Vitae

Autores

Bolu Li Yuan 718197 Álvaro García García 741417 Héctor Toral Pallás 798095

Enlaces del proyecto

Link de despliegue: vitop.xyz >Credenciales: >- Usuario: hi@vitop.xyz >- Contraseña: admin

Link de la documentación en swagger: api-docs

¿Qué es Vitop?

Vitop es una aplicación web cuyo principal objetivo es ayudar a las personas a llevar una dieta sana y equilibrada. Para ello, la app ofrece la busqueda y almacenamiento de recetas con las que poder generar menus personalizados. Además, aporta información al usuario mediante distintas métricas de las que hablaremos posteriormente. Esta aplicación no busca solo mantener a los usuarios sanos, sino que también tiene como objetivo facilitar y simplificar algunas de las tareas del día a día.

Resumen

Vitop by Vitae es un lugar donde buscar y almacenar recetas y menús. En función de tu peso, altura y dieta, se ofrecerán un tipo u otro de dietas con el objetivo de satisfacer de forma personalizada las necesidades nutricionales de los usuarios.

Esta aplicación busca no solo mantener a los usuarios sanos, sino que también tiene como objetivo facilitar y simplificar algunas tareas del día a día, algunas de estas funcionalidades son la posibilidad de ver en un calendario de forma organizada nuestros próximos menús, o la función de generar la lista de la compra pensando en nuestras próximas comidas.

Analisis de requisitos

En las etapas inciales del proyecto se realizó un pequeño analisis de cuales podrían ser los principales problemas de las personas de entre 20-50 años. Algunos de los mas relevantes fueron: - Falta de tiempo para planificar un menu familiar - Desconocimiento sobre cocina (recetas, ingredientes, etc) - Demasiados sitios donde buscar recetas - Llevar un registro del progreso - MAS

Propuestas similares

Existen varias aplicaciones que comparten funcionalidad con vitop, pero ninguna comparte todas sus funcionalidades, ni su portabilidad (la mayoría de las alternativas son para android)

Name	Lista de la compra	Metricas nutricionales	Consultar recetas	Comentarios de las recetas	Price
Nootric	❌	✅	✅	❌	❌
Mealime	✅	❌	✅	❌	❌
LifeSum	✅	✅	✅	❌	❌
Dommuss	✅	❌	✅	❌	❌

Desarrollo

Para comenzar el desarrollo se decidio seguir una estrategía basada en la funcionalidad del usuario es decir, primero se realizarían un mockup de las vistas que estos tendrían para posteriormente diseñar los modelos de datos necesarios para almacenar la información a servir e implementar las funcionalidades justas para cubrir las necesidades de estos.

Diseño de funcionalidad

Para el diseño de la funcionalidad alojada en el backend se decidió que esta debería estar organizada en base al sujeto paciente de la acción, por ejemplo una funcionalidad que añade recetas debería estar separada de la que permite al usuario hacer login, de esta forma tendríamos más ficheros pero más pequeños.

Arquitectura de alto nivel

Utilizamos una arquitectura MERN. Mongo para la base de datos el cual se comunica con Express mediante un middleware(Mongoose), esta a su vez crea una API rest para el frontend. La particularidad que tiene el frontend es que además de utilizar solo react, utilizamos next js como segundo backend de este. Uno de los motivos de esta decisión fue que debido a algunos problemas durante el desarrollo hubo que buscar alternativas para restringir el acceso a algunas rutas que requisieran un token de sesión, otro de los motivos fue que next js tiene un componente(Image) para la carga de imágenes en el cliente, que mejora la experiencia de usuario, el motivo de la utilización de este componente es que dependiendo del tamaño del dispositivo con el que el usuario se conecte a la aplicación, el componente realizará un 'resize' de la imagen de forma automática, además de permitir poder cambiar la calidad con la que se muestran, de tal manera que la carga de imágenes se realizará de forma más rápida. Esto tiene

sentido en nuestra aplicación ya que el 90% o más del contenido de nuestra app son imágenes.

