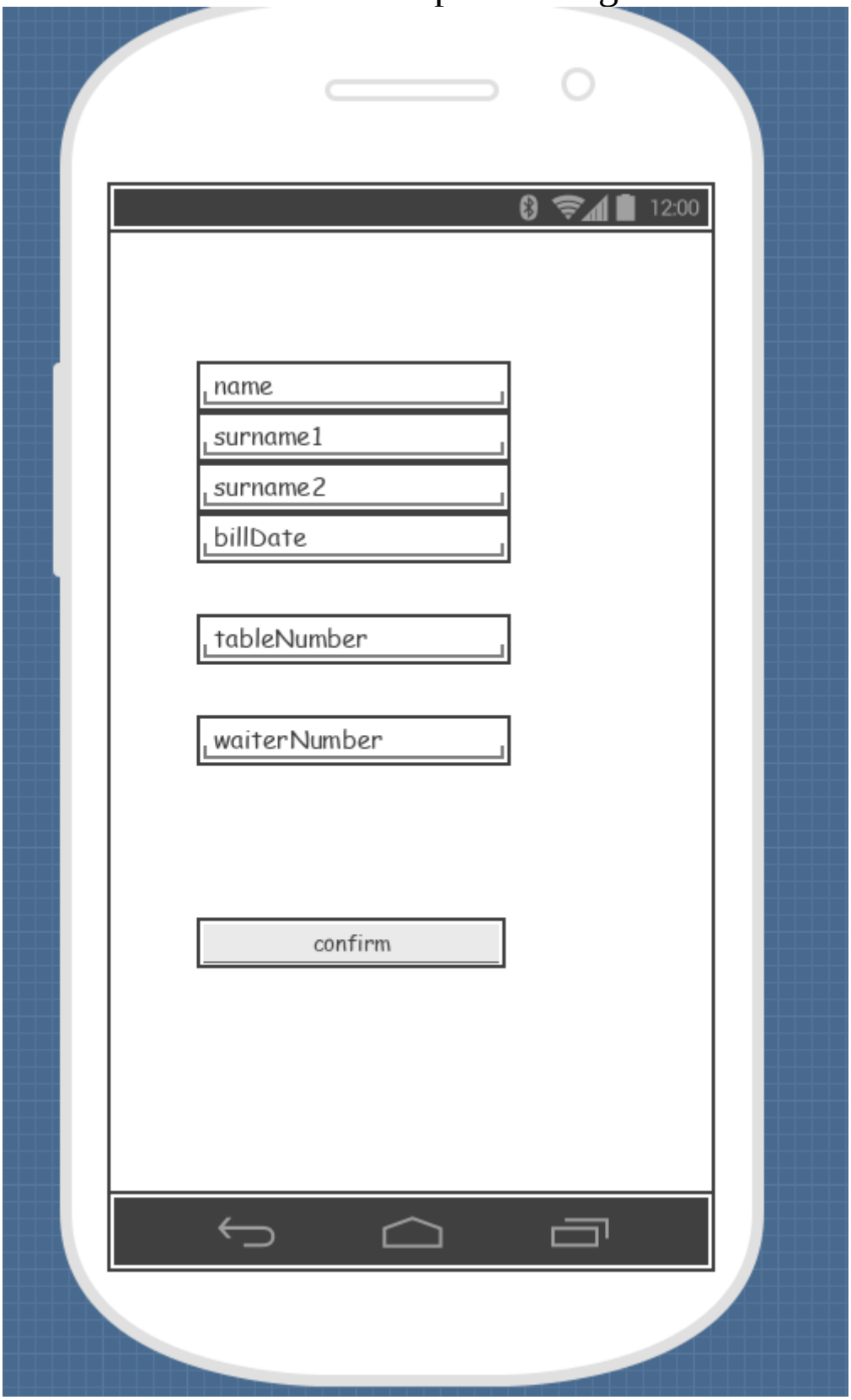
6. Descripción del diseño de la APP mediante Mockups

