Cheffer-Documentation

Florencia Cartoni
Bartolomé Peirano
Magdalena Valdés

I. Introducción: Cheffer es una nueva red social, en donde se abre un nuevo espacio para que sus usuarios puedan compartir todas sus recetas al mundo! Para así todas las personas tengan la oportunidad de cocinar nuevos y ricos platos desde su hogar.

II. Modelo de Datos:
En el siguiente siguiente diagrama se puede apreciar el modelo entidad-relación de la aplicación:

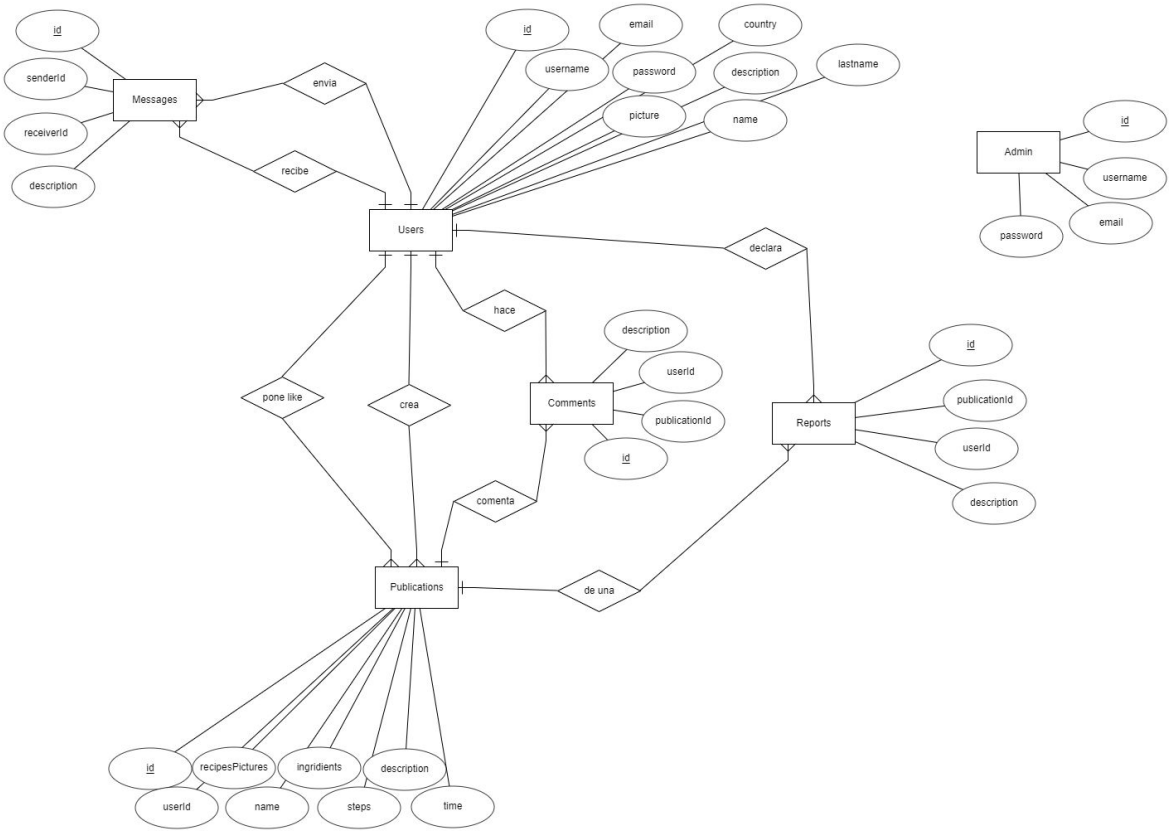

Imagen 1 : Diagrama Entidad-Relación Cheffer

Cabe destacar que en el diagrama falta la relación “follow” que es de N-N entre User con User, esto se debe a que el programa no permite agregarle más relaciones a User.

Por cada entidad del modelo, se creó una tabla en la base de datos. Además por cada relación N-N se creó otra tabla intermedia. En detalle:

Los modelos que se crearon son:

- **Users:** Con los siguientes atributos:
  - id (Integer)
  - name (String)
  - lastname (String)
  - username (String)
  - email (String)
  - password (String)
  - picture (String)
  - country (String)
  - description (String)

- **Publications:** Con los siguientes atributos:
  - id (Integer)
  - name(String)
  - recipesPictures (String)
  - ingredients (String)
  - time (Float)
  - userId (Integer)
  - steps (String)
  - description (String)

- **Messages:** Con los siguientes atributos:
  - id (Integer)
  - senderId (Integer)
  - receiverId (Integer)
  - description (String)

- **Comments:** Con los siguientes atributos:
  - id (Integer)
  - description (String)
  - userId (Integer)
  - publicationId (Integer)

- **Reports:** Con los siguientes atributos:
  - id (Integer)
  - description (String)
  - userId (Integer)
  - publicationId (Integer)

- **Admins:** Con los siguientes atributos:
  - id (Integer)
  - username (String)
  - email (String)
  - password (String)

*Cada modelo viene con una migración a la base de datos que crea su tabla respectiva.