Implementación

Backend

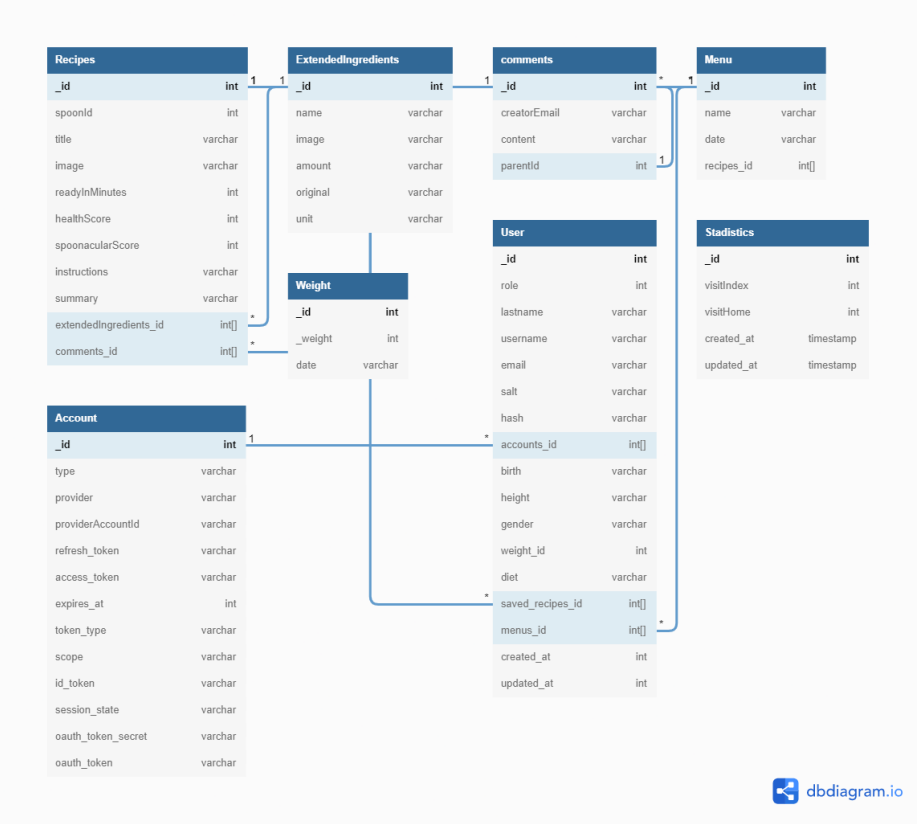
Para la implementación de backend utilizamos las siguientes herramientas: Lint y Husky para hacer un código mas unificado y limpio para el proyecto, github para el control de versiones y visual studio code de IDE. La estructura de ficheros es el siguiente en la raiz guardamos solo la app.js y otros ficheros autogenerados(package.json,travis,.env). Como decisión de implementación se optó por crear una carpeta "lib" donde meter los archivos que no encajaban en los controladores (por ejemplo el que embebe las funciones que se comunican con la api de spoonacular). Contenidos de cada carpeta:

- Controllers: Gestiona las llamadas a la base de datos y se compone de :
  - Admin.js: Encargado de devolver las estadísticas
  - Comment.js: Gestiona los comentarios de las recetas,
  - Inventory.js: Gestiona todo lo que tiene que ver con la interacción del usuario registrado y las recetas(guardar y eliminar recetas del storage) y menu (guardar y eliminar menús)
  - Newsletter.js: Suscribirse al correo
  - Recipes.js: Guardar,consultar y eliminar recetas
  - Recovery.js: Recuperar la contraseña
  - User.js: Autentificación , conexiones de cuentas, creación modificación de los datos del usuario
- Routes: Las rutas de la Rest API
  - Inventory.js : Provee al front las consultas de inventory y recipes (Se necesita token para acceder a ellas)
  - Newsletter.js : Provee al front un punto de acceso para registrar usuarios en una lista de correo de mailchimp.
  - Recovery.js : Provee al front las consultas de recovery
  - User.js: Provee al front las consultas de User, Comment y Admin
- Models: Se compone de los esquemas de Mongoose y la inicialización de la base de datos,se explica mas adelante en la sección de Modelo de datos
- Lib: Otros archivos útiles
  - Spoonacular.js: Aquí están las funciones que comunican el backend con el API de spoonacular
  - Http.js: Alberga las funciones que manejan métodos no implementados
  - Auth.js: Funciones para comprobar la autenticación.
  - emails.js: En este archivo se encuentran las interacciones por correo electrónico, como el correo de recuperación de contraseña.
  - dates.js: Contiene funciones de procesamiento y parseo de fechas.

Dicho esto utilizamos middlewares tales como: * next-auth para la gestion de

autentificaciones. * Winston como herramienta para hacer loggers. * Nodemailer para mandar un enlace único al usuario con el que pueda cambiar su contraseña. * Mongoose para la consulta de base de datos. * JsonWebToken para la seguridad de los endpoint.