Login

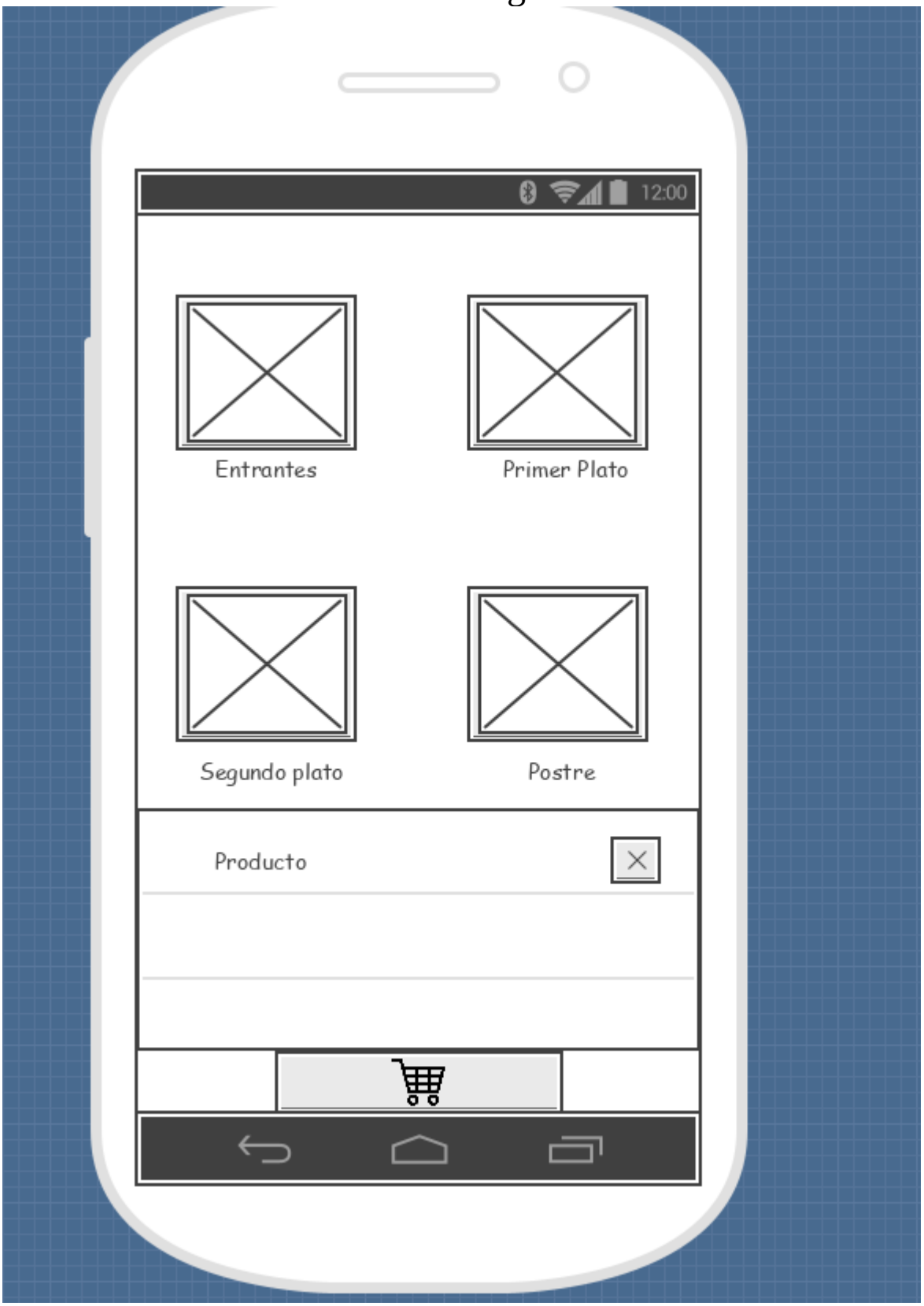
Sign In



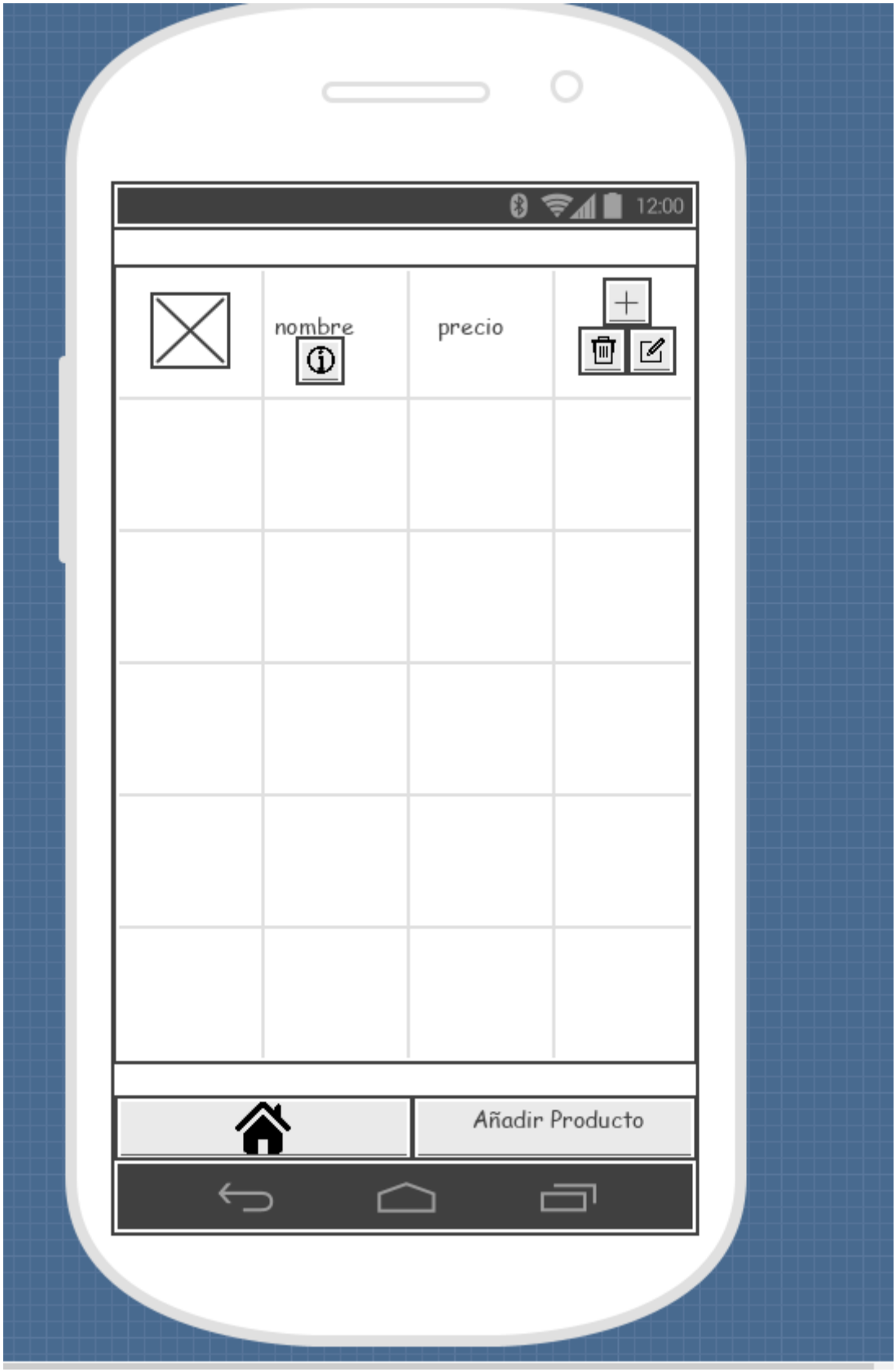
Crear Factura despues de logearse



