

COOKPEDIA

Ezequiel Ares Rodríguez



I.E.S Rafael Alberti
Desarrollo de Aplicaciones Web

Contenido

Introducción.....	3
Descripción.....	4
Receta.....	4
Usuario.....	8
Like.....	9
Comentario.....	10
Ingrediente.....	11
Instalación.....	13
Guía de estilos.....	17
Nombre de la empresa.....	17
Eslogan.....	17
Colores.....	17
Tipografía.....	18
Iconografía.....	18
Prototipado.....	19
WireFrame - Móvil.....	19
Mockup – Móvil.....	20
Mockup – Móvil noche.....	21
Wireframe – Desktop.....	22
Mockup – Desktop.....	24
Mockup – Desktop Modo Noche.....	26
Diseño.....	28
Entidad relación.....	28
UML.....	29
Desarrollo.....	30
Secuencia de desarrollo y dificultades encontradas.....	30
Herramientas de control de versiones y revisión del código.....	31
Pruebas.....	32
Despliegue.....	32
Descripción del proceso.....	32
Manual.....	35
Header.....	35
Home.....	35
Recipes.....	36

Log in / Register	37
Actualizar usuario	38
Liked	38
Upload recipe	39
Receta.....	40
Conclusiones.....	41
Índice de tablas e imágenes	42
Bibliografía.....	44

Introducción

Las expectativas que tengo con este proyecto es poder crear por primera vez una aplicación sólida en cuanto a back y diseño por mi cuenta que ofrezca diferentes elementos como recetas, comentar en estas, guardar recetas en una lista para verlas luego, poder buscar rectas con diferentes filtros, etc...

La idea de hacer un proyecto sobre recetas viene desde hace un tiempo ya que por gusto personal me encantaría saber cocinar y tener a mi disposición alguna web en la que se expliquen cientos de recetas y que tenga cierto nivel en cuanto a diseño sería una buena opción para aprender, tenía dudas sobre si llevar a cabo esta idea pero cuando tuvimos una reunión con los compañeros de administrativo y uno de ellos propuso una idea bastante parecida a lo que me planteé en su día por lo que decidí realizar el proyecto con el objetivo de la página de recetas.

En cuanto a tecnologías usadas en la aplicación he utilizado para el back Flask, el cual es un framework de Python que habíamos usado anteriormente para realizar una API de un juego y me convenció para este proyecto ya que también iba a usar una API y es bastante sencillo de utilizar, dentro de Flask tenemos una serie de requisitos para ejecutarlo los cuales se detallan en el archivo requirements.txt pero no nos tendremos que preocupar por esto ya que mediante el Docker-compose que he creado se ejecuta automáticamente la instalación de estos con el comando `pip install`. Sobre el front he usado React, al igual que Flask lo habíamos usado recientemente en clase y me parecía la mejor opción a la hora de realizar el front, dentro de React he usado FontAwesome el cual nos permite introducir ciertos iconos a la hora de renderizar el HTML, por lo demás he usado diferentes librerías como `navigate` para cambiar de página desde el código al terminar una función o el mismo `routes` el cual nos permite realizar un enrutador para los componentes.

Sobre aplicaciones similares a esta no he realizado ninguna con la temática de cocina, pero si nos enfocamos en el back por ejemplo podemos ver que me baso mucho en la API del juego que he mencionado anteriormente o incluso el “Instagram” que hicimos con Laravel, el cual me ha ayudado a enfocar las relaciones de las entidades y algunos métodos. Sobre el front lo más parecido que he realizado fue el mini proyecto que hicimos de diseño junto a desarrollo cliente, en el cual creé una página sobre música, pero con un diseño bastante simple y centrado en destacar elementos, esto me ha ayudado a realizar el diseño del proyecto ya que, aunque las temáticas sean diferentes también se basa en las mismas ideas en cuanto a diseño de ser simple y destacar ciertos elementos.

Descripción

En cuanto al resultado obtenido estoy bastante satisfecho en cuanto a funcionalidades ya que excepto uno o dos métodos o funcionalidades que no he podido implementar por falta de tiempo o planteamiento tales como filtrar búsquedas por tags (campo el cual he acabado eliminando de la entidad receta por no poder realizar el filtro), búsqueda por ingredientes u modificación de recetas. Vamos a ir detallando las funcionalidades que sí he podido implementar (omitiré el CRUD de cada entidad para no dar más información de la necesaria al ser algo muy sencillo, aunque mencionaré si hay algo que destacar en algún CRUD):

Receta

```
def get(self, recipe_id):
    query = sqlalchemy.text('SELECT r.id, r.nombre, r.descripcion, r.imagen, r.video, r.pasos, r.id_usuario as id_usuario, count(l.id) '
                             'as likes, u.nombre as nombreUsuario FROM receta r, like l, usuario u '
                             'WHERE r.id = :recipe_idRequest AND r.id = l.receta_id AND r.id_usuario = u.id GROUP BY r.id;')

    result = db.session.execute(query, {"recipe_idRequest": recipe_id})
    resultMapping = result.mappings().all()
    r = resultMapping[0]

    if (r["video"] != None):
        if (r["pasos"] != None):
            recipe = {"id": r["id"], "nombre": r["nombre"], "imagen": r["imagen"], "likes": r["likes"], "nombreUsuario":
                      r["nombreUsuario"], "id_usuario": r["id_usuario"], "descripcion": r["descripcion"],
                      "video": r["video"], "pasos": r["pasos"]}
        else:
            recipe = {"id": r["id"], "nombre": r["nombre"], "imagen": r["imagen"], "likes": r["likes"], "nombreUsuario":
                      r["nombreUsuario"], "id_usuario": r["id_usuario"], "descripcion": r["descripcion"],
                      "video": r["video"]}
    else:
        if (r["pasos"] != None):
            recipe = {"id": r["id"], "nombre": r["nombre"], "imagen": r["imagen"], "likes": r["likes"], "nombreUsuario":
                      r["nombreUsuario"], "id_usuario": r["id_usuario"], "descripcion": r["descripcion"],
                      "pasos": r["pasos"]}
        else:
            recipe = {"id": r["id"], "nombre": r["nombre"], "imagen": r["imagen"], "likes": r["likes"], "nombreUsuario":
                      r["nombreUsuario"], "id_usuario": r["id_usuario"], "descripcion": r["descripcion"]}

    return recipe
```

Descripción 1

Get – Tuve que modificar el obtener una receta ya que necesito obtener el número de likes y el nombre del usuario que las creó en el mismo json (esta sentencia se repetirá en cualquier método que implique devolver una receta), luego tenemos una serie de “if” los cuales se encargan de comprobar si la receta tiene ciertos campos y crear el json dependiendo de estos, luego devuelve el json.

```
@flask_praetorian.auth_required
def post(self):
    data = request.values
    imagen = request.files['imagen']

    carpeta = current_app.root_path
    imagen.save(carpeta + "/static/recetas/" + imagen.filename)
    imagen_new_user = "http://localhost:5000/static/recetas/" + imagen.filename

    try:
        video = request.files['video']
        video.save(carpeta + "/static/recetas/" + video.filename)
        video_new_user = "http://localhost:5000/static/recetas/" + video.filename

        if (data["pasos"] != ""):
            new_recipe = {"nombre": data["nombre"], "descripcion": data["descripcion"], "imagen": imagen_new_user,
                          "pasos": data["pasos"], "video": video_new_user, "usuario": data["id_usuario"]}

        elif (data["pasos"] == ""):
            new_recipe = {"nombre": data["nombre"], "descripcion": data["descripcion"], "imagen": imagen_new_user,
                          "video": video_new_user, "usuario": data["id_usuario"]}

    except KeyError:
        new_recipe = {"nombre": data["nombre"], "descripcion": data["descripcion"], "imagen": imagen_new_user,
                      "pasos": data["pasos"], "usuario": data["id_usuario"]}

    receta = RecetaSchema().load(new_recipe)
    db.session.add(receta)
```

Descripción 2

Post – Para crear la receta dividiremos el código en dos partes al ser demasiado largo, la primera se encarga de guardar la imagen de la receta en el almacenamiento local y luego comprobamos si tiene video, si tiene lo guardamos y comprobamos si tiene además el campo de pasos, si no disponemos de video pasamos a crear el json y lo cargamos y guardamos en la base de datos.

```
try:
    ingredientes = json.loads(data["ingredientes"])
    listaIngredientes = []
    listaIngredientesReceta = []

    for r in ingredientes:
        listaIngredientes.append({'ingrediente': Ingrediente.query.filter(Ingrediente.id.in_(r["ingrediente_id"])).all()[0], "cantidad": r["cantidad"]})

    for ingrediente in listaIngredientes:
        listaIngredientesReceta.append(IngredienteReceta(receta_id=receta.id, ingrediente_id=ingrediente["ingrediente"].id,
                                                         cantidad=ingrediente["cantidad"]))

    like = Like(usuario_id=data["id_usuario"], receta_id=receta.id)

    db.session.bulk_save_objects(listaIngredientesReceta)
    db.session.add(like)

except KeyError:
    print("Sin ingredientes")
db.session.commit()

return RecetaSchema().dump(receta), 201
```

Descripción 3

En la siguiente parte del código obtenemos la lista de ingredientes de la receta que vamos a crear, luego recorremos esta lista introduciendo en una nueva el objeto ingrediente de cada ingrediente de la lista y una vez terminado recorremos la nueva lista para ir creando las relaciones IngredienteReceta. Luego creamos también el like a nuestra propia receta (es necesario para que funcionen los demás métodos de la aplicación) y los añadimos a la base de datos.

```
@api_receta.route("/list")
class RecetaListController(Resource):

    def get(self):
        order = request.args.get('order')

        if (order == "date"):
            query = sqlalchemy.text('SELECT r.id, r.nombre as nombre, r.imagen, count(l.id) as likis, u.nombre as '
                                   'nombreUsuario FROM receta r, like l, usuario u WHERE r.id = l.receta_id '
                                   'AND u.id = r.id_usuario group by r.id order by r.id desc')

            result = db.session.execute(query)
            resultMapping = result.mappings().all()

            recipes = {r["id"]: [
                {"id": r["id"], "nombre": r["nombre"], "imagen": r["imagen"], "likes": r["likis"], "nombreUsuario":
                 r["nombreUsuario"]} for r in resultMapping]

        elif (order == "likes"):
            query = sqlalchemy.text('SELECT r.id, r.nombre as nombre, r.imagen, count(l.id) as likis, u.nombre as '
                                   'nombreUsuario FROM receta r, like l, usuario u WHERE r.id = l.receta_id '
                                   'AND u.id = r.id_usuario group by r.id order by likis desc')

            result = db.session.execute(query)
            resultMapping = result.mappings().all()

            recipes = {r["nombre"]: [
                {"id": r["id"], "nombre": r["nombre"], "imagen": r["imagen"], "likes": r["likis"], "nombreUsuario":
                 r["nombreUsuario"]} for r in resultMapping]

        return recipes
```

Descripción 4

List – Con este método devolvemos todas las recetas de la base de datos filtrados por fecha de creación o número de likes.

```
@api_receta.route("/count")
class RecetaListController(Resource):

    def get(self):
        count = Receta.query.count()
        return count
```

Descripción 5

Count - Con este método obtenemos la cuenta de todas las recetas de la base de datos, lo usamos para obtener una receta aleatoria entre las que hay creadas desde el front.

```
@api_receta.route("/search/<string:name>")
class RecetaController(Resource):
    def get(self, name):
        order = request.args.get('order')
        name = '%' + name + '%'

        if (order == "date"):
            query = sqlalchemy.text('SELECT r.id, r.nombre as nombre, r.imagen, count(l.id) as likes, u.nombre as '
                                   'nombreUsuario FROM receta r, like l, usuario u WHERE r.id = l.receta_id ' +
                                   'AND u.id = r.id_usuario AND r.nombre LIKE :nameRequest group by r.id order by r.id desc')
            result = db.session.execute(query, {"nameRequest": name})
            resultMapping = result.mappings().all()

            recipes = [{"id": r["id"], "nombre": r["nombre"], "imagen": r["imagen"], "likes": r["likes"], "nombreUsuario":
                        r["nombreUsuario"]} for r in resultMapping]

        elif (order == "likes"):
            query = sqlalchemy.text('SELECT r.id, r.nombre as nombre, r.imagen, count(l.id) as likes, u.nombre as '
                                   'nombreUsuario FROM receta r, like l, usuario u WHERE r.id = l.receta_id ' +
                                   'AND u.id = r.id_usuario AND r.nombre LIKE :nameRequest group by r.id order by likes desc')

            result = db.session.execute(query, {"nameRequest": name})
            resultMapping = result.mappings().all()

            recipes = [{"id": r["id"], "nombre": r["nombre"], "imagen": r["imagen"], "likes": r["likes"], "nombreUsuario":
                        r["nombreUsuario"]} for r in resultMapping]

        return recipes
```

Descripción 6

Search – Con este método obtenemos las recetas filtrando por una búsqueda realizada por del usuario y ordenando mediante fecha de creación o número de likes.

```
@api_receta.route("/home")
class RecetaController(Resource):
    def get(self):
        count = Receta.query.count()

        query = sqlalchemy.text('SELECT r.id, r.nombre as nombre, r.imagen, count(l.id) as likes, u.nombre as nombreUsuario FROM receta r, like l, ' +
                                'usuario u WHERE r.id = l.receta_id AND u.id = r.id_usuario AND r.id BETWEEN (:count-5) AND :count ' +
                                'group by r.id order by r.id desc')

        result = db.session.execute(query, {"count": count})
        resultMapping = result.mappings().all()

        return [{"id": r["id"], "nombre": r["nombre"], "imagen": r["imagen"], "likes": r["likes"], "nombreUsuario":
                r["nombreUsuario"]} for r in resultMapping]
```

Descripción 7

Home – Este método nos devuelve las 6 últimas recetas creadas por los diferentes usuarios.