Las Asociaciones que se crearon:

- **User-User:** Es una asociación N-N, la cual utiliza la tabla intermedia “"user_followers", para así modelar el proceso de follows.

- **User-Publication:** Aquí se tienen 3 asociaciones. La primera es 1-N, la cual se utiliza para asociar al usuario con las publicaciones que crea. La segunda es N-N, la cual utiliza la tabla intermedia “likes", para así modelar el proceso de likes. La tercera es N-N, la cual utiliza la tabla intermedia “saved_publications", para así modelar el proceso de guardar publicaciones.

- **User-Report:** Es una asociación 1-N, la cual se utiliza para asociar al usuario con los reports que crea.

- **User-Comment:** Es una asociación 1-N, la cual se utiliza para asociar al usuario con los comentarios que crea.

- **User-Message:** Es una asociación 1-N, la cual se utiliza para asociar al usuario con los mensajes que envía y que recibe.

- **Publication-Report:** Es una asociación 1-N, la cual se utiliza para asociar una publicación con la denuncia que se le hace.

- **Publication-Comment** Es una asociación 1-N, la cual se utiliza para asociar una publicación con los comentarios que se le hacen.

III. Estructura de la App:

Arquitectura de software: La aplicación web se basa en el estilo de arquitectura de software Modelo Vista Controlador (MVC), en donde básicamente se crean modelos a través de sequelize, los cuales son controlados por los routers y que también permiten manejar las vistas html.

Para inicializar el proyecto se utilizó un template basado en Koa Js, para más información pueden revisar la documentación en el siguiente link https://koajs.com/

Para crear modelos y migraciones se utilizó el ORM Sequelize, para más información pueden revisar su documentación en https://sequelize.org/master/

Carpetas y archivos: El principal directorio con el que se trabaja en la aplicación es “Src”, dentro de él se encuentran varios subdirectorios y archivos en donde se construye el proyecto. Los principales directorios y archivos a trabajar quedan representados y explicados de mejor manera con los siguientes esquemas:

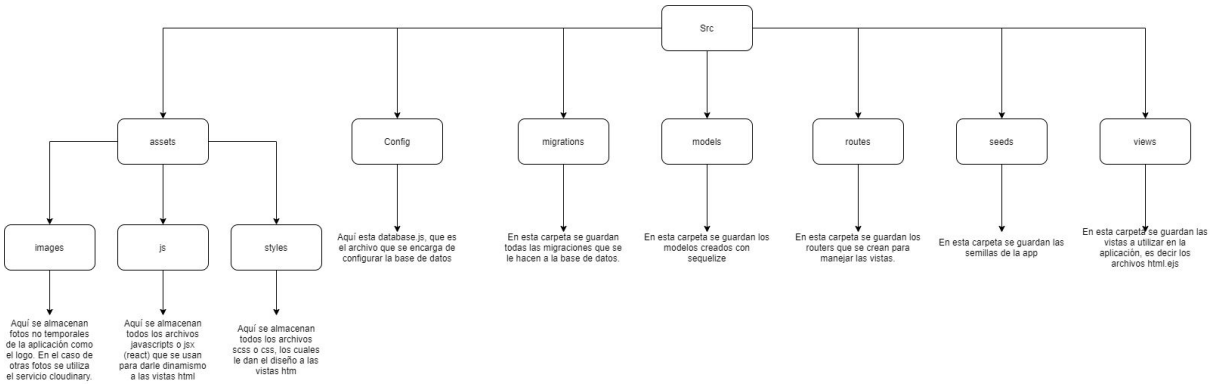

Imagen 2 : Diagrama sobre los principales subdirectorios de src.