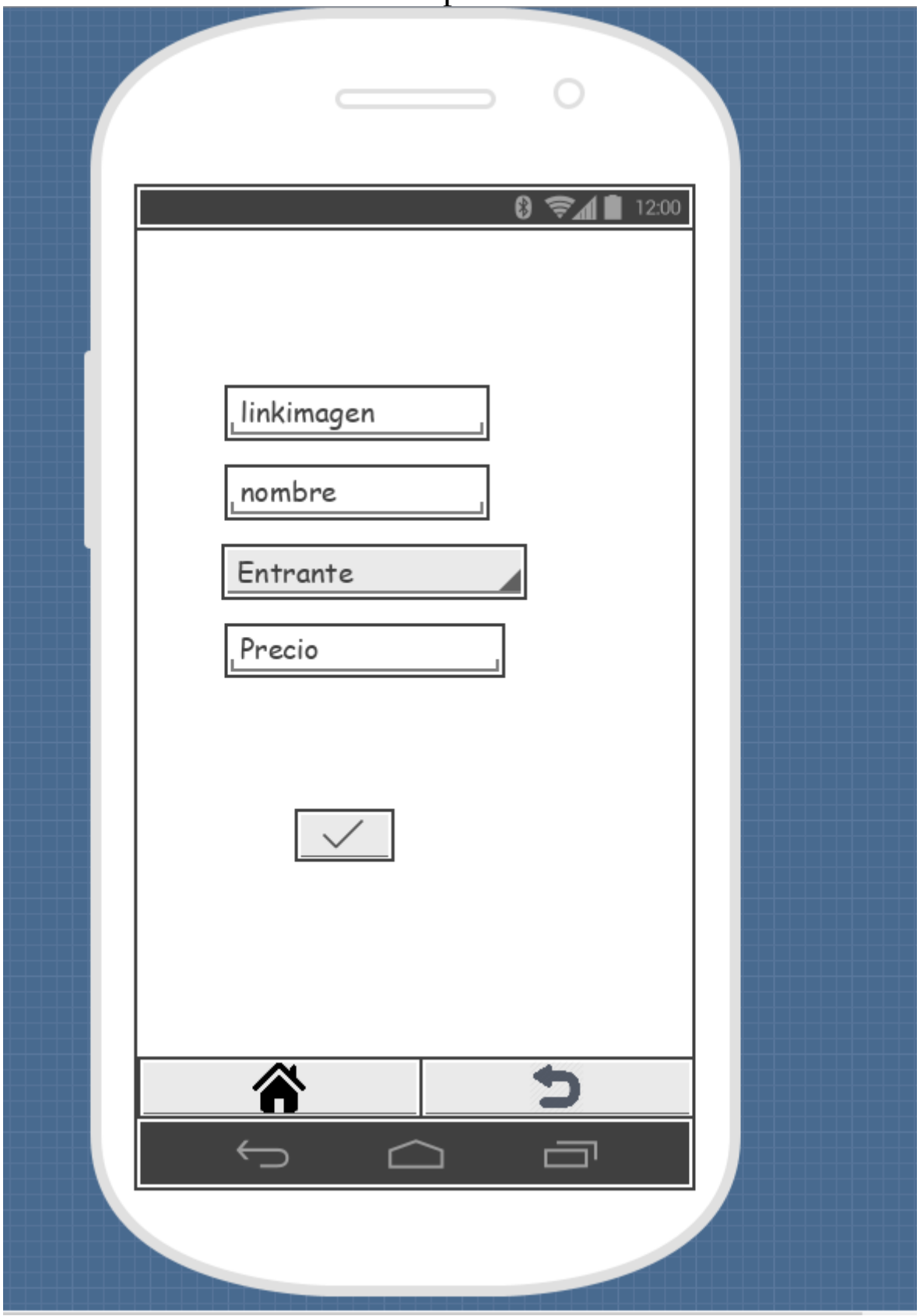
Main Page


12:00
Entrantes
Primer Plato
Segundo plato
Postre
Producto