Usuario

```
@flask_praetorian.auth_required
def put(self, user_id):
    data = request.values

    if (data['imagen'] != ""):
        if (data['hashed_password'] != ""):
            new_user = {"id": user_id, "nombre": data["nombre"], "nick": data["nick"], "email": data["email"],
                        "hashed_password": data["hashed_password"], "imagen": data["imagen"]}
        else:
            new_user = {"id": user_id, "nombre": data["nombre"], "nick": data["nick"], "email": data["email"],
                        "imagen": data["imagen"]}

    elif (data['imagen'] == ""):
        imagen = request.files['nuevaImagen']

        carpeta = current_app.root_path
        imagen.save(carpeta + "/static/usuarios/" + imagen.filename)

        imagen_new_user = "http://localhost:5000/static/usuarios/" + imagen.filename

        if (data['hashed_password'] != ""):
            new_user = {"id": user_id, "nombre": data["nombre"], "nick": data["nick"], "email": data["email"],
                        "hashed_password": data["hashed_password"], "imagen": imagen_new_user}
        else:
            new_user = {"id": user_id, "nombre": data["nombre"], "nick": data["nick"], "email": data["email"],
                        "imagen": imagen_new_user}

    new_user = UsuarioSchema().load(new_user)

    if str(new_user.id) != user_id:
        abort(400, "no coincide el id")

    if (data["hashed_password"] != ""):
        guard = flask_praetorian.Praetorian()
        guard.init_app(current_app, Usuario)
        new_user.hashed_password = guard.hash_password(new_user.hashed_password)

    db.session.commit()
    return UsuarioSchema().dump(new_user)
```

Descripción 8

Put – Para modificar el usuario he tenido que cambiar un poco el método Put del CRUD, para empezar, comprobamos si vamos a cambiar la imagen, si es así la guardamos y creamos el json para el nuevo usuario, lo cargamos para tener el objeto usuario de la base de datos y comprobamos que el id es correcto, si vamos a cambiar también la contraseña del usuario la registramos en Praetorian y guardamos en el campo contraseña la contraseña hasheada, luego guardamos el usuario.

```
def post(self):
    data = request.values

    try:
        imagen = request.files['imagen']

        carpeta = current_app.root_path
        imagen.save(carpeta + "/static/usuarios/" + imagen.filename)

        imagen_new_user = "http://localhost:5000/static/usuarios/" + imagen.filename

        new_user = {"nombre": data["nombre"], "nick": data["nick"], "email": data["email"],
                    "hashed_password": data["hashed_password"], "imagen": imagen_new_user}
    except KeyError:
        imagen = "http://localhost:5000/static/usuarios/anon.jpg"

        new_user = {"nombre": data["nombre"], "nick": data["nick"], "email": data["email"],
                    "hashed_password": data["hashed_password"], "imagen": imagen}

    user = UsuarioSchema().load(new_user)

    guard = flask_praetorian.Praetorian()
    guard.init_app(current_app, Usuario)
    user.hashed_password = guard.hash_password(user.hashed_password)

    db.session.add(user)
    db.session.commit()

    return UsuarioSchema().dump(user), 201
```

Descripción 9

Post – Para crear el usuario he tenido que controlar si vamos a introducir una imagen o no, si la introducimos la guardamos en el almacenamiento local y si no le ponemos una por defecto, hashamos la contraseña y guardamos el usuario.

Like

```
@api.route("/receta/<receta_id>")
class LikeController(Resource):
    def get(self, receta_id):
        query = sqlalchemy.text('SELECT count(l.id) as "likes" FROM like l JOIN receta r on l.receta_id = r.id WHERE r.id = :receta_idRequest')
        result = db.session.execute(query, {"receta_idRequest": receta_id})
        resultMapping = result.mappings().all()
        return {"likes": resultMapping[0]["likes"]}
```

Descripción 10

/receta/<receta_id> - Con este método devolvemos el número de likes que tiene una receta.

```
@api.route("/recetas/<user_id>")
class LikeController(Resource):
    def get(self, user_id):
        query = sqlalchemy.text(
            'SELECT r.id, r.nombre as nombre, r.imagen, count(l.id) as likis, u.nombre as nombreUsuario FROM receta r, '
            'like l, usuario u WHERE r.id IN (SELECT receta_id FROM like WHERE usuario_id = :user_idRequest) '
            'AND r.id = l.receta_id AND r.id_usuario = u.id AND r.id_usuario != :user_idRequest group by r.id')

        result = db.session.execute(query, {"user_idRequest": user_id})
        resultMapping = result.mappings().all()

        return {"id": [
            {'id': r["id"], "nombre": r["nombre"], "imagen": r["imagen"], "likes": r["likis"], "nombreUsuario":
            r["nombreUsuario"]} for r in resultMapping]
```

Descripción 11

/recetas/<user_id> - Con este método obtenemos todas las recetas a las que hemos dado like (excepto las que hemos creado nosotros mismos, ya que estas ya tienen like y no deberíamos mostrarlas)

```
@api.route("/tiene/<recipe_id>/<user_id>")
class LikeController(Resource):
    def get(self, recipe_id, user_id):
        query = sqlalchemy.text(
            'SELECT * FROM like WHERE receta_id = :recipe_idRequest AND usuario_id = :user_idRequest;')

        result = db.session.execute(query, {"recipe_idRequest": recipe_id, "user_idRequest": user_id})
        resultMapping = result.mappings().all()

        if (len(resultMapping) > 0):
            return {"result": True, "idLike": resultMapping[0].id}

        return {"result": False, "idLike": ""}
```

Descripción 12

/tiene/<recipe_id>/<user_id> - Con este método sabremos si el usuario tiene like en una receta específica y el id del like.

Comentario

```
@flask_praetorian.auth_required
def post(self):
    data = request.values

    if (data["imagen"] == ""):
        new_comment = {'usuario': data['usuario_id'], 'receta': data['receta_id'],
                       'contenido': data['contenido'], 'imagen': None}

    elif (data['imagen'] != ""):
        imagen = request.files['imagenFile']

        carpeta = current_app.root_path
        imagen.save(carpeta + "/static/comentarios/" + imagen.filename)

        imagen_new_comment = "http://localhost:5000/static/comentarios/" + imagen.filename

        new_comment = {'usuario': data['usuario_id'], 'receta': data['receta_id'],
                       'contenido': data['contenido'], 'imagen': imagen_new_comment}

    if (data['padre_id'] != ""):
        new_comment['padre'] = data['padre_id']
    elif (data['padre_id'] == ""):
        new_comment['padre'] = None

    comment = ComentarioSchema().load(new_comment)

    db.session.add(comment)
    db.session.commit()

    return ComentarioSchema().dump(comment), 201
```

Descripción 13

Post – He tenido que modificar este método para, dependiendo de los atributos que lleguen desde el request, se cree un json específico y guarde la imagen en el almacenamiento local.

```
@api_comentario.route("/padre/<receta_id>")
class ComentarioController(Resource):
    def get(self, receta_id):
        query = sqlalchemy.text('SELECT c.*, u.imagen as imagenUsuario, u.nombre as nombreUsuario FROM comentario c, usuario u JOIN '
                                'receta r on c.receta_id = r.id WHERE r.id = :receta_idRequest AND c.padre_id IS NULL '
                                'AND u.id = c.usuario_id ORDER BY c.id desc')
        result = db.session.execute(query, {"receta_idRequest": receta_id})
        resultMapping = result.mappings().all()

        return [{"id": r["id"], "usuario_id": r["usuario_id"], "receta_id": r["receta_id"], "imagenUsuario": r["imagenUsuario"],
                "nombreUsuario": r["nombreUsuario"], "imagen": r["imagen"], "contenido": r["contenido"]}
                for r in resultMapping]
```

Descripción 14

/padre/<receta_id> - Con este método obtenemos todos los comentarios principales (que no son hijos de otros) de una receta específica.

```
@api_comentario.route("/hijo/<receta_id>")
class ComentarioController(Resource):
    def get(self, receta_id):
        query = sqlalchemy.text('SELECT c.*, u.imagen as imagenUsuario, u.nombre as nombreUsuario FROM comentario c, usuario u JOIN receta '
                                'r on c.receta_id = r.id WHERE r.id = :receta_idRequest AND c.padre_id IS NOT NULL AND u.id = c.usuario_id '
                                'ORDER BY c.id desc')
        result = db.session.execute(query, {"receta_idRequest": receta_id})
        resultMapping = result.mappings().all()

        return [{"id": r["id"], "usuario_id": r["usuario_id"], "receta_id": r["receta_id"], "padre_id": r["padre_id"],
                "imagenUsuario": r["imagenUsuario"], "nombreUsuario": r["nombreUsuario"], "imagen": r["imagen"],
                "contenido": r["contenido"]} for r in resultMapping]
```

Descripción 15

/hijo/<receta_id> - Con este método obtenemos todos los comentarios que son hijos de otros de una receta específica.

Ingrediente

```
@api_ingrediente.route("/recipe/<recipe_id>")
class IngredienteListController(Resource):
    def get(self, recipe_id):
        query = sqlalchemy.text('SELECT i.id, i.nombre, ir.cantidad FROM ingrediente_receta ir, ingrediente i WHERE '
                                'ir.recipe_id = :recipe_idRequest AND i.id IN (SELECT ingrediente_id FROM receta '
                                'WHERE id = :recipe_idRequest) GROUP BY i.id;')
        result = db.session.execute(query, {"recipe_idRequest": recipe_id})
        resultMapping = result.mappings().all()

        ingredients = [{"id": r["id"], "nombre": r["nombre"] + ' - ' + r["cantidad"]} for r in
                        resultMapping]

        return ingredients
```

Descripción 16

/recipe/<recipe_id> - Con este método obtenemos la lista de ingredientes que componen una receta.

```
@api_ingrediente.route("/busqueda/<string:name>")
class IngredienteListController(Resource):

    def get(self, name):
        name = '%' + name + '%'

        query = sqlalchemy.text('SELECT * FROM ingrediente WHERE nombre LIKE :nameRequest;')

        result = db.session.execute(query, {"nameRequest": name})
        resultMapping = result.mappings().all()

        ingredients = {r["id"]: [{r["id"], "nombre": r["nombre"]}]} for r in
            resultMapping

        return ingredients;
```

Descripción 17

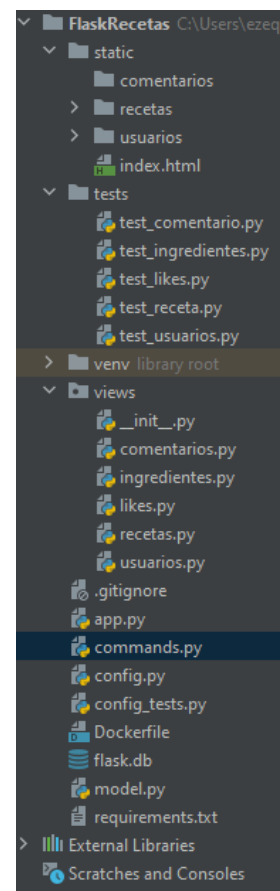
/búsqueda/<string:name> - Con este método obtenemos los diferentes ingredientes según el texto que hayamos introducido, lo usamos a la hora de crear una receta desde el front para indicar al usuario si ya existe el ingrediente introducido.

En cuanto a la organización del código en la parte back de la aplicación tenemos la carpeta static en la que guardaremos las imágenes y vídeos de las diferentes entidades.

En la carpeta test se encuentran todos los test unitarios realizados a las diferentes entidades del proyecto.

En view tenemos los controladores de las entidades con los diferentes métodos explicados anteriormente.

Luego tenemos el archivo app.py en la raíz del back el cual se encarga de iniciar la aplicación, aparte en el mismo lugar encontramos la configuración (config.py), la base de datos (flask.db), el modelo de las entidades y relaciones (model.py) y el Dockerfile de la parte back.



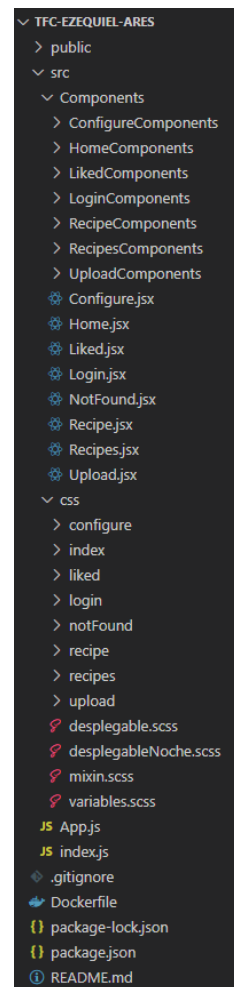
Descripción 18

En cuanto a la organización de la parte front nos encontramos con una configuración típica de React, tenemos una carpeta public donde van todas las imágenes o vídeos usados en el front.

Luego tenemos la carpeta src en la que tenemos los archivos app.js e index.js que son los encargados de renderizar los diferentes componentes.

Dentro de src tenemos la carpeta components en la que se encuentran los diferentes componentes React que hemos creado para realizar el front de la aplicación. He organizado el front teniendo un componente “padre” por cada página de la web y luego en una carpeta dentro de componentes he ido añadiendo los diferentes componentes que forman la página, de esta forma desglosamos un poco el código y lo organizamos de una mejor manera.

También dentro de src tenemos la carpeta css donde tenemos las hojas de estilo (en este caso en formato scss) de las diferentes páginas de la aplicación.