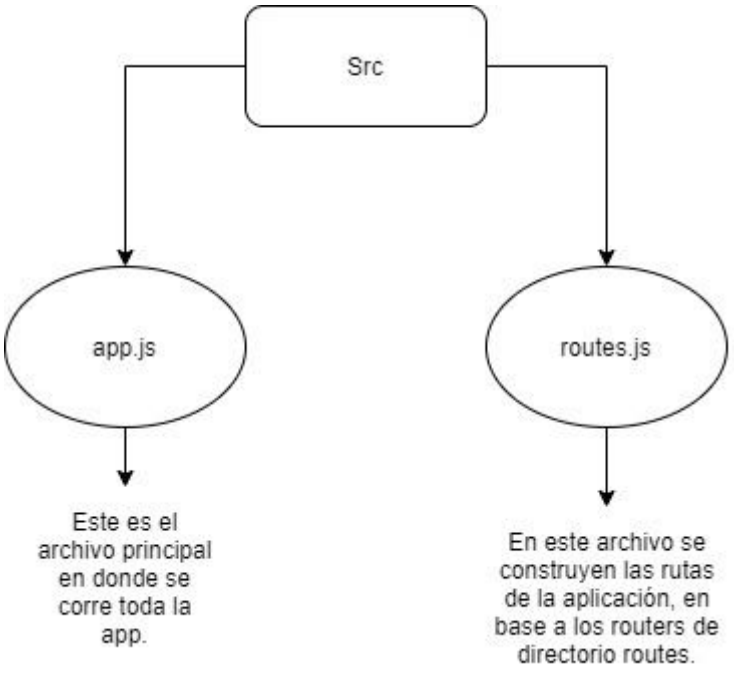

Imagen 3 : Diagrama sobre los principales archivos de src.

Se recomienda revisar bien las explicaciones de cada uno de los diagramas. Para tener una mejor visión del flujo de la app.

Frontend: Respecto al Frontend, este se trabaja a través de las vistas html.ejs dentro del directorio src/views. Estas vistas obtienen su diseño a partir del css de los archivos que se encuentra dentro de src/assets/styles. Además, estas vistas obtienen dinamismo a través del javascript y jsx del framework react de los archivos que se encuentran dentro de src/assets/js. Un ejemplo de Javascript sería el dark mode dado por el archivo darkmode.js. Un ejemplo de react sería el botón para darle like a las publicaciones, el cual está dado por LikeButton.jsx.

```
document.addEventListener('DOMContentLoaded', function() {

    console.log("aqui en el javascript bart")
    document.addEventListener("keypress", function(e){
        e = e || window.event;
        console.log("aqui en el eventoo bart")


        if (e.key === 'Enter' ){
            document.documentElement.classList.toggle('dark-mode');
        };

    })
})
```

Imagen 4: Ejemplo de javascript en el frontend de src/js/darkmode.js

```
.like-button{
    height: 20px;
    width: 25px;
    background-color: white;
    border-color: white;
    color: white;
    text-decoration: none;
    cursor: pointer;
}
```

Imagen 5: Ejemplo de css en el frontend de src/js/LikeButtonc.css

Backend: El backend está hecho en base a puro javascript, cómo se mencionó antes, en base a un framework llamado koa.js. Para entender mejor esta parte se recomienda revisar src/routes.js , src/app.js y src/routes/<cualquier router o archivo js>. El funcionamiento básico consiste en routers que trabajan en función de javascript, para construir las rutas de la aplicación y para pasarle objetos o información a las vistas. Cabe destacar que la aplicación se monta en base a una secuencia de middlewares.

```
router.get("publications.index","/", loadUser, loadPublication, async (ctx) => {
    const publications = await ctx.orm.publication.findAll();
    const { user, publication } = ctx.state;
    await ctx.render("publications/index", {
        user,
        publication,
        publications,
        newPublicationPath: ctx.router.url("publications.new", {userId: user.id}),
        publicationPath: (publication) => ctx.router.url("publications.show", {id: publication.id, userId: user.id}),
    })
});
```

Imagen 4: Ejemplo de uno de los routers del tipo get de src/routes/publications.js

Servicios externos: Se utilizaron dos servicios externos en la aplicación. En primer lugar se utilizó Cloudinary para almacenar las fotos de usuarios y de recetas subidas.En segundo lugar se utilizó el mailer Sendgrid, para poder enviar emails a los usuarios cuando se registran o para cuando envían un mensaje. Para más información pueden revisar su documentación respectiva.
Cloudinary: https://cloudinary.com/documentation
Mailer: https://sendgrid.com/docs/

Deploy: Para el deploy de la aplicación web, se utilizó heroku. Se puede visitar utilizando el siguiente link https://cheffer-3mosqueteros.herokuapp.com/ .
Para la base de datos se utilizó el resource Heroku Postgres.
También se utilizaron las siguientes variables para configurar el deploy:

- DATABASE_URL
- CLOUDINARY_URL
- HASH_SECRET
- JWT_SECRET
- SENDGRID_PASS
- SENDGRID_USER

IV. API consumida:

La API elegida es de esta página:
https://rapidapi.com/spoonacular/api/recipe-food-nutrition/pricing
Funciona de la siguiente forma: se hace un request de tipo GET, el cual recibe el nombre de la receta por la cual se está preguntando y como respuesta muestra las calorías que contiene.