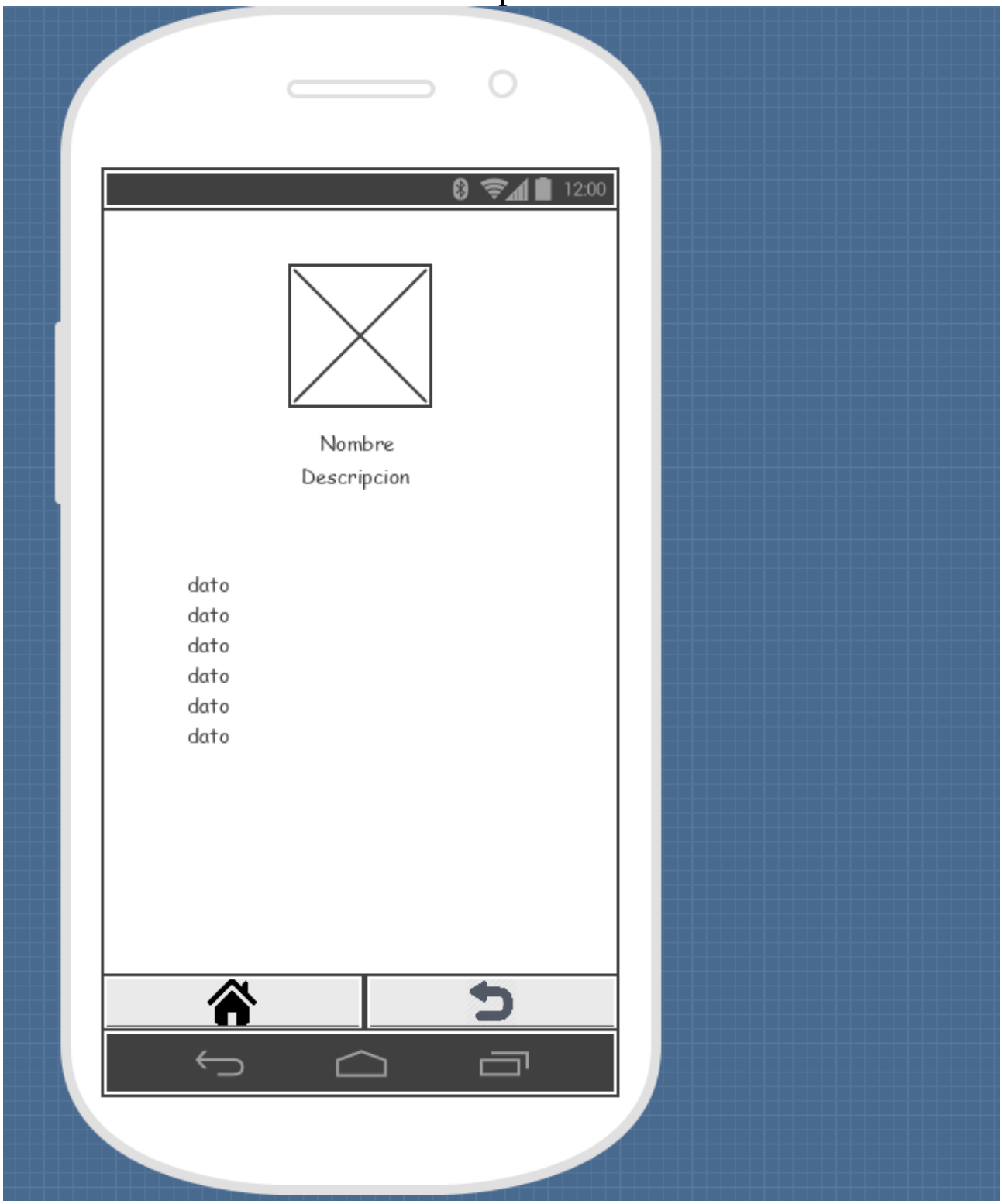
Menu



Añadir productos



Info del producto

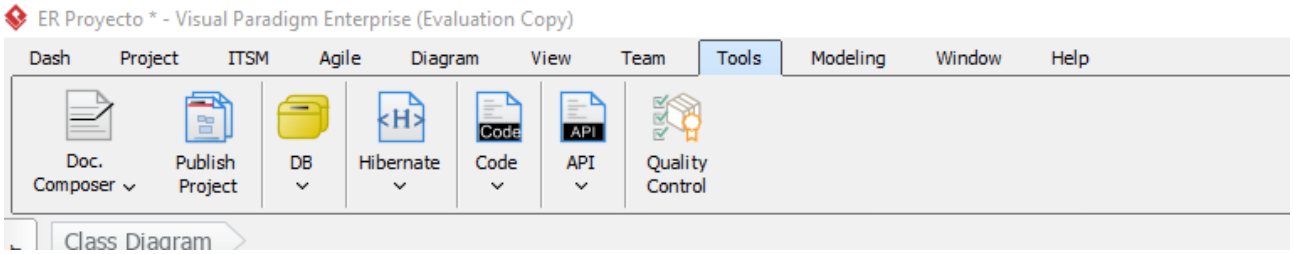


7. Descripción del funcionamiento del sistema y especificaciones técnicas para la aplicación del servidor y las APPs móvil y Web. Usabilidad