Descripción 19

Instalación

A continuación, detallaré los pasos necesarios para instalar e iniciar mi proyecto junto con diferentes capturas y explicaciones del proceso:

```
FROM python:3

RUN apt -qq -y update \
    && apt -qq -y upgrade

ADD . /FlaskRecetas
WORKDIR /FlaskRecetas

RUN pip3 install -r requirements.txt

CMD ["python3", "-m", "flask", "run", "--host=0.0.0.0"]
```

Instalacion 1

Para empezar, detallaremos los Dockerfile que hemos usado para levantar los contenedores con la aplicación, en cuanto al back he usado una imagen Python como base, luego actualizo los paquetes del sistema y copio los archivos del back a una carpeta

en el contenedor, luego cambiamos el Workdir para movernos a la carpeta. Instalamos las dependencias que tenemos en el requirements.txt y lanzamos el servidor con el comando descrito en la imagen.

```
FROM node:13.12.0-alpine

WORKDIR /home/app

COPY . .

RUN npm install

CMD ["npm", "start"]
```

Instalacion 2

En cuanto al front el Dockerfile es algo más sencillo ya que, basándonos en la imagen de Node, cambiamos el Workdir a la carpeta donde deberá ir nuestro front y copiamos los archivos en esta. Luego hacemos un npm install para instalar los paquetes necesarios e iniciamos la aplicación.

```
version: "2.2"
services:
  web:
    build: ./tfc-ezequiel-ares
    container_name: front-tfc-ezequiel
    ports:
      - "3000:3000"
    depends_on:
      - back
    volumes:
      - ${PWD}/tfc-ezequiel-ares:/app
    stdin_open: true
    tty: true

  back:
    build: ./FlaskRecetas
    container_name: back-tfc-ezequiel
    ports:
      - "5000:5000"
    volumes:
      - ${PWD}/FlaskRecetas/flask.db:/flask.db
      - ${PWD}/FlaskRecetas/static:/static
```

Instalacion 3

Por último, tenemos el Docker-compose en el cual definimos los servicios usando como base los anteriores Dockerfile, exponemos los puertos necesarios para cada servicio y le indicamos los volúmenes que necesitamos para que haya persistencia en la base de datos y los archivos que subamos.

Desarrollo de aplicaciones web

```
C:\Users\ezequ\Desktop>git clone https://github.com/IES-Rafael-Alberti/dwes-documentacion-proyecto-EzequielAres.git
```

Instalacion 4

Para empezar con la instalación, debemos tener instalado en el equipo Docker (o Docker Desktop si estamos en un sistema operativo Windows), una vez instalado iniciamos el servicio de Docker desktop si estamos en Windows y clonamos el repositorio (para ello también tendremos que tener instalado git en el equipo).

```
C:\Users\ezequ\Desktop>cd dwes-documentacion-proyecto-EzequielAres-main
C:\Users\ezequ\Desktop\dwes-documentacion-proyecto-EzequielAres-main>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: A26C-6835

Directorio de C:\Users\ezequ\Desktop\dwes-documentacion-proyecto-EzequielAres-main
08/06/2022  15:19    <DIR>          .
08/06/2022  15:19    <DIR>          ..
08/06/2022  15:19                455 docker-compose.yml
08/06/2022  15:19            13.462 EntidadRelacionTFC.pdf
08/06/2022  15:19    <DIR>          FlaskRecetas
08/06/2022  15:19            41.344 propuesta-modelo-proyecto-2022.pdf
08/06/2022  15:19    <DIR>          tfc-ezequiel-ares
08/06/2022  15:19            19.294 UML_TFC.pdf
                   4 archivos              74.555 bytes
                   4 dirs 21.261.848.576 bytes libres

C:\Users\ezequ\Desktop\dwes-documentacion-proyecto-EzequielAres-main>
```

Instalacion 5

Una vez clonado el repositorio nos movemos a la carpeta raíz de este y comprobamos que hemos clonado el repositorio satisfactoriamente.

```
C:\Users\ezequ\Desktop\dwes-documentacion-proyecto-EzequielAres-main>docker-compose up -d
WARNING: The PWD variable is not set. Defaulting to a blank string.
Creating network "dwes-documentacion-proyecto-ezequielares-main_default" with the default driver
Building back
[*] Building 1.1s (2/3)
=> [internal] load build definition from Dockerfile                                0.0s
=> => transferring dockerfile: 243B                                              0.0s
=> [internal] load .dockerignore                                                 0.0s
=> => transferring context: 2B                                                  0.0s
=> [internal] load metadata for docker.io/library/python:3                     1.0s
```

Instalacion 6

Una vez comprobada la integridad de los archivos del proyecto podemos iniciarlo simplemente corriendo el comando docker-compose up -d en la raíz de la carpeta, esto ejecutará los Dockerfile dentro del Docker-compose y creará la imagen de estos, la primera vez tardará ya que necesita mover ciertas carpetas al contenedor para que funcione correctamente.

```
Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
WARNING: Image for service web was built because it did not already exist. To rebuild this image you must use `docker-compo
e build` or `docker-compose up --build`.
Creating back-tfc-ezequiel ... done
Creating front-tfc-ezequiel ... done
```

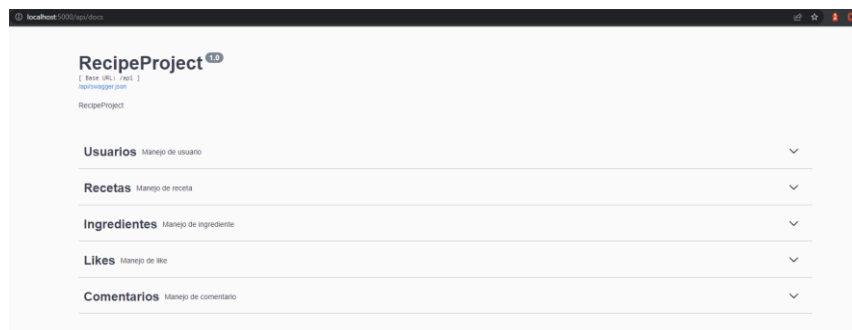
Instalacion 7

Una vez terminado tendremos esta respuesta indicando que se han creado los dos contenedores correctamente.



Instalacion 8

Podemos comprobar el funcionamiento de la parte front entrando en el localhost:3000, el cual nos debe dirigir a la página home de la aplicación.



Instalacion 9

También podemos acceder al localhost:5000/api/docs para ver la documentación de la API y así comprobar que tenemos respuesta del servidor (también podemos comprobar que funciona simplemente navegando por la página y viendo que cargan las recetas, etc...)

Si las recetas no cargan, pero tenemos acceso al servidor desde la dirección nombrada en la captura anterior es problema del CORS, para solucionarlo tendremos que instalar en nuestro navegador web un plugin para activar el CORS.

En cuanto a iniciar sesión dejo una tabla con los diferentes usuarios de la aplicación:

Nombre de usuario	Contraseña	Es administrador
Ezequiel	pestillo	Sí
Ana	pestillo	No
Paco	pestillo	Sí
María	pestillo	No
Alejandro	pestillo	Sí

Guía de estilos

A continuación, detallaré los diferentes elementos que componen la identidad visual de mi proyecto, además de la guía de estilos utilizada en la web y detallaremos los wireframes y mockups de algunas de las diferentes secciones de la página.

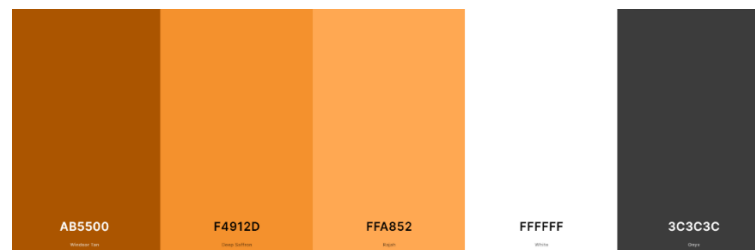
Nombre de la empresa

Para mi proyecto he pensado como nombre de empresa “Cookpedia” el cual combina las palabras “Cook” del inglés que significa cocinar o cocinero y el sufijo “pedia” que proviene del griego “paideia” el cual significa educación o enseñanza, podemos encontrar este sufijo en diferentes palabras como enciclopedia, logopedia o incluso Wikipedia, por lo que pienso que es fácil distinguir la finalidad de mi página web.

Eslogan

El eslogan que he escogido para mi página web es “looking recipes?” el cual se traduciría como “¿Buscas recetas? Dando a entender que la página es el lugar idóneo para encontrar miles de recetas, además tiene un punto cómico ya que, con el nombre de la web, el contenido de la página, etc... se sobreentiende que la persona está buscando recetas de cocina.

Colores



Guía estilos 1

Esta es la paleta de colores que he elegido para mi proyecto, para empezar tenemos 3 tonalidades diferentes de naranja, he elegido este color porque es el que transmite mejor la sensación de página de cocina (por gusto personal es el primer color en el que pensé cuando me planteé una web de cocina, y según las sensaciones también es el más adecuado), al ser un color cálido transmite un ambiente agradable además de que destaca lo suficiente para llamar la atención lo cual es necesario en una web de comida, las tonalidades las combinaremos con los demás colores de la web y los usaremos en diferentes situaciones para resaltar más o menos los textos.

Para el fondo tenemos un blanco puro el cual nos permite resaltar bastante los títulos con los colores anteriores y el texto de un negro.

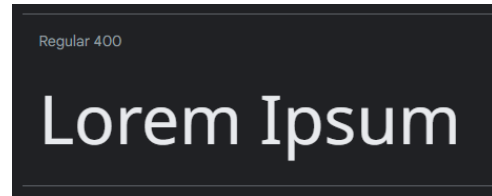
Esta paleta también nos ayuda bastante en el modo oscuro de la web ya que destaca muy bien si le añadimos un borde negro a los títulos con las tonalidades naranjas, como fondo utilizaremos el último color, el cual es un gris oscuro con el que podremos destacar el texto blanco de forma sencilla.

Tipografía



Guía estilos 2

Croissant One



Guía estilos 3

Noto Sans

En cuanto a las tipografías he escogido una fuente la cual capte la atención del usuario y transmita algo de fuerza, pero siendo agradable.

Croissant One es la elegida para los diferentes elementos que queremos destacar como títulos, menús, logo, etc... con esta fuente tenemos un pequeño problema y es que si tenemos mucho texto es un poco complicado distinguir las letras por lo que le he aplicado un borde negro y un pequeño espaciado entre letras para arreglarlo.

La fuente **Noto Sans** es una fuente de tipo San Serif que utilizaremos para los diferentes textos de la página como detalles, comentarios, pasos, ingredientes, etc... ya que esta fuente es fácil de entender.

Como podemos ver, no he querido combinar muchas fuentes diferentes para no saturar la página con diferentes estilos ya que pienso que sería contraproducente para el objetivo que tengo con la web el cual es destacar los elementos necesarios con la máxima simpleza posible.

Iconografía



Guía estilos 4

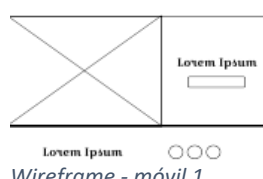
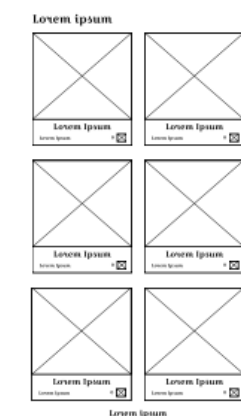
Para empezar, tenemos el imagotipo del proyecto, en este combinamos el logotipo el cual es la parte de Cookpedia con la tipografía que hemos descrito en el punto

anterior y el isotipo, el cual es un libro de recetas el cual se compone de los colores de la guía de estilos que componen la identidad visual del proyecto y un gorro de cocina para simbolizar que es un libro de recetas. Con este imago tipo he querido transmitir el objetivo de la página para que cualquiera que lo vea sepa inmediatamente el contenido de la web, ya sea viendo el imago tipo completo o cada parte. Además, destaca bastante y consigue transmitir esa calidez y ambiente agradable que he intentado mantener con los diferentes elementos que componen el proyecto.

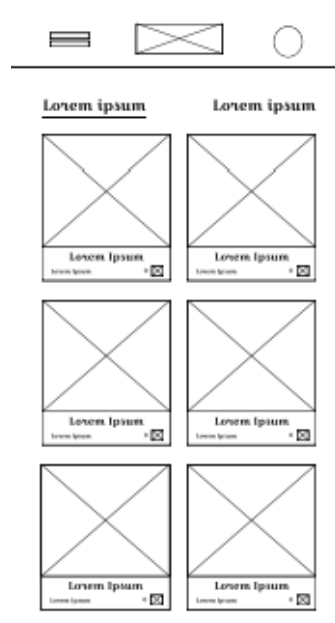
Prototipado

Para el prototipado de la web he seguido el patrón mobile first por lo cual he empezado el diseño de los wireframe y mockup desde su versión móvil. A continuación, adjuntaré los diferentes diseños tanto móviles como de escritorio.