V. API expuesta: para exponer la API se crearon 4 rutas, estas pueden ser probadas desde postman y desde la segunda aplicación

1. **Desde postman:** Creamos una carpeta llamada Cheffer que contiene 4 carpetas
- auth: tiene el request New auth de tipo post, se accede a él con la ruta: localhost:3000/api/auth
  Para probarlo, se debe mandar un body en formato json de la forma:

```
{
    "email": "mvaldes12@uc.cl",
    "password": "123"
}
```

  - Si el mail y la contraseña son correctos, devuelve el token de la siguiente forma:

```
{
    "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOjEsImlhdCI6MTYwNjc1Mjk3
NX0.QUiHpcSGQXq8iJVRaPkn9uxZRk8Uqvux5cLCCShJEuw"
```

```
}
```

- Si el mail o la contraseña son incorrectos, devuelve:

```
{
    "errorValidation": "Incorrect password or email"
}
```

- Si el body no tiene las llaves “email” y “password” (o están mal escritos), devuelve: (en este ejemplo solo se mandó la contraseña)

```
{
    "errorValidation": "WHERE parameter \"email\" has invalid
\"undefined\" value"
}
```

- users: tiene el request Get user data de tipo get, se accede a él con la ruta: localhost:3000/api/users/me
  Para probarlo, se debe mandar el token en la autentificación
  - Si el token es válido, devuelve:

```
{
    "id": 1,
    "name": "Magdalena",
    "lastname": "Valdes",
    "username": "mvaldes12",
    "email": "mvaldes12@uc.cl",
    "password":
"$2b$10$EOfb9im8Tw3B6M1XHSOfqeBtznDzKk5/AaQUrZ3x9N4Aax5GGTarG",
    "picture": "",
    "country": "Chile",
    "description": "",
    "savedRecipes": null,
    "followers": null,
    "blockedUsers": null,
    "createdAt": "2020-11-17T14:59:02.668Z",
    "updatedAt": "2020-11-17T14:59:02.668Z"
}
```

  - Si el token es inválido, devuelve: “Authentication Error”

- publications:
  1. Tiene el request List publications de tipo get, se accede a él con la ruta: localhost:3000/api/publications
     Para probarlo, se debe mandar el token en la autentificación
     - Si el token es válido, devuelve:

```
[
    {
        "id": 3,
        "name": "tortaaaaa",
        "recipesPictures": null,
        "ingredients": "manjar",
        "recipesVideos": null,
```

```
        "time": 1,
        "userId": 1,
        "steps": "revolver",
        "ranking": null,
        "stepsPictures": null,
        "description": "torta manjar",
        "createdAt": "2020-11-25T16:55:34.020Z",
        "updatedAt": "2020-11-25T16:55:34.020Z",
        "publicationUrl":
"http://localhost:3000/api/publications/3"
    },
    {
        "id": 4,
        "name": "pie de limon",
        "recipesPictures": null,
        "ingredients": "limon",
        "recipesVideos": null,
        "time": 1,
        "userId": 1,
        "steps": "hornear",
        "ranking": null,
        "stepsPictures": null,
        "description": "muy rico",
        "createdAt": "2020-11-25T16:55:54.589Z",
        "updatedAt": "2020-11-25T16:55:54.589Z",
        "publicationUrl":
"http://localhost:3000/api/publications/4"
    },
    {
]
```


- Si el token es inválido, devuelve: “Authentication Error”

2. Tiene el request Get publication de tipo get, se accede a él con la ruta: localhost:3000/api/publications/id
   Para probarlo, se debe mandar el token en la autentificación
   - Si el token es válido y el id existe, devuelve:


```
{
    "id": 3,
    "name": "tortaaaaa",
    "recipesPictures": null,
    "ingredients": "manjar",
    "recipesVideos": null,
    "time": 1,
    "userId": 1,
    "steps": "revolver",
    "ranking": null,
```

```
    "stepsPictures": null,
    "description": "torta manjar",
    "createdAt": "2020-11-25T16:55:34.020Z",
    "updatedAt": "2020-11-25T16:55:34.020Z",
    "publicationByIdUrl": "http://localhost:3000Error: No route
found for name: publication/:id"
}
```


- Si el token es válido y el id no existe, devuelve:


```
{
    "errorNotFound": "This id does not exist"
}
```


- Si el token es inválido, devuelve: “Authentication Error”