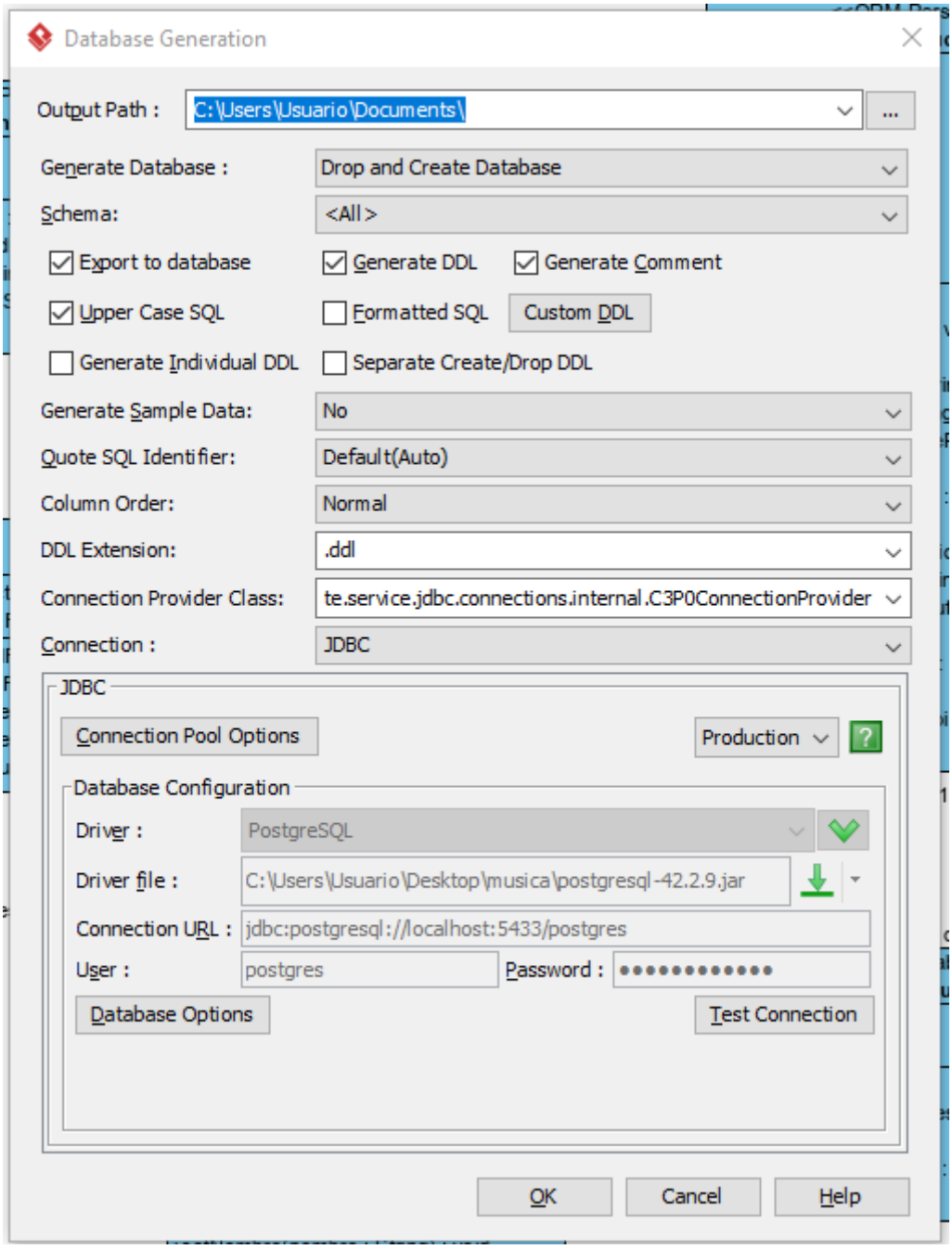
El sistema diseñado es para que hagan uso de ella los camareros del restaurante en la que se use la App. Podrán logearse o incluirse en la base de datos. Una vez logeados, podrán crear una nueva factura con los datos del cliente y la mesa en la que estan. Una vez creada les llevará a la página principal donde podrán ver los distintos tipos de productos: Entrantes, primer plato, segundo plato y postre. Además, podrán ver su carro en la parte de abajo de esta interfaz. Clickando en una de estas imagenes les llevará al menú DE ESE TIPO DE PRODUCTO, es decir, filtra por Entrante, primer plato, etc. Desde ahí tienen varias opciones:
-Ver la información de uno de los productos de la lista, llevándoles a una interfaz aparte
-Añadir productos al menú, llevándoles a otra interfaz aparte donde meter los datos
-Añadir productos al carro de ese pedido/factura
Una vez han finalizado los clientes con su pedido, bastaría con volver a la página principal y darle al botón de abajo para finalizar con ese pedido. Esto les llevará a la página de Login otra vez, reseteando la App y podiendo hacer otro pedido para otra mesa.