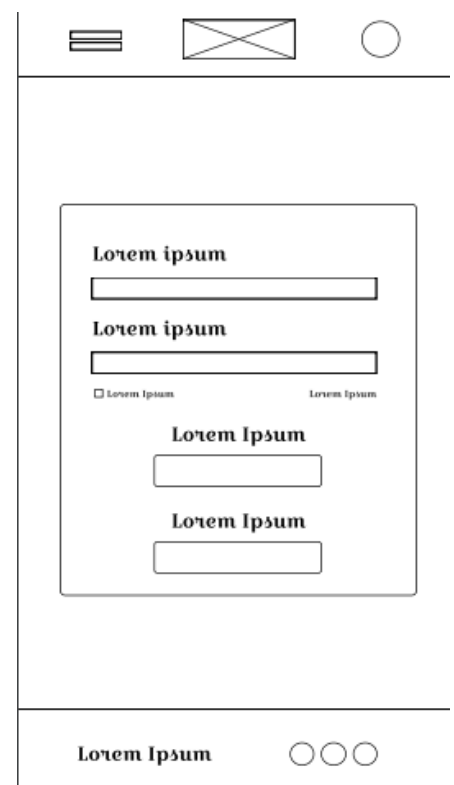
WireFrame - Móvil



Wireframe - móvil 1

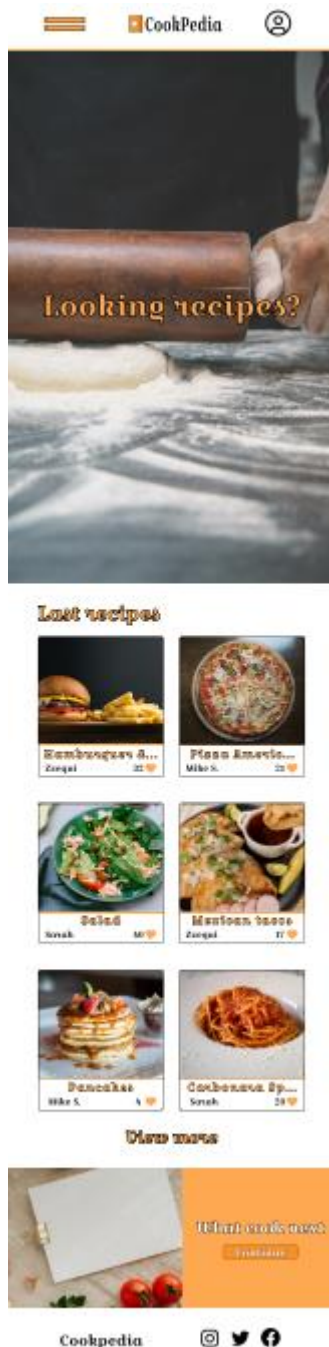


Wireframe - móvil 2

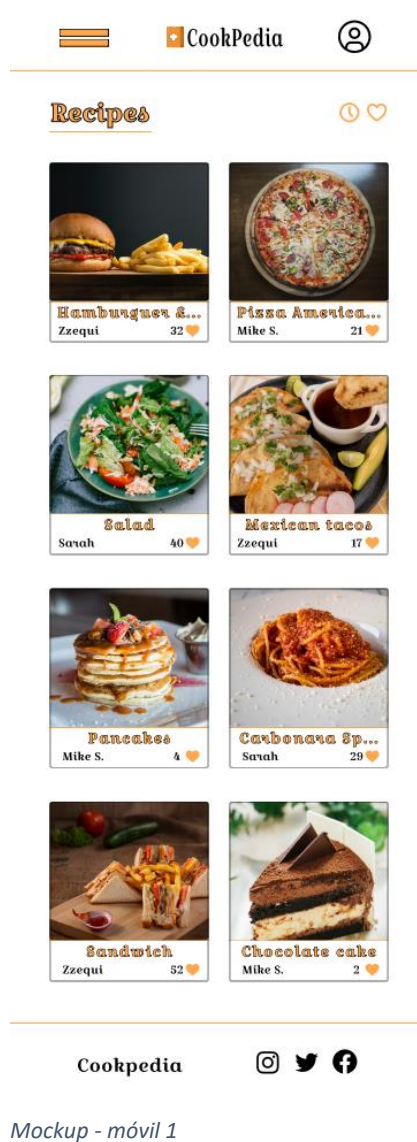


Wireframe - móvil 3

Mockup – Móvil



Mockup - móvil 3

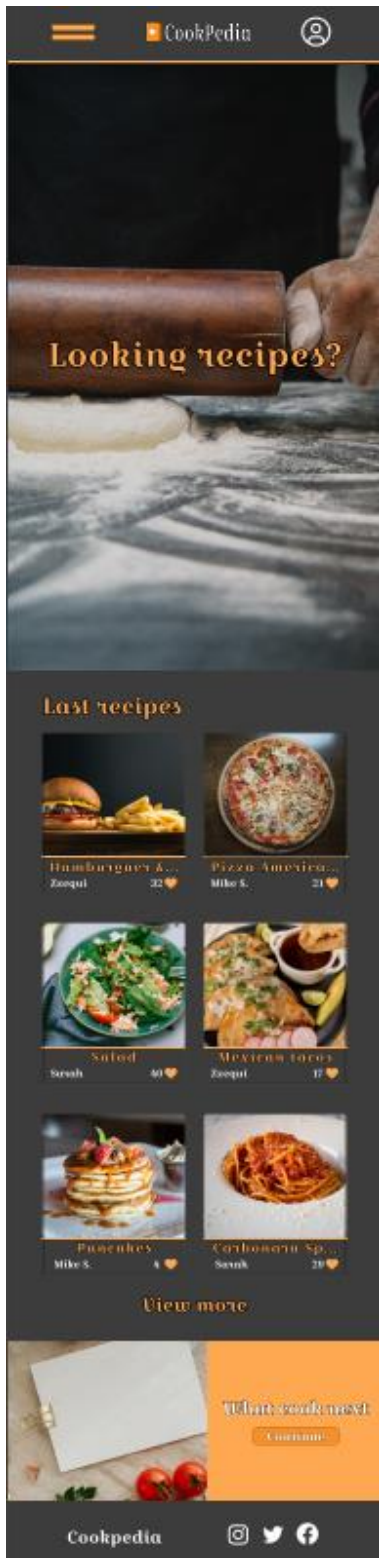


Mockup - móvil 1



Mockup - móvil 2

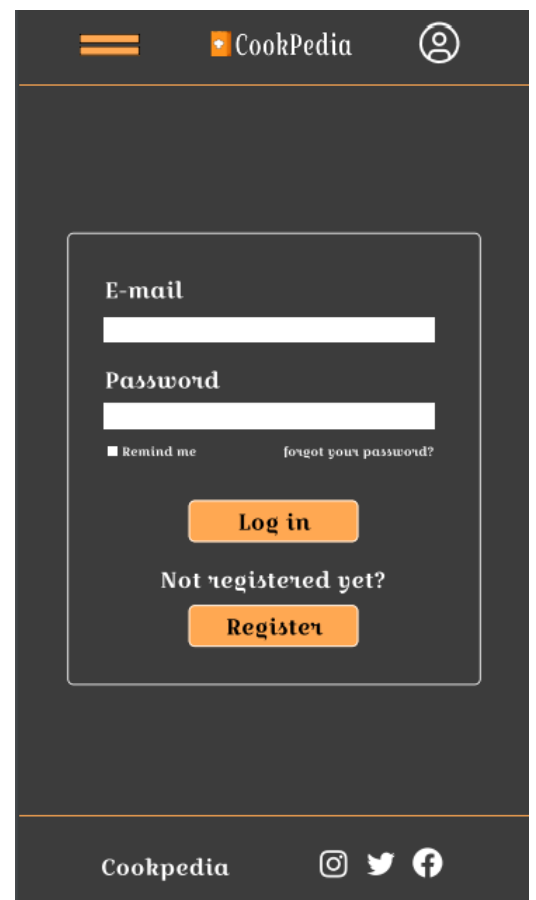
Mockup – Móvil noche



Mockup - móvil 4

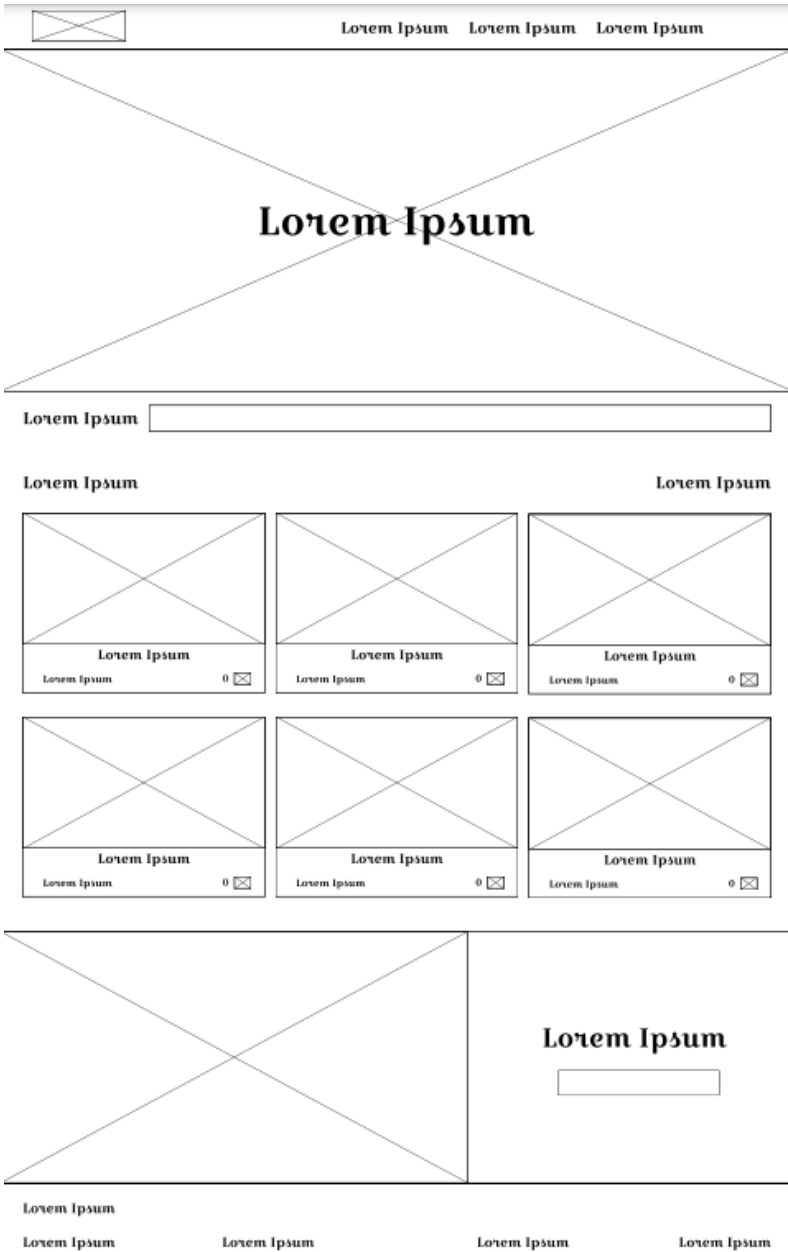


Mockup - móvil 6

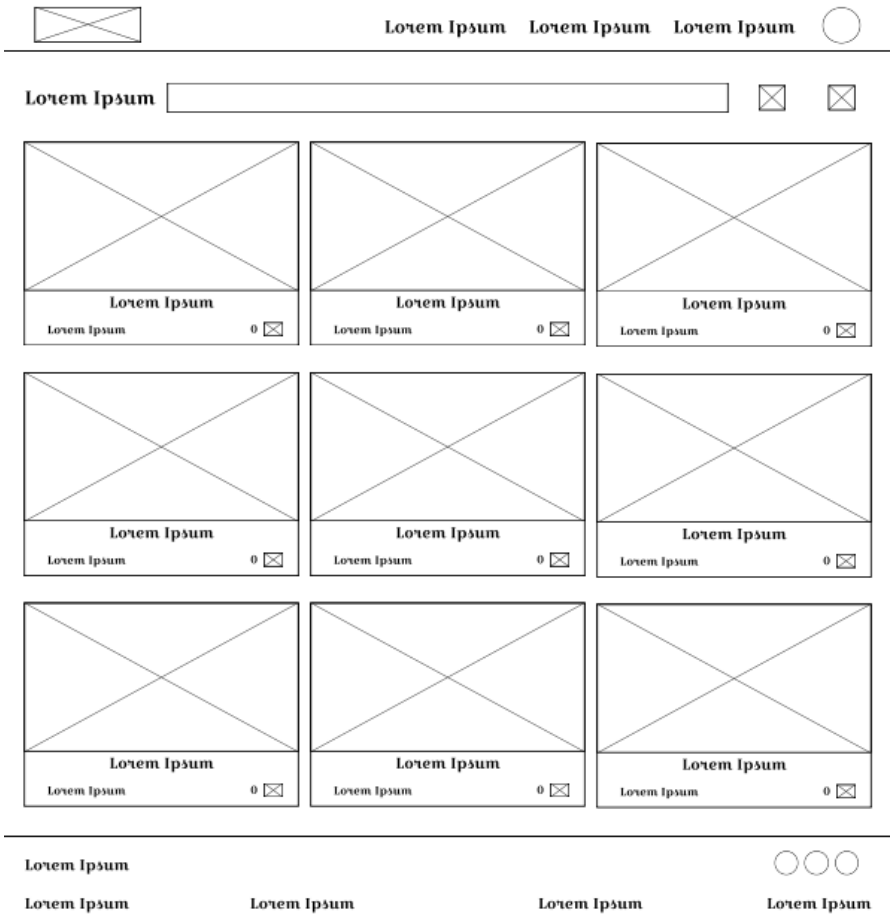


Mockup - móvil 5

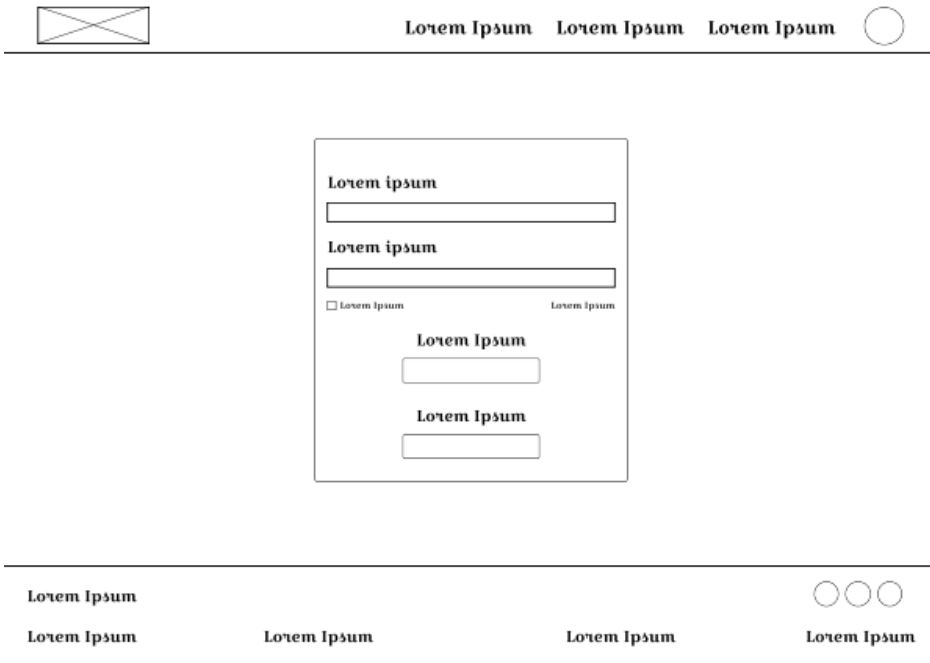
Wireframe – Desktop



Wireframe - Desktop 1

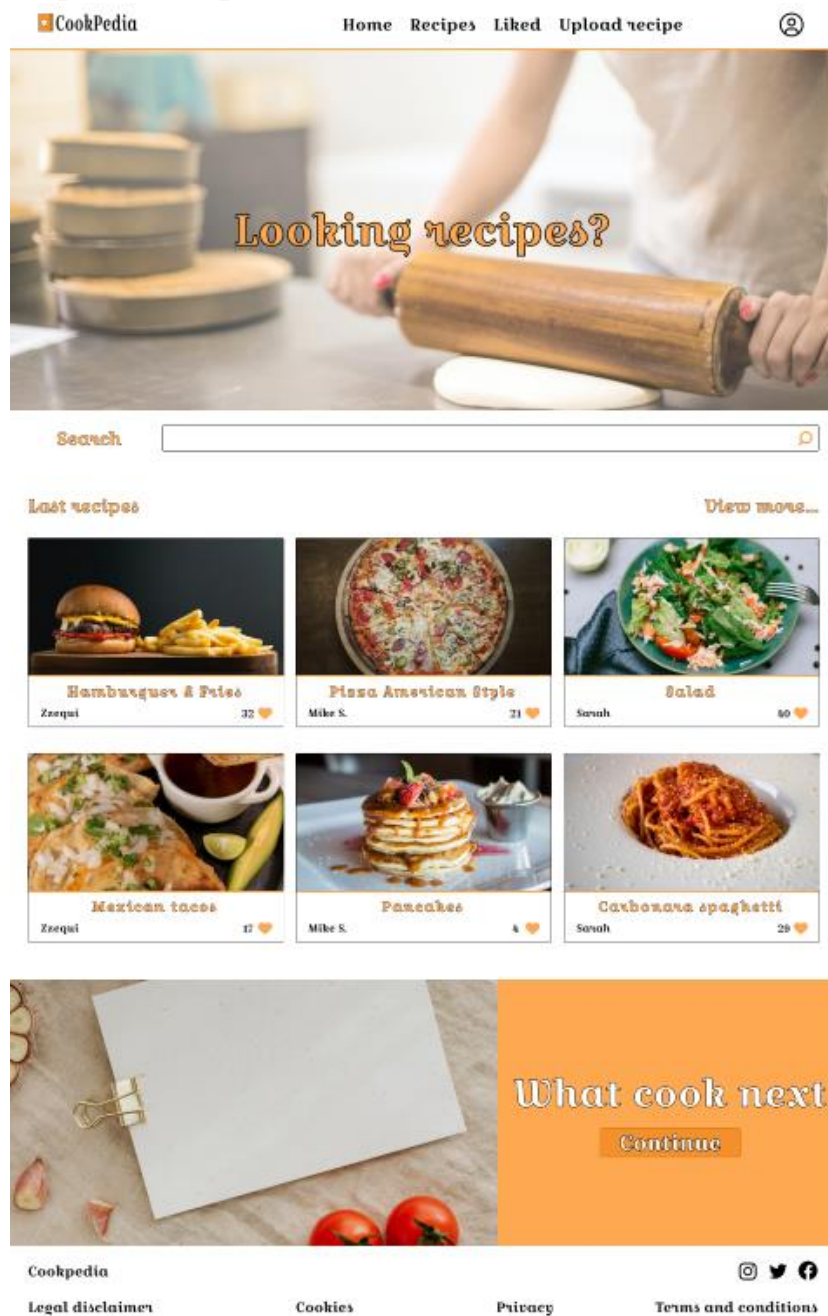


Wireframe - Desktop 2

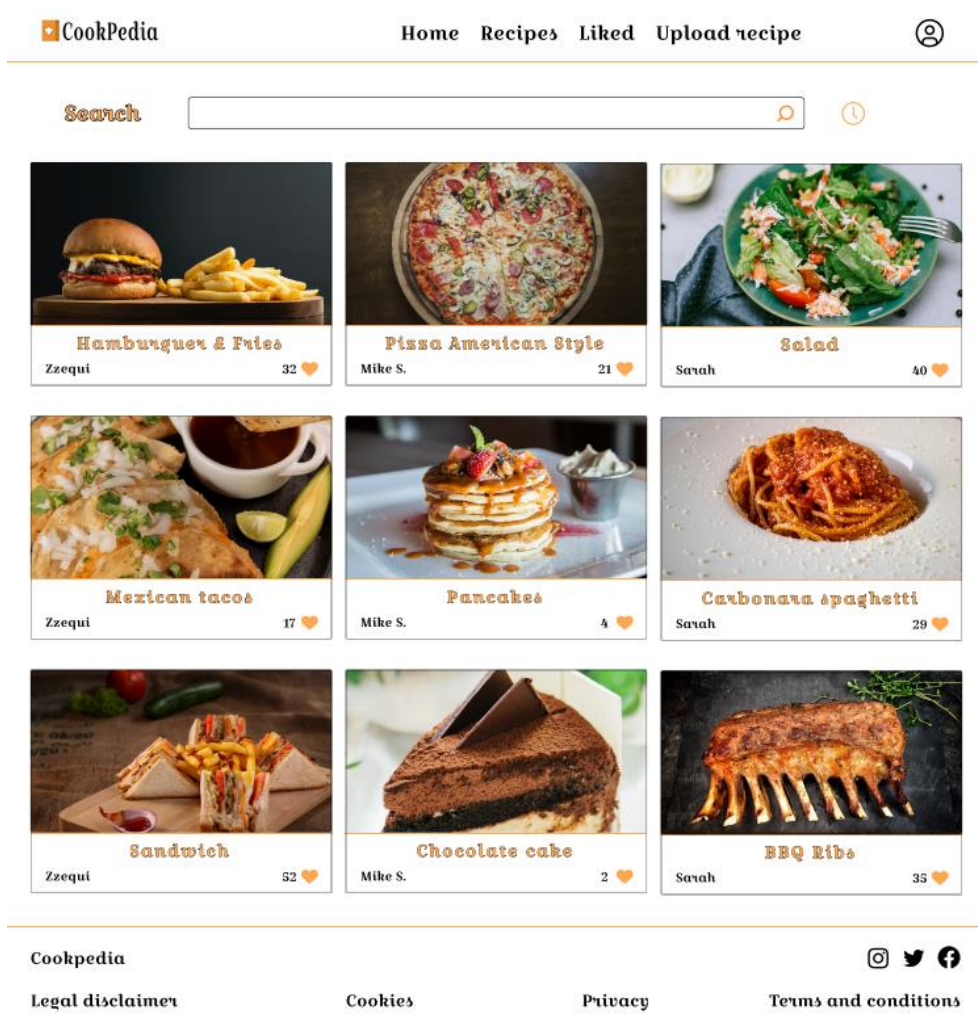


Wireframe - Desktop 3

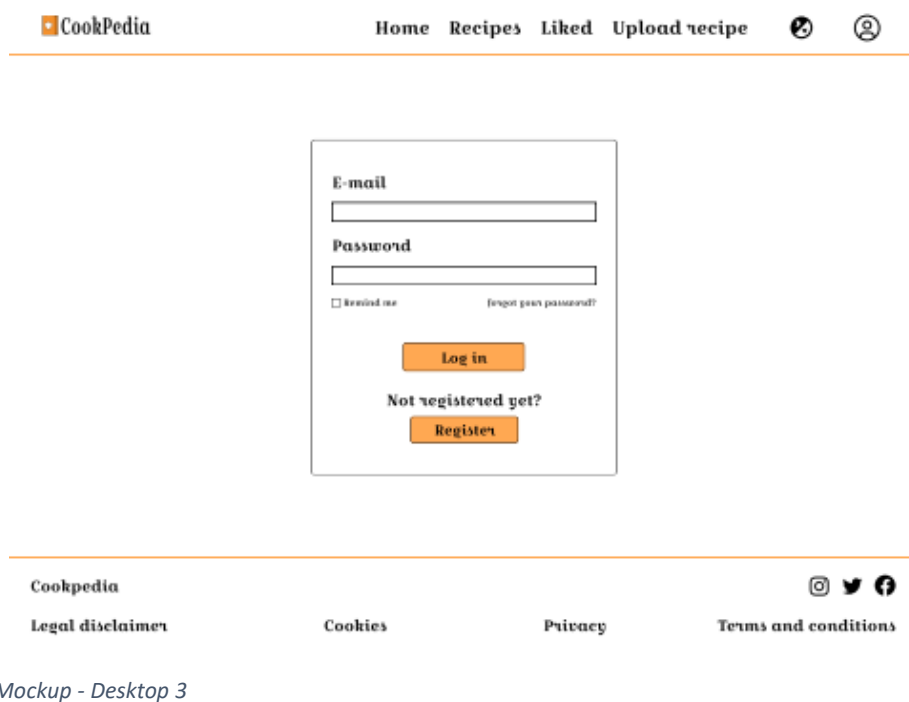
Mockup – Desktop



Mockup - Desktop 1

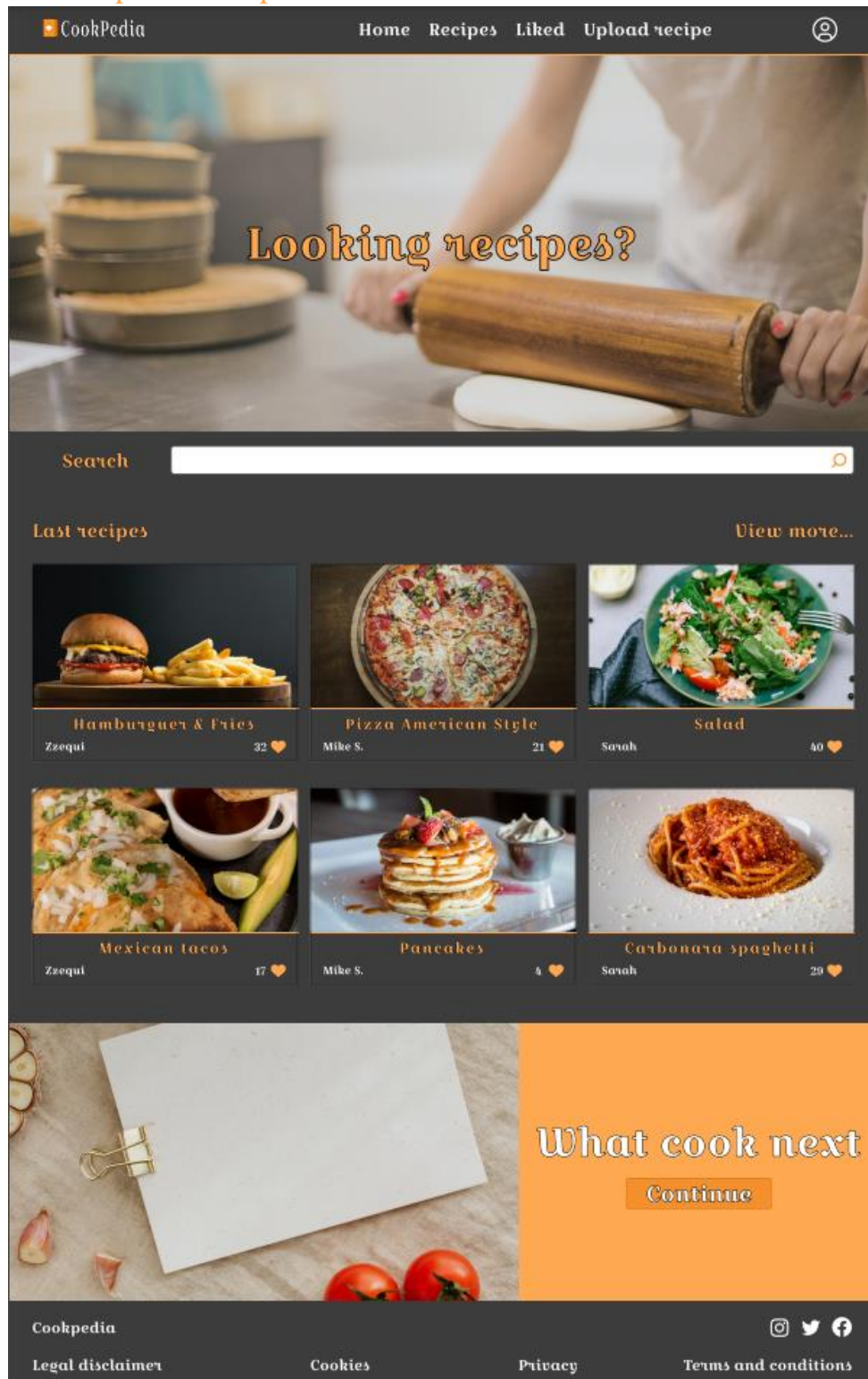


Mockup - Desktop 2

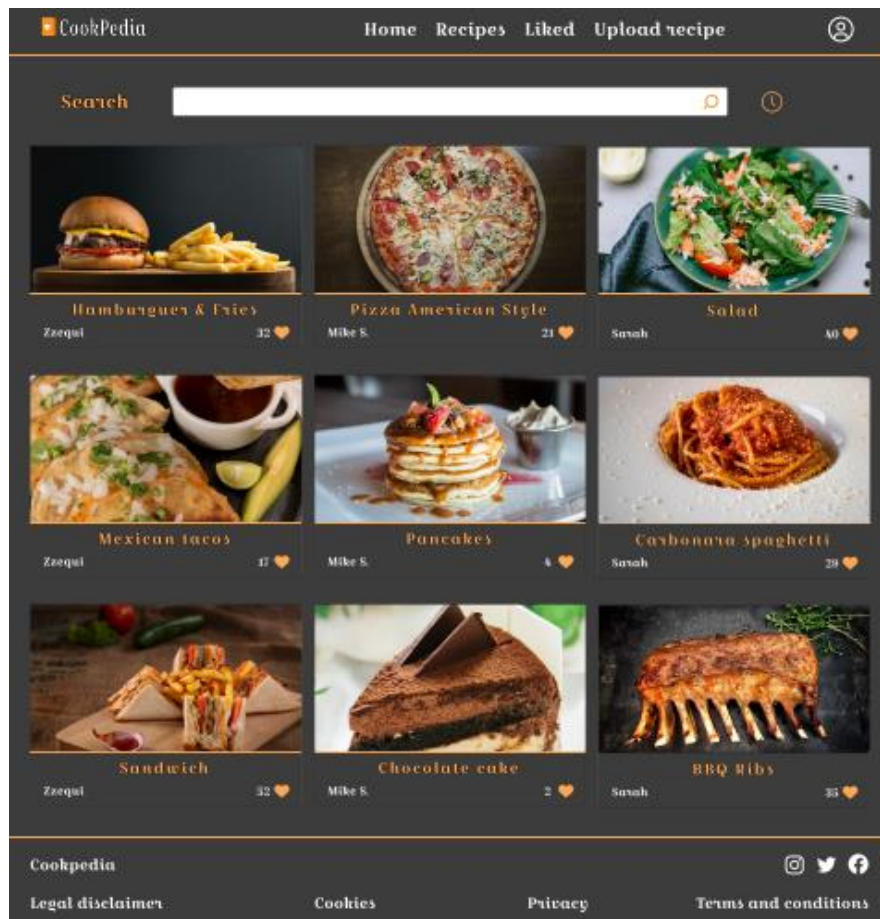


Mockup - Desktop 3

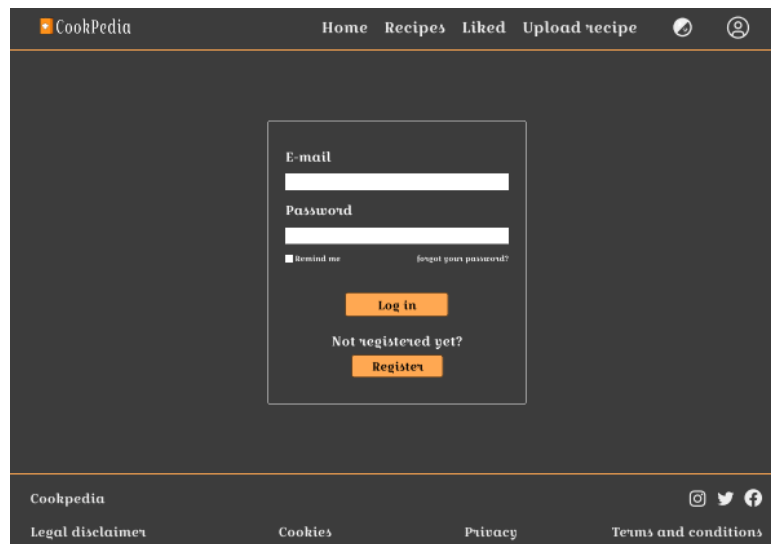
Mockup – Desktop Modo Noche



Mockup - Desktop 4



Mockup - Desktop 5



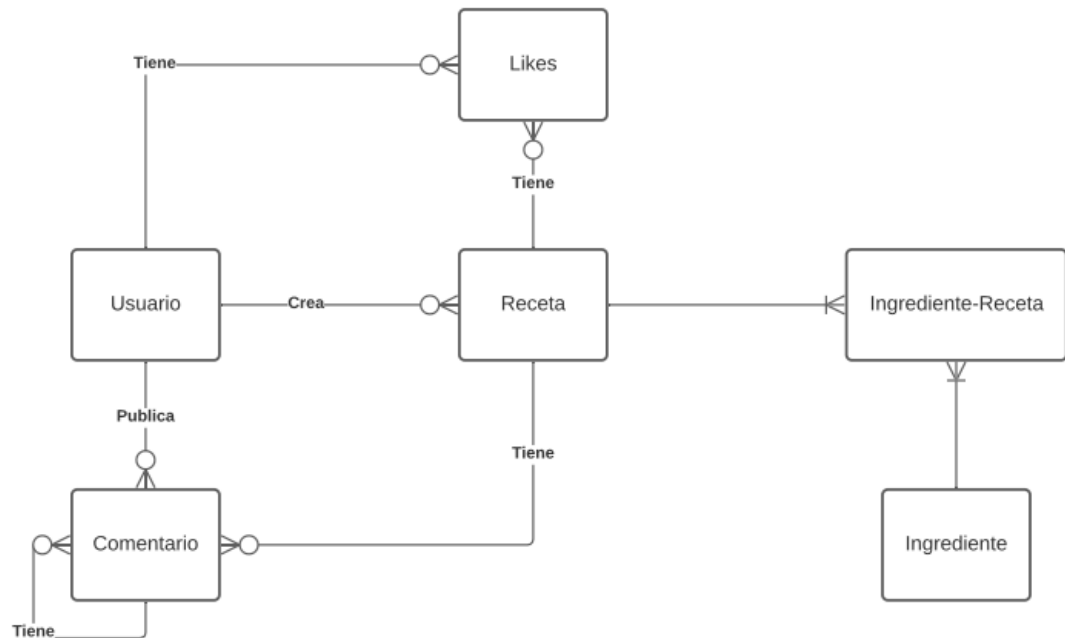
Mockup - Desktop 6

Como hemos podido ver en las imágenes anteriores, se ha decidido usar un diseño simple sin muchos elementos en pantalla para destacar las recetas de la web, en la carpeta adjunta tendremos los documentos con todos los mockups y wireframes de la página ya que mostrar todos en la documentación sería pesado.

Diseño

A continuación, adjuntaré alguna imagen de la entidad relación y el UML del proyecto además de justificar las relaciones y entidades creadas en la aplicación.

Entidad relación



Diseño 1

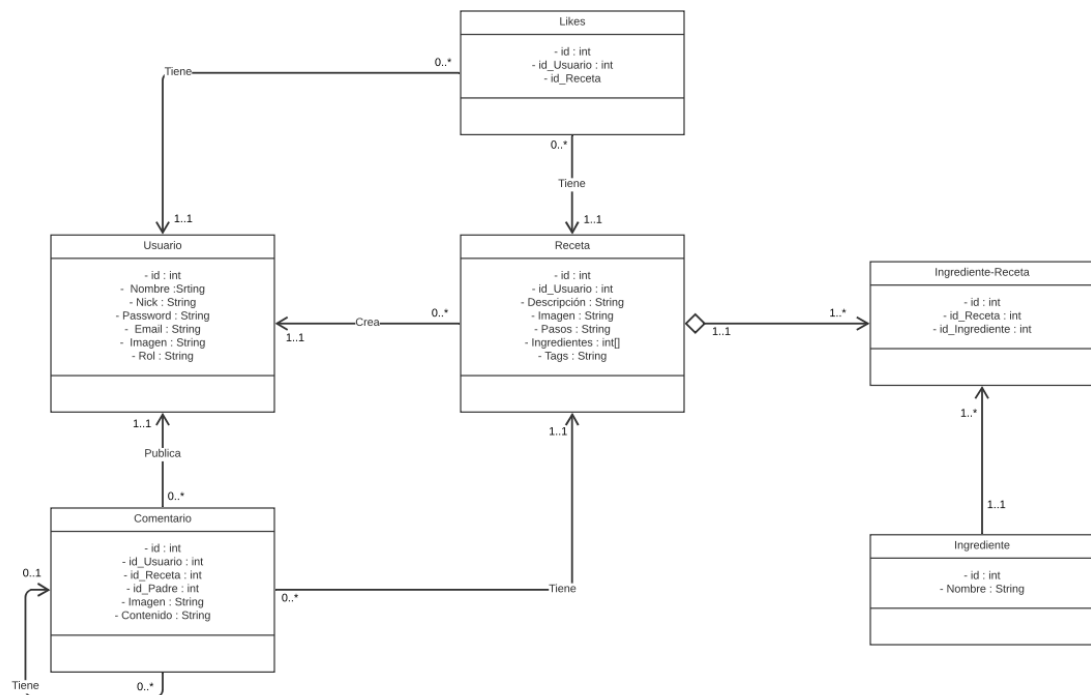
Como podemos ver, la aplicación dispone de 6 entidades con diferentes entidades. Comenzando con la entidad Usuario vemos que tiene relaciones con Likes, Receta y comentario, esto se debe a que necesitamos guardar a qué usuario pertenece cada entidad además que podrá estar presente múltiples filas de las entidades.

Mas abajo vemos la entidad comentario, la cual como hemos descrito antes está relacionada con Usuario, pero también con Receta, estas relaciones son necesarias para identificar un comentario específico. También podemos ver que disponemos de una relación reflexiva la cual nos permite distinguir si el comentario es hijo de otro.

En cuanto a la entidad Likes no es más que una tabla con relación a Usuario y Receta para guardar el número de likes que tiene una receta.

Para terminar, tenemos Ingrediente, la cual es una entidad que tiene una relación muchos a muchos con Receta la cual se refleja con la creación de una tabla externa Ingrediente-Receta, la cual nos permitirá saber qué ingredientes tiene una receta.

UML



Diseño 2