Modelo de datos El modelo de datos es el siguiente: - Users: Que contienen todo lo relevante a los datos de un usuario: - Datos personales:(lastname,username,email,birth,height,gender,diet) - Datos para la administracion de la cuenta: - Array de Accounts: Cuentas sociales que se asocian al usuario - salt,hash - Datos que componen el inventory del usuario: - saved_recipes: Se guardan las _id de mongoDB de las recetas,en principio se decidio guardarlo como subdocumento,pero al ser un documento que podrian modificar todos los usuarios ademas de guardarlos,lo mas sencillo para mantener la consisntecia en la base datos es guardarlo como object id . - menus_id: Se compone de una fecha un nombre y varias id de recetas,se guardo como object id por la misma razon anterior - weight: Es un subdocumento que se compone del peso del usuario y una fecha,el cual esto servira para las graficas del home del usuario. - Accounts: Son las cuentas sociales que puede tener un usuario (Twitter y/o Google) - Recipes: Es un esquema creado a partir del propio esquema de las recetas de nuestra api publica (Spoonacular) salvo que añadimos un par de campos adicionales: - Comentarios: comentarios que pueden escribir los usuarios. - nutrientes: Los macronutrientes de la receta - Stadistics: Aquí guardamos en forma de array un elemento por día, los datos relevantes para sacar una visión general del tráfico que tenemos en la app.

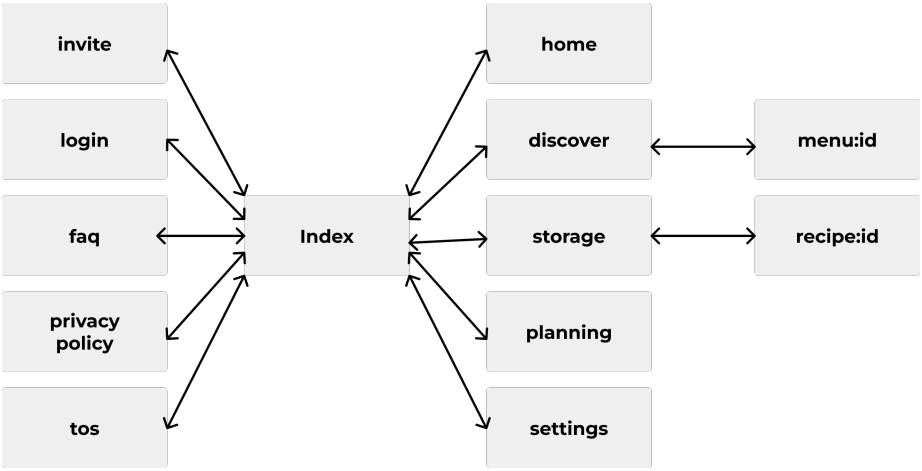


Frontend

Apariencia Para el diseño de la UI se decidió seguir un layout simple y eficáz, que le simplificara la vida al usuario, separando así las parte de acceso con las diferentes paginas quedando así esta plantilla:

Finalmente, a la hora de realizar la implementación, fue cambiando ligeramente pero siempre siendo fiel al diseño original.

Modelo de navegación Como modelo de navegación tenemos las distintas rutas que se muestran en la parte inferior. Las de la izquierda de index son rutas visibles a todos los usuarios y las de la parte derecha son especificas para usuarios registrados. Faltaria una que estaría aislada del resto que sería la del administrador cuyo enlace seria `/admin`.



Analíticas

Usuario

Con los parámetros dados por el usuario junto a su menú le ofrecemos al usuario como de completa es su dieta. Junto a las proporciones de nutrientes que consume. También se le proporcionan gráficos de la fluctuación de su IMC conforme al tiempo.

Administrador

Por la parte de administrador consideramos los datos mas importantes son la afluencia de usuarios y los usuarios registrados, en un futuro pensamos implementar más funcionalidades que permitar medir el número de usuarios conectados y cantidad de consultas, para que el administrador tenga una mejor visión de la utilización de la app.

Despliegue

Backend

Para el despliegue del backend se optó por utilizar el PaaS de Heroku, para el despliegue solo hubo que conectar el repositorio del backend con la cuenta de heroku, configurar las variables de entorno y darle a deploy.

Frontend

Por otro lado, la parte del frontend tuvo más problemas, la parte del servidor de next donde estabamos realizando las primeras consultas para precargar la página con datos estaba "configurada" para acceder a las cookies de sesion de next-auth con el nombre de la cookie de desarrollo por lo que tuvimos que cambiar el nombre al que utilizaban las cookies seguras en este paquete, además, tuvimos otra serie de problemas debido a las promesas de js ya que el servidor de despliegue que escogimos para el frontend nos estaba dando errores con el control de errores de promesas al usar un paquete llamado react hot toast. Finalmente nos dimos cuenta de que no era problema del control de errores sino de que el paquete estaba recibiendo incorrectamente la promesa.