Para el uso:
Logearse o crearse una cuenta con la cual logearse.
Escribir los datos de la factura: datos del cliente, mesa en la que se encuentran, y numero del camarero que les atiende.
En el main, elegir el tipo de plato que va a querer el cliente escoger.
En el menú, podemos darle a info si queremos ver la información de un producto concreto; podemos agregar nuevos productos a este y podemos agregar productos al carro dandole al más.
Una vez tengamos el pedido entero, darle al botón para volver a Home y finalizar el pedido con el botón de abajo.

Para la instalación de la app:
Es simple. Diseñamos la base de datos, podemos hacer uso de ello con la app Visual Paradigm. Una vez hecho el diagrama ER y diagrama de clases (hacemos uso del archivo ER Proyecto) accedemos a la pestaña tools y en DB seleccionamos Generate database.



Rellenamos con los datos de nuestra base de datos Postgres (requiere instalar postgres previamente) y aceptamos. La base de datos está creada



Para poner en marcha el ORM, desde una terminal accedemos a la ruta de la carpeta ORM-Node y una vez en ella hacemos el comando **npm run start** para que la api empiece a funcionar.

Una vez está la api funcionando, accedemos a nuestra App. Las rutas ya están hechas por lo que no debería haber problemas de funcionamiento entre la api y la App.

Documentación Postman: https://documenter.getpostman.com/view/9084390/SWE9ZcqE?version=latest

Usabilidad

útil: Es capaz de hacer login, o de registrarse. Es capaz de crear pedidos. Es capaz de crear productos, añadirlos a un pedido.

Fácil de aprender: La aplicación es sencilla, no es difícil de aprender, y siempre tiene la posibilidad de volver atrás por si se da el caso de pérdida

El usuario es capaz de iniciar y controlar acciones. Haciendo clic en los variados botones o rellenando los formularios de las interfaces

El usuario es capaz de interacturar con la aplicación con el uso de los botones y los formularios que aparecen

Simplicidad del diseño la aplicación no está sobrecargada. La información que aparece es la justa y necesaria.

El diseño sigue pautas que conforman la interfaz de una manera clara y consistente.

El diseño de las interfaces intenta economizar los elementos y contenidos, para comunicar lo máximo con la mínima cantidad de elementos necesarios.

La aplicación es consistente. Las operaciones similares siempre se van a ver de manera similar.

La aplicación intenta ser lo más legible posible. La información intenta ser lo más fácil de ubicar y leer posible.

Hay mínima sorpresa. No va a haber nada que sorprenda al usuario

Los iconos son los adecuados. No va a existir confusión respecto a los iconos o a las imágenes escogidas y lo que ocurrirá en ese caso.