Debido a que este UML lo hice en la primera etapa del desarrollo del proyecto hay campos que se han eliminado con el tiempo, igualmente lo veo una buena manera de visualizar que el proyecto va cambiando y adaptándose a las necesidades del front. A continuación, mencionaré algunos aspectos importantes de las entidades.

En cuanto al Usuario tuve que eliminar el campo de Rol ya que no lo implementé en el proyecto, en vez de esto dispone de un campo booleano que me permite saber si es administrador, no se han realizado más cambios en esta entidad.

Sobre la entidad Receta eliminé el campo tags el cual iba a usar como filtro en las búsquedas, pero por temas de tiempo no lo pude implementar, como no tenía ningún otro uso decidí eliminar el campo.

Sobre la entidad IngredienteReceta añadí un campo cantidad para poder introducir una cantidad diferente por cada ingrediente y receta, ya que habrá recetas que con un mismo ingrediente usen diferentes cantidades.

Desarrollo

Secuencia de desarrollo y dificultades encontradas

En cuanto al desarrollo del proyecto en mi caso lo he ido dividiendo en etapas, la primera etapa la centré en la creación de las entidades y controladores con su CRUD, además de realizar los test unitarios necesarios para probar los métodos creados. En esta etapa tuve algunos problemas en cuanto a la creación de las relaciones ya que por ejemplo nunca había plasmado una relación reflexiva en Flask, pero gracias a la documentación oficial pude sacar adelante este problema.

Una vez creadas las entidades y demás me centré en crear la guía de estilos y prototipado del proyecto para tener una idea más clara de los puntos de entrada en la API que necesitaría más adelante, en esta etapa no tuve ningún problema (aparte de problemas creativos, me costó un poco visualizar cómo estructurar la página y qué elementos añadir), una vez terminado realicé los métodos que pensé necesarios para suplir las necesidades del front.

Cuando terminé toda la parte del prototipado comencé la aplicación en React, al principio fue un problema estructurar los componentes que iba a necesitar, en toda la etapa del desarrollo del front tuve algunos problemas:

- CSS – Ya que necesitaba múltiples estilos por cada página de la web y sus modos noche tuve problemas para importar los diferentes estilos. Estuve investigando diferentes plugin de React como Styled-components, pero no conseguía solucionar el problema como me gustaría. Al final, la única solución que se me ocurrió fue hacer un require de los diferentes estilos de cada página y nombrar las clases de los elementos con un prefijo que define la página donde se encuentra (Ej. Recipе__elemento).
- Autenticación – Otro problema fue como guardar la autenticación de usuario ya que para algunas peticiones necesitamos saber el jwt del usuario actual o su id, mi primera idea fue utilizar props ya que es algo propio de React, pero no podría hacerlo con esto ya que no puedo comunicar componentes hijos y hermanos, también investigué con algún plugin, pero al final mi solución fue guardar el jwt y el id del usuario en el localStorage del navegador.