Base de Datos

Problemas

Problemas encontrados

Dificultades de conexion entre next js y express js por problemas de cors. Dificultad a la hora de desplegar,queriamos añadir un subdominio para el backend pero al final no se pudo hacer. Problemas con el merge en github deshicimos muchas horas de trabajo del backend. Muchos bugs debido a los asyncs y awaits.

Problemas potenciales

Como se comenta más adelante , un paquete de npm del front y uno del back presentan vulnerabilidades por ser versiones antiguas, de todas formas, estas vulnerabilidades aún no han sido explotadas. Existe también el problema potencial de usos limitados tanto del captcha como de la propia Api de Spoonacular. El captcha tiene un máximo de un millón de usos diarios por lo que no es un problema. Para Spoonacular, el límite son 150 recetas al día, para optimizar este valor, cada vez que hacemos una consulta de información, traemos esa información a la base de datos, de forma que si se vuelve a necesitar una información ya consultada, se trae de la base de datos.

Distribución de tiempo

En primer lugar realizamos una división de tareas en las que se decidió que integrante del grupo realizaría cada tarea, Héctor se encargaría del frontend y la integración con del backend, mientras Álvaro y Bolu realizarían el backend. Tras varias semanas de desarrollo, y tener ambas partes "acabadas" se procedió a la conexión de ambas partes donde se descubrió que la mayoría de los endpoints del backend estaban fallando. Desde ese momento, Héctor decidió buscar una alternativa que consistio en descartar el uso de passport ya que su configuración habia sido incorrecta e instalar next-auth que soluciono el problema en pocos minutos, además de esto, algunos de los filtros y lógica que se habían pensado implementar en el backend se decidieron pasar al frontend puesto que cambiar la estructura del backend estaba resultando complicada.

Conclusiones

Resultado Una app completa y sencilla con un diseño limpio hecha con Tailwind, optimizada y lista para albergar grandes cantidades de usuarios gracias a las decisiones tecnologicas escalables (MERN stack)

Valoraciones personales

Álvaro

Llegué a esta asignatura con poca experiencia en tecnologías web, y a lo largo de toda la asignatura he aprendido muchos contenidos que me han permitido ayudar a mi equipo a sacar adelante este trabajo. Considero que el grupo ha trabajado mucho y pese a ser uno menos, el resultado creo que nada tendrá que envidiar a otros grupos.

Héctor

Al haber realizado algún que otro proyeto pequeño ya habia tenido la posibilidad de trabajar con estas tecnologías. Lo que si que me llevo es la visión de otros

compañeros de cara a afrontar los diferentes trabajos realizados como alguna que otra herramienta para mejorar mis desarrollos.

Bolu

No me gustaba javascript al principio ya que estoy acostumbrado a un lenguaje fuertemente tipado como Java,sin embargo,a lo largo del proyecto pude acostumbrarme no solo a una IDE que da menos feedback del codigo que escribes(aun con Eslint era facil cometer errores) sino que aprendi a escribir codigo mas simple y corto. Tambien valoro el aprendizaje que me he llevado al usar MongoDB,haciendo las consultas mucho mas faciles,rapidas y potentes una vez los dominas(sobretodo joins).

Anexos (Extras)

Loggin con herramientas externas

A traves de nextAuth hemos implementado login con Google y con Twitter, de forma que si es una cuenta nueva, crea una cuenta con el correo asociado al servicio. Por otro lado si la cuenta de Vitop ya esta creada, también es posible asociarla con la de un servicio con el mismo correo electrónico.

Analizador de cógido estático

Hemos utilizado Snyk como herramienta para analizar el codigo de forma estática en busca de vulenerabilidades. Esta programado para que regularmente haga test. Actualmente existe una vulverabilidad media en el frontend y otra en el backend, ambas por el uso de paquetes en versiones obsoletas, no es posible solucionarlo ya que son dependencias de otros paquetes, en cualquier caso, estas vulnerabilidades aun no se ha encontrado la forma de explotarlas.

Captcha

Como herramienta para comprobar que el usuario es un humano hemos utilizado la herramienta reCaptcha de google. Está en la pantalla de login, impidiento al usuario acceder a no ser que haya demostrado no ser un robot. Se ha optado por esta herramienta dada su sencillez y facilidad a la hora de utilizar.

Google Analytics

Realizamos la conexión de google analytics con la aplicación mediante un script que se ejecuta en la “raiz” del proyecto (/pages/_app.js). Este detecta los cambios de la ruta del navegador y va guardando los nuevos lugares de acceso y el número de accesos permitiendo así realizar estudios sobre el público que está tiendo la página

Mailchimp

También realizamos la integración con un servicio para generar embudos de ventas para mandar correos electrónicos ya sea de spam o contenido por suscripción