3. Tiene el request Create publication de tipo post, se accede a él con la ruta: localhost:3000/api/publication/new
   Para probarlo, se debe mandar el token en la autentificación y un body en formato json de la forma:


```
{
    "name": "tacos",
    "ingredients": "masa",
    "steps": "hacer la carne",
    "time": 3,
    "description": "muy buenooooo"
}
```


   - Si el body tiene las llaves “name” “ingredients”, “steps” y “time” , se crea correctamente la publicación y devuelve:


```
{
    "id": 11,
    "name": "tacos",
    "ingredients": "masa",
    "steps": "hacer la carne",
    "time": 3,
    "description": "muy buenooooo",
    "userId": 1,
    "updatedAt": "2020-11-30T18:07:13.469Z",
    "createdAt": "2020-11-30T18:07:13.469Z",
    "recipesPictures": null,
    "recipesVideos": null,
    "ranking": null,
    "stepsPictures": null
}
```

- Si el body no tiene las llaves “name” “ingredients”, “steps” y “time” (o están mal escritos), devuelve: (en este ejemplo solo no se mandó la el nombre)

```
{
    "errorValidation": "notNull Violation: publication.name
cannot be null"
}
```

- Si el token es inválido, devuelve: “Authentication Error”

4. Tiene el request List comments de tipo get, se accede a él con la ruta: localhost:3000/api/publications/id/comments
   Para probarlo, se debe mandar el token en la autentificación
   - Si el token es válido y el id existe, devuelve:

```
[
    {
        "id": 1,
        "description": "muy buen pie de limón!!!!",
        "userId": 1,
        "publicationId": 4,
        "createdAt": "2020-11-25T16:59:26.819Z",
        "updatedAt": "2020-11-25T16:59:26.819Z",
        "commentUrl": "http://localhost:3000Error: No route
found for name: publication/:id/comments"
    },
    {
        "id": 2,
        "description": "Me encanto tu pie",
        "userId": 1,
        "publicationId": 4,
        "createdAt": "2020-11-25T19:44:36.319Z",
        "updatedAt": "2020-11-25T19:44:36.319Z",
        "commentUrl": "http://localhost:3000Error: No route
found for name: publication/:id/comments"
    }
]
```

   - Si el token es válido y el id no existe, devuelve:

```
{
    "errorNotFound": "This id does not exist"
}
```

   - Si el token es inválido, devuelve: “Authentication Error”

5. Tiene el request Get comment de tipo get, se accede a él con la ruta: localhost:3000/api/publications/comments/id
   Para probarlo, se debe mandar el token en la autentificación
   - Si el token es válido y el id existe, devuelve:

```
{
    "id": 1,
    "description": "muy buen pie de limón!!!!",
    "userId": 1,
    "publicationId": 4,
    "createdAt": "2020-11-25T16:59:26.819Z",
    "updatedAt": "2020-11-25T16:59:26.819Z",
    "commentUrl": "http://localhost:3000Error: No route found for name: publication/:id/comments"
}
```

   - Si el token es válido y el id no existe, devuelve:

```
{
    "errorNotFound": "This id does not exist"
}
```

   - Si el token es inválido, devuelve: “Authentication Error”

6. Tiene el request Create comment de tipo post, se accede a él con la ruta: localhost:3000/api/publications/id/comments/new
   Para probarlo, se debe mandar el token en la autentificación y un body en formato json de la forma:

```
{
    "description": "que buena receta!"
}
```

   - Si el body tiene la llave “description” , se crea correctamente el comentario y devuelve:

```
{
    "id": 12,
    "description": "mvaldes12@uc.cl",
    "userId": 1,
    "publicationId": 3,
    "updatedAt": "2020-11-30T18:20:13.715Z",
    "createdAt": "2020-11-30T18:20:13.715Z"
}
```

   - Si el body no tiene la llave “description” (o está mal escrito), devuelve:

```
{
    "errorValidation": "notNull Violation: comment.description cannot be null"
}
```

- Si el token es inválido, devuelve: “Authentication Error”

2. **Desde la segunda aplicación:** se pueden probar todos los request en este link: https://app2-cheffer.herokuapp.com/users/signin, donde se utilizó Axios para hacer los request.

VI. Proyecciones: Creemos que cheffer ha llegado a ser una aplicación web de gran calidad, pero que igual se puede seguir desarrollando. Áreas interesantes para seguir avanzando puede ser la creación de una aplicación móvil en base a react native. O crear más funcionalidades en el frontend con React. O implementar un sistema para poder comprar platos ya hechos dentro de la aplicación. También, se puede seguir avanzando en perfeccionar el css de la página.