- Renderización – Luego otro problema fue donde se renderizan los componentes, ya que estos se crean dentro de una etiqueta con el id root en el index de React, en cuanto a estructura de la página se renderizaba dentro de un div lo cual no es lo óptimo en cuanto a

estructuración del HTML por lo que cambié el lugar de renderización al body para así poder introducir etiquetas como main o footer (aunque esto no lo recomienda React).

- API – En cuanto a la API creía haber realizado todos los puntos de entrada necesarios para crear directamente todo el front, pero nada más lejos de la realidad, los puntos de entrada estaban bien (aunque faltaba alguno que otro) pero no devolvía todos los datos que necesitaba en el front, por lo que tuve que cambiar los puntos de entrada por sentencias SQL personalizadas para tener todos los datos solicitados.

Después de haber realizado todos los cambios necesarios en la API y terminar el front realicé el despliegue en Docker, desplegar la app de React fue bastante sencillo ya que pude ver muchos ejemplos en diferentes foros, en cuanto a desplegar el back me fue más complicado ya que conseguía correr el contenedor, pero no tenía acceso al servidor, después de muchas pruebas cambiando el Dockerfile me di cuenta que el problema era el comando usado para correr internamente el servidor Flask. Una vez cambiado pude desplegar sin problemas la aplicación, probando la persistencia de datos.

Por último, me dediqué a mejorar diferentes aspectos de la aplicación y comentar todo el código (también cambié algunos elementos de las pruebas unitarias, ya que al tener que cambiar puntos de entrada de la API tuve que modificar las pruebas de estos).

Herramientas de control de versiones y revisión del código

En cuanto a este punto he usado como herramienta de control de versiones git, alojando en un repositorio de GitHub el proyecto, con esto he tenido algunos problemas ya que aparte de no ser muy constante con los commits del repositorio no me di cuenta que el proyecto base que usé para el front estaba enlazado en el repositorio como un submódulo y no lo supe hasta la etapa final del proyecto cuando ya tenía todo el front y back realizado, me puse a investigar y tuve que borrar el submódulo y subir el front de forma normal, después de haber realizado estos cambios en el repositorio no tuve ningún problema.

En cuanto a revisión del código las herramientas automáticas de visual studio code y pycharm han sido encargadas de esta tarea, estas marcan con diferentes tipos de errores (avisos, elementos a mejorar, errores graves) los diferentes elementos del código, en la última etapa de las descritas en el

punto anterior me puse a mejorar el código siguiendo las indicaciones de estas herramientas.

Pruebas

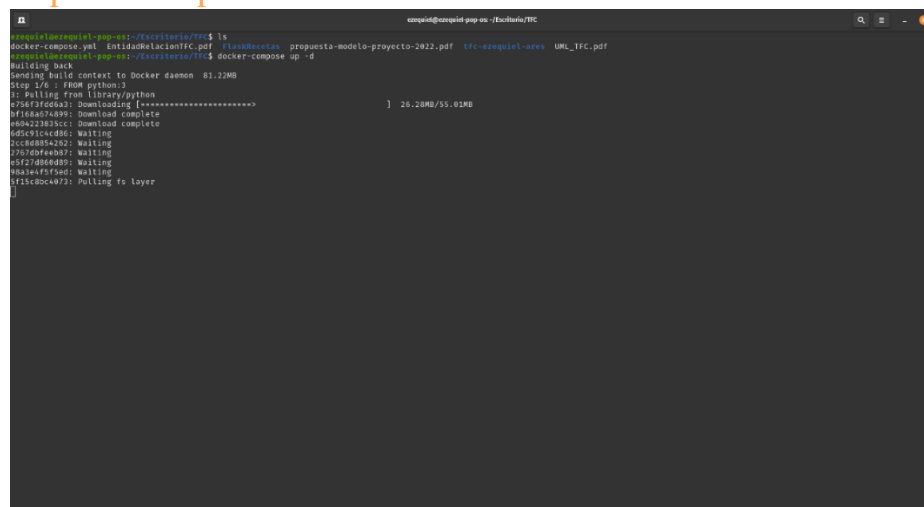
En cuanto a las pruebas realizadas en el código aparte de una revisión manual con Insomnia y probando todo tipo de posibilidades desde las diferentes páginas del front, también realicé pruebas unitarias de algunos métodos creados en las entidades.

Estos test consisten en probar el correcto funcionamiento del CRUD de todas las entidades realizadas, como algunos métodos fueron adaptados para el front tuve que cambiar algunas pruebas para en vez de mandar un json mandar un multipart/form-data con los datos necesarios para probar los diferentes métodos.

Despliegue

El despliegue de la aplicación se ha realizado con Docker, para ello he creado un Dockerfile para el front y back con los elementos necesarios para su correcto funcionamiento (copiar contenido local a carpetas del contenedor, correr ciertos comandos para inicializar la aplicación dentro del contenedor, etc...), una vez creados estos Dockerfile he creado un Docker-compose el cual recoge los dos anteriores Dockerfile y establece los volúmenes, puertos y configuraciones necesarias para que estos se comuniquen entre ellos, que tengamos acceso a los contenedores desde nuestro equipo local y que se mantenga persistencia de datos para los elementos necesarios como la base de datos del servidor y la carpeta estática donde se guardan las imágenes y videos de las entidades.

Descripción del proceso



```
ezequiel@ezequiel-pop-us: ~/Escritorio/TFC
$ docker-compose up -d
Building back
Sending build context to Docker daemon 61.22MB
Step 1/6 : FROM python:3
3: Pulling from library/python
07583f6d6a3: Downloading [*****] 25.29MB/55.91MB
0f166a574999: Download complete
480d22283ccc: Download complete
6d5c91c4c486: Waiting
1cc408b54202: Waiting
79240f4e0d87: Waiting
e5f27d86d889: Waiting
8ba4eef3f9ed: Waiting
f15cb0c4973: Pulling fs layer
]
```

Despliegue 1

Desarrollo de aplicaciones web

Para describir el proceso de la mejor manera posible he eliminado todas las imágenes y contenedores que tenía en el equipo, para desplegar el proyecto simplemente nos colocamos en la carpeta raíz del repositorio (donde se encuentra el archivo docker-compose) y ejecutamos el comando Docker-compose up -d (para que mantenga ejecutando en segundo plano, si nos bloqueará la terminal).

```
Successfully built 3a0d0e0e0e0e
Successfully tagged tfc-back:latest
WARNING: Image for service back was built because it did not already exist. To rebuild this image you must use `docker-compose build` or `docker-compose up --build`.
Building web
Sending build context to Docker daemon 714MB
Step 1/7 : FROM node:13.12.0-alpine
13.12.0-alpine: pulling from library/node
a4d03a93944: Pull complete
4000d932280e: Pull complete
c274c500937: Pull complete
f3446470f297: Pull complete
Digest: sha256:81cc85e728fab3827da76a181b3a286caefab025f256e2c80909409bfe814766
Status: Downloaded newer image for node:13.12.0-alpine
--> 483334dc5f5
Step 2/7 : WORKDIR /app
--> Running in 84eed3d56aa
Removing intermediate container 84eed3d56aa
--> 6345dc608df
Step 3/7 : COPY package.json .
--> cbff5f0bf45
Step 4/7 : COPY package-lock.json .
--> 4cfff39d1a3f
Step 5/7 : RUN npm install
--> Running in 6245dc608df
npm WARN read-shrinkwrap This version of npm is compatible with lockfileVersion@0, but package-lock.json was generated for lockfileVersion@1. I'll try to do my best with it!
> @fontawesome/fontawesome-common-types@6.1.1 postinstall /app/node_modules/@fontawesome/fontawesome-common-types
> node attribution.js
Font Awesome Free 6.1.1 by @fontawesome - https://fontawesome.com
License - https://fontawesome.com/license/free (Icons: CC BY 4.0, Fonts: SIL OFL 1.1, Code: MIT License)
Copyright 2022 Fonticons, Inc.
> core-js@2.6.3 postinstall /app/node_modules/core-js
> node -e "try{require('./postinstall')}catch(e){}"
Thank you for using core-js ( https://github.com/zloirock/core-js ) for polyfilling JavaScript standard library!
The project needs your help! Please consider supporting of core-js:
> https://opencollective.com/core-js
> https://patreon.com/zloirock
> https://paypal.me/zloirock
> bitcoin: bc1qea754qtsm3r2rayg0thz9fau63ux0fstz
Also, the author of core-js ( https://github.com/zloirock ) is looking for a good job -)
> core-js-pure@2.6.3 postinstall /app/node_modules/core-js-pure
> node -e "try{require('./postinstall')}catch(e){}"
```

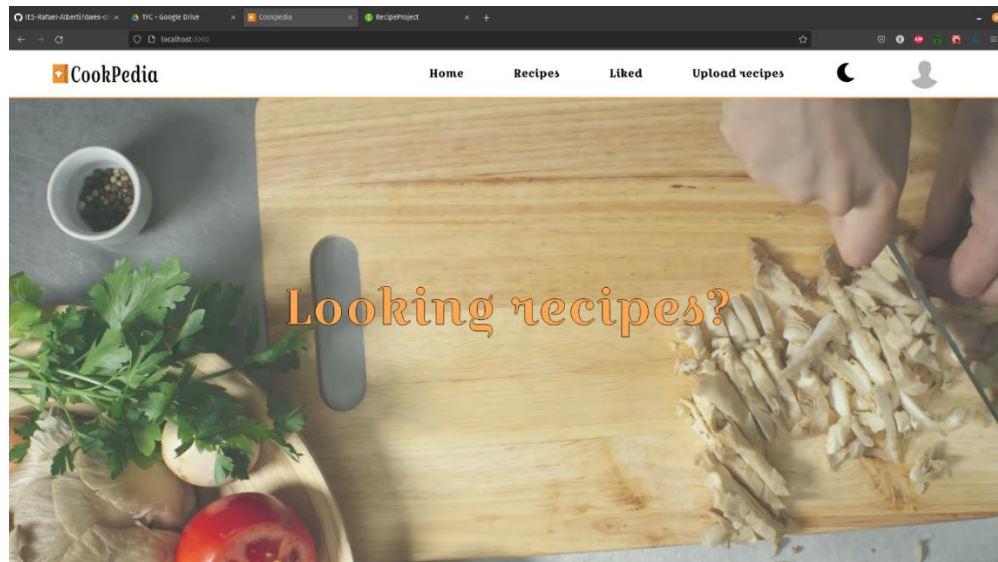
Despliegue 2

Una vez ejecutado el anterior comando se empezarán a descargar las imágenes necesarias y se ejecutarán las diferentes configuraciones de los Dockerfiles, en esta captura podemos ver el comienzo de descarga del front y su configuración.

```
The project needs your help! Please consider supporting of core-js:
> https://opencollective.com/core-js
> https://patreon.com/zloirock
> https://paypal.me/zloirock
> bitcoin: bc1qea754qtsm3r2rayg0thz9fau63ux0fstz
Also, the author of core-js ( https://github.com/zloirock ) is looking for a good job -)
> @fontawesome/fontawesome-svg-core@6.1.1 postinstall /app/node_modules/@fontawesome/fontawesome-svg-core
> node attribution.js
Font Awesome Free 6.1.1 by @fontawesome - https://fontawesome.com
License - https://fontawesome.com/license/free (Icons: CC BY 4.0, Fonts: SIL OFL 1.1, Code: MIT License)
Copyright 2022 Fonticons, Inc.
> @fontawesome/free-solid-svg-icons@6.1.1 postinstall /app/node_modules/@fontawesome/free-solid-svg-icons
> node attribution.js
Font Awesome Free 6.1.1 by @fontawesome - https://fontawesome.com
License - https://fontawesome.com/license/free (Icons: CC BY 4.0, Fonts: SIL OFL 1.1, Code: MIT License)
Copyright 2022 Fonticons, Inc.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})
added 1406 packages from 692 contributors and audited 1407 packages in 15.677s
168 packages are looking for funding
run `npm fund` for details
found 18 vulnerabilities (2 moderate, 4 high, 12 critical)
run `npm audit fix` to fix them, or `npm audit` for details
Removing intermediate container 6345dc608df
--> 18d8b0a632cc
--> a370ef5c1d
Step 7/7 : CMD ["npm", "start"]
--> Running in 6245dc608df
Removing intermediate container 6245dc608df
--> 25f5f5d19031
Successfully built 25f5f5d19031
Successfully tagged tfc-web:latest
WARNING: Image for service web was built because it did not already exist. To rebuild this image you must use `docker-compose build` or `docker-compose up --build`.
Creating back-tfc-ezequiel ... done
Creating front-tfc-ezequiel ... done
Creating tfc-ezequiel-pop-01 -> EzequielAresRodriguez
```

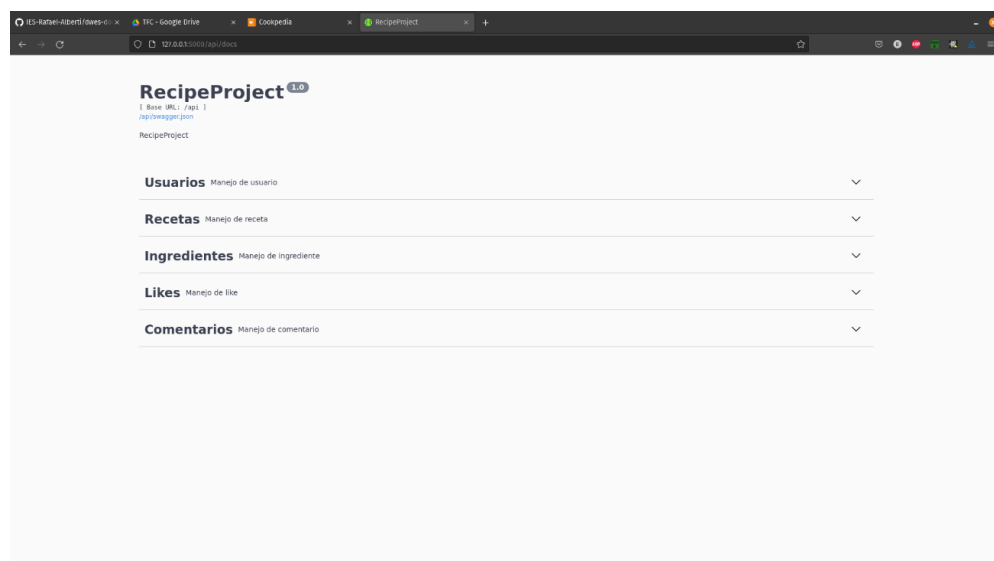
Despliegue 3

Una vez terminado los diferentes comandos internos del Dockerfile podremos ver al final que se han creado correctamente los contenedores.



Despliegue 4

Ahora si nos dirigimos al puerto 3000 de nuestro localhost tendremos acceso al front de la aplicación.



Despliegue 5

También si nos dirigimos al `localhost:5000/api/docs` podremos ver la documentación de los puntos de entrada de la API, con esto comprobamos que tenemos acceso al servidor.

Para que las peticiones se realicen correctamente desde el front y podamos cargar las recetas, iniciar sesión, etc... debemos tener un plugin instalado en el navegador para activar el CORS, si no lo instalamos las peticiones no funcionarán.

Manual

Header

[Home](#)[Recipes](#)[Liked](#)[Upload recipes](#)

Manual 1

Sobre el Header no hay mucho que comentar, tenemos el logo del proyecto el cual nos permite dirigirnos al home de la aplicación, los diferentes enlaces a las distintas partes de la web, un botón el cual nos permite cambiar el estilo de la página a modo noche o viceversa y un icono de usuario o foto de perfil (dependiendo de si hemos iniciado sesión).

Home

Search



Manual 2

Para empezar, el primer método que nos encontramos en el home de la aplicación se trata de un buscador simple, usando como único filtro el nombre de la receta que queremos buscar. Explicaremos con más detalle el método de búsqueda en la página de listado de recetas, pero para mencionarlo de forma superficial este método usa el mismo que esa página.

Last recipes

[View more...](#)

Manual 3

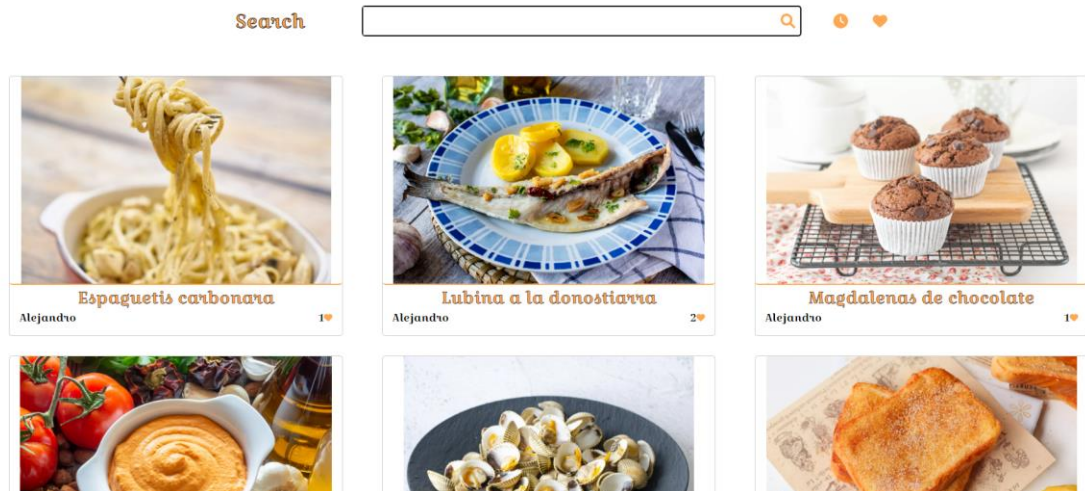
La siguiente sección de la página es el de últimas recetas, esta funcionalidad lo que hace es ofrecer al usuario las últimas 6 recetas que se han creado por los diferentes usuarios de la aplicación. Si queremos tener el listado de todas las recetas de las que disponemos hacemos click en view more.



Manual 4

En cuanto a este apartado tenemos una función en la parte front la cual se encarga de obtener el número de recetas que tenemos en la aplicación y redirigir al usuario a la página de detalles de una receta aleatoria.

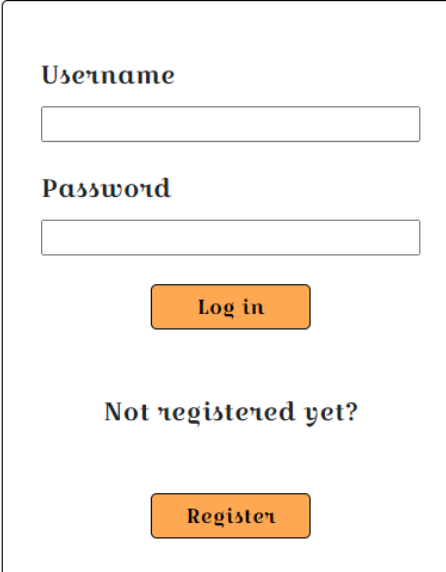
Recipes



Manual 5

Si entramos a la sección de recetas sin haber buscado nada desde el home llamaremos al punto de entrada /List de recetas, el cual nos devuelve todas las recetas de la base de datos según el filtrado que hayamos seleccionado (por defecto se ordena por fecha de creación), en cambio, si buscamos algo se hará una consulta de búsqueda a la base de datos con el nombre que hemos introducido y uno de los dos filtros que tenemos disponibles a la derecha del buscador (por defecto se usará el filtro de fecha, el cual antepone las recetas creadas recientemente, este filtro se usará también cuando busquemos una receta desde el home).

Log in / Register



A login and registration form with a light gray border. It contains two input fields: 'Username' and 'Password'. Below the password field is an orange 'Log in' button. Further down is the text 'Not registered yet?' followed by an orange 'Register' button.

Username

Password

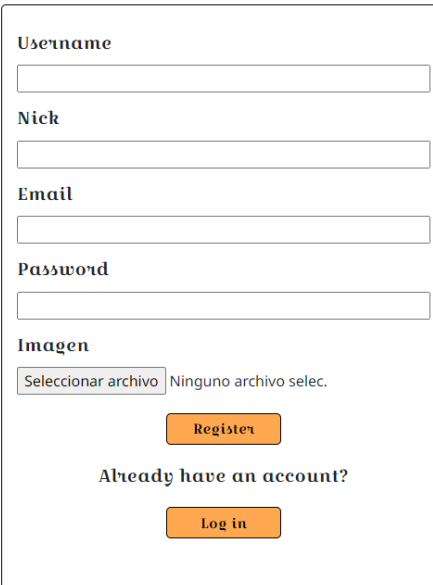
Log in

Not registered yet?

Register

Manual 6

Para tener acceso a todas las funcionalidades de la aplicación necesitaremos estar loggeados, para ello nos dirigimos a la página de log in haciendo click en el icono del Header, luego podremos introducir nuestras credenciales si ya hemos creado una cuenta.



A registration form with a light gray border. It contains five input fields: 'Username', 'Nick', 'Email', and 'Password'. Below the password field is a section for 'Imagen' with a file selection button labeled 'Seleccionar archivo' and the text 'Ninguno archivo selec.'. Below this is an orange 'Register' button. Further down is the text 'Already have an account?' followed by an orange 'Log in' button.

Username

Nick

Email

Password

Imagen

Seleccionar archivo Ninguno archivo selec.

Register

Already have an account?

Log in

Manual 7

Si no tenemos un usuario creado en la aplicación solo tendremos que introducir los datos necesarios (La imagen es opcional) y nos redirigirá al home una vez creado el usuario. Una vez dentro podremos ver que tenemos nuestra foto de perfil donde antes teníamos el icono de usuario (Si no hemos introducido imagen a la hora de registrarnos tendremos una foto por defecto).

Actualizar usuario

Username

Nick

Email

Password

Image
 Ninguno archivo selec.

Manual 8

Si damos click en nuestra foto de perfil tendremos acceso a un formulario en el que podremos cambiar los diferentes campos de nuestro usuario como cambiar el correo, nombre o poner una imagen diferente, también podremos hacer un log out o borrar el usuario.

Liked



HomeRecipesLikedUpload recipes



Salsa alioli
Paco 4♥



Arroz con leche
Ezequiel 2♥



Tarta de limón sin queso
Ezequiel 5♥

Manual 9

Si una vez iniciado sesión vamos a la sección de Liked podremos ver las recetas que hemos estado guardando, si no hemos iniciado sesión e intentamos entrar en esta página nos redirigirá al formulario de log in.

Upload recipe

Name	<input type="text"/>		
Description	<div>Enter description here...</div>		
Steps	<div>Enter steps here...</div>		
Ingredients	<input type="text"/>	Quantity	<input data-bbox="1141 786 1177 822" type="button" value="+"/>
Image	<div>Seleccionar archivo Ninguno archivo selec.</div>		
Video	<div>Seleccionar archivo Ninguno archivo selec.</div>		
<div>Save</div>			

Manual 10

A continuación, tenemos el formulario para subir recetas, en este introduciremos el nombre, descripción y pasos de manera normal introduciendo el texto, pero en cuanto a los ingredientes según el texto que introduzcamos se hará una petición al servidor para saber si ese ingrediente ya está creado, luego introducimos la cantidad necesaria y añadimos el ingrediente a la lista con el botón que encontramos a la derecha (si el ingrediente no está creado al pulsar el botón se crea en la base de datos y se añade a la lista), adjuntamos una imagen y un video de forma opcional y guardamos la receta. Una vez creada nos redirigirá a la receta ya creada.

Receta



Tarta de limón sin queso

La tarta de limón sin queso es una elaboración deliciosa, fresquita y sencilla de preparar. Un pastel de limón ideal para después de una comida, ya que es ligero y con un sabor agradable. Además, esta receta que os presentamos no contiene queso, se elabora con yogur y con nata. Por otro lado, no necesita horno y es ideal para el verano. Sin embargo, aunque no necesite horno, si es necesario 3-4 horas para que esté la tarta fría, incluso, el día siguiente está más buena.

Ingredientes

- Galletas - 150gr
- Mantequilla - 80gr
- Yogur natural - 3
- Nata - 200mm
- Limón - 2
- Hoja gelatina - 6

Direcciones

Para empezar con la receta de tarta de limón sin queso, primero debes preparar la base de la tarta. Para ello, tritura las galletas y derrite la mantequilla. Incorpora la mantequilla junto con las galletas y amasa hasta que los dos ingredientes queden bien integrados. Cubre el fondo de un molde de 20 cm con la masa de galletas. A continuación, aplasta bien con ayuda de una cuchara para que quede bien compacta y uniforme la base. Mientras, reserva en la nevera. En un bol con agua fría pon en remojo las hojas de gelatina. Exprime los limones y, si te gusta más sabor a limón, puedes añadir también la ralladura de los mismos. En un bol añade el zumo de los 2 limones, el azúcar y dos cucharadas de agua. A continuación introduce en el microondas unos segundos para que esté

Ezequiel

5 ♥

Manual 11

Cuando entramos en la página de una receta tendremos al principio de esta la imagen que hemos introducido en el formulario anterior, si disponemos de video se sustituirá la imagen por este, luego tendremos el nombre de la receta y del usuario junto con el número de likes, podremos hacer o quitar like a la receta (si la receta es del mismo usuario tendrá por defecto like, pero no podremos eliminar el like ya que es necesario), si somos administradores de la aplicación o la receta la hemos creado nosotros tendremos un icono de papelera para eliminar la receta.

Un poco más abajo tendremos la descripción, ingredientes y pasos a seguir de la receta.

Comments

Image

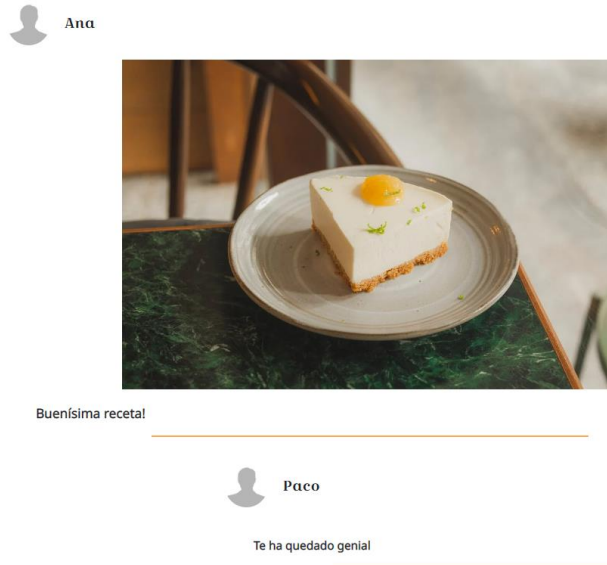
Seleccionar archivo Ninguno archivo selec.

Save

Insert your comment here...

Manual 12

Si seguimos bajando podremos ver el formulario de comentario, tendremos que haber iniciado sesión para dejar un comentario en cualquier receta. Podremos incluir de forma opcional una imagen al comentario.



Manual 13

Más abajo tendremos la lista de comentarios de la receta junto con los usuarios que lo crearon, como podemos ver en la imagen hay un comentario más pegado a la parte derecha que los demás, esto indica que el comentario es un comentario hijo del que está más arriba (es una forma de responder con un comentario a otro comentario). Debajo de cada comentario tendremos otro formulario para poder contestar a cualquier comentario.

Conclusiones

Comparado con la idea original del proyecto estoy contento por haber podido realizar un proyecto sólido de principio a fin junto con su despliegue, si es cierto que hay ideas del principio que se han ido quedado por el camino ya sea debido a tiempo o problemas con el planteamiento en el código de estas. Algunas de las ideas que se han quedado en el tintero son la búsqueda filtrada por los tags que disponen cada receta, búsqueda con diferentes ingredientes (Ej. Buscar recetas con queso y carne picada) o modificar comentarios creados. Aunque los puntos mencionados anteriormente se podrán ir implementando con el tiempo.

A pesar de estos puntos no implementados pienso que la aplicación sigue siendo sólida en cuanto a funcionamiento ya que puedes crear de manera sencilla recetas, buscar por filtros como nombre, tiempo de subida o número de likes (lo cual lo hace intuitivo y rápido para encontrar las recetas necesarias), guardar recetas para más adelante con el botón de like, modificar los datos de tu cuenta o borrarla, etc...

En resumen, todavía quedan puntos a mejorar en mi proyecto, pero creo que igualmente en el punto que se encuentra la aplicación es bastante funcional.

Índice de tablas e imágenes

Descripción 1.....	4
Descripción 2.....	5
Descripción 3.....	5
Descripción 4.....	6
Descripción 5.....	6
Descripción 6.....	7
Descripción 7.....	7
Descripción 8.....	8
Descripción 9.....	9
Descripción 10.....	9
Descripción 11.....	9
Descripción 12.....	10
Descripción 13.....	10
Descripción 14.....	11
Descripción 15.....	11
Descripción 16.....	11
Descripción 17.....	12
Descripción 18.....	12
Descripción 19.....	13
Instalacion 1	13
Instalacion 2	14
Instalacion 3	14
Instalacion 4	15
Instalacion 5	15
Instalacion 6	15
Instalacion 7	15
Instalacion 8	16
Instalacion 9	16
Guia estilos 1	17
Guia estilos 2.....	18
Guia estilos 3.....	18
Guia estilos 4.....	18
Wireframe - móvil 1	19
Wireframe - móvil 2	19
Wireframe - móvil 3	19
Mockup - móvil 1.....	20
Mockup - móvil 2.....	20
Mockup - móvil 3.....	20
Mockup - móvil 4.....	21
Mockup - móvil 5.....	21

Mockup - móvil 6.....	21
Wireframe - Desktop 1.....	22
Wireframe - Desktop 2.....	23
Wireframe - Desktop 3.....	23
Mockup - Desktop 1	24
Mockup - Desktop 2	25
Mockup - Desktop 3	25
Mockup - Desktop 4	26
Mockup - Desktop 5	27
Mockup - Desktop 6	27
Diseño 1.....	28
Diseño 2.....	29
Despliegue 1	32
Despliegue 2	33
Despliegue 3	33
Despliegue 4	34
Despliegue 5	34
Manual 1	35
Manual 2	35
Manual 3	35
Manual 4	36
Manual 5	36
Manual 6	37
Manual 7	37
Manual 8	38
Manual 9	38
Manual 10	39
Manual 11	40
Manual 12	40
Manual 13	41

Bibliografía

- [Documentación SQLAlchemy](#)
- [Documentación Flask](#)
- [Documentación Marshmallow](#)
- [Documentación Docker](#)
- [Documentación React](#)
- [Stackoverflow](#)
- [Fotos y videos de stock](#)
- [Recetas para el seeder](#)