Esta clase va a ser

- grabada

Clase 23. PROGRAMACIÓN BACKEND

Práctica integradora & Product Cloud: Despliegue de nuestra aplicación

CODERHOUSE

Temario

22 - Parte II

Frameworks de desarrollo: Nestjs (Parte II)

- ✓ Conexión de Nestjs a Mongo
- ✓ Autenticación con JWT

23 - Parte I

Segunda Práctica Integradora

- ✓ Skills
- ✓ Práctica integradora

23 - Parte II

Product Cloud: Despliegue de nuestro aplicativo

- ✓ Despliegue de nuestro aplicativo
- ✓ Configuración de pipeline en Railway.app

Objetivos de la clase - Parte I

- Hacer una integración práctica de todos los conceptos vistos hasta el momento, bajo el desarrollo de un proyecto paralelo a nuestro proyecto final.

¿Cómo organizar la clase?

Estructura general de la clase	Tiempo de dedicación	Enfoques
Skills	10 min	✓ No detenerse a explicar cada skill ✓ Utilízalo como motivador, no como stopper.
Práctica integradora	1hr 20 min (contemplando break a tu elección)	✓ Recuerda que estás retomando un proyecto de práctica integradora previo. No pierdas tiempo repasando cosas de la práctica integradora anterior. ✓ Esta práctica integradora supone un caso muy particular, y es que es de carácter más relajado, ya que las cosas a integrar son bastante sencillas, cosas como logging y documentación, haciendo que el resto de los temas sea bastante subjetivo según sea el caso.
Presentación del desafío	10 min	
Espacio de dudas y consultas	15 min	✓ Puedes traer dudas con respecto a contenidos vistos en el curso o bien relacionadas con el terreno profesional.

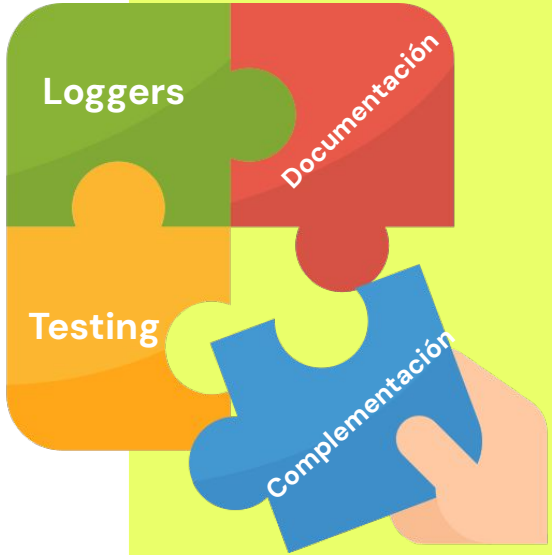
¿Qué estamos por ver?

CODERHOUSE

Práctica integradora

La mejor forma de repasar los temas vistos hasta el momento, es hacer un repaso integrado de todos los elementos.

Si bien es correcto repasar el código parte por parte, es importante que como desarrolladores comencemos a trabajar en nuestra **lógica de integración**, es decir, tenemos que tener siempre contemplado el cómo vamos a juntar todo lo aprendido, para tener un proyecto sólido



CODERHOUSE

Elementos a integrar

Elementos a integrar

En el desarrollo de esta práctica de integración, repasarás y conectarás los siguientes conceptos:

- ✓ Logging
- ✓ Documentación
- ✓ Complementación

CODERHOUSE

Skills para esta práctica integradora

Skills para Logging

- ✓ Comprender la importancia de utilizar un logger
- ✓ Comprender el uso de Winston Logger y aplicarlo
- ✓ Entender los diferentes tipos de transportes
- ✓ Entender sobre los niveles de logging
- ✓ Configurar nuestros propios niveles de logging

CODERHOUSE

Skills para documentación

- ✓ Comprender la importancia de documentar
- ✓ Comprender el uso de Swagger para documentación
- ✓ Realizar la Swaggerización por archivos de cada Módulo
- ✓ Comprender sobre los elementos que compone un módulo Swaggerizado (schemas, inputs, requestBodies, responses, etc)

CODERHOUSE

Skills para testing

- ✓ Comprender sobre módulos de Testing
- ✓ Conocimientos de realización de Testing unitario
- ✓ Conocimiento sobre Test de integración
- ✓ Uso de Mocha
- ✓ Uso de Chai
- ✓ Uso de SuperTest

CODERHOUSE

Importante

La práctica hace al maestro... si consideras que aún necesitas practicar algún aspecto de los skills mencionados, siempre puedes repasar las clases previas. Sin embargo, esto no significa que no puedas avanzar.

¡Esta práctica integradora te ayudará a sentirte más seguro en estos temas!

Desarrollo de esta práctica integradora

1. Aplicación de un logger que utilice múltiples transportes
2. Documentación elemental del proyecto en cuanto a sus elementos principales
3. Testeo de funcionalidades principales de los módulos por separado e integrados.

¡Comenzamos!

Práctica de integración sobre tu ecommerce

CODERHOUSE

DESAFÍO COMPLEMENTARIO

Consigna

Con base en el proyecto que venimos desarrollando, toca solidificar algunos procesos

Aspectos a incluir

- ✓ Mover la ruta suelta ***/api/users/premium/:uid*** a un router específico para usuarios en /api/users/

- ✓ Modificar el modelo de User para que cuente con una nueva propiedad “documents” el cual será un array que contenga los objetos con las siguientes propiedades
  - name: String (Nombre del documento).
  - reference: String (link al documento).

  No es necesario crear un nuevo modelo de Mongoose para éste.
- ✓ Además, agregar una propiedad al usuario llamada “last_connection”, la cual deberá modificarse cada vez que el usuario realice un proceso de login y logout

DESAFÍO COMPLEMENTARIO

Aspectos a incluir

- ✓ Crear un endpoint en el router de usuarios ***api/users/:uid/documents*** con el método POST que permita subir uno o múltiples archivos. Utilizar el middleware de Multer para poder recibir los documentos que se carguen y actualizar en el usuario su status para hacer saber que ya subió algún documento en particular.

- ✓ El middleware de multer deberá estar modificado para que pueda guardar en diferentes carpetas los diferentes archivos que se suban.
  - Si se sube una imagen de perfil, deberá guardarlo en una carpeta ***profiles***, en caso de recibir la imagen de un producto, deberá guardarlo en una carpeta ***products***, mientras que ahora al cargar un documento, multer los guardará en una carpeta ***documents***.
- ✓ Modificar el endpoint ***/api/users/premium/:uid*** para que sólo actualice al usuario a premium si ya ha cargado los siguientes documentos:
  - Identificación, Comprobante de domicilio, Comprobante de estado de cuenta

DESAFÍO COMPLEMENTARIO

Aspectos a incluir

En caso de llamar al endpoint, si no se ha terminado de cargar la documentación, devolver un error indicando que el usuario no ha terminado de procesar su documentación.
(Sólo si quiere pasar de user a premium, no al revés)

Formato

- ✓ Link al repositorio de GitHub con el proyecto completo (no incluir node_modules).

Sugerencias

- ✓ Corrobora que los usuarios que hayan pasado a premium tengan mayores privilegios de acceso que un usuario normal.

¿Dudas, preguntas, consultas?

Break

¡30 minutos y volvemos!

Temario

23 - Parte I

Cuarta Práctica Integradora

- ✓ Skills
- ✓ Práctica integradora

23 - Parte II

Product Cloud: Despliegue de nuestro aplicativo

- ✓ Despliegue de nuestro aplicativo
- ✓ Configuración de pipeline en Railway.app

24

Pasarelas de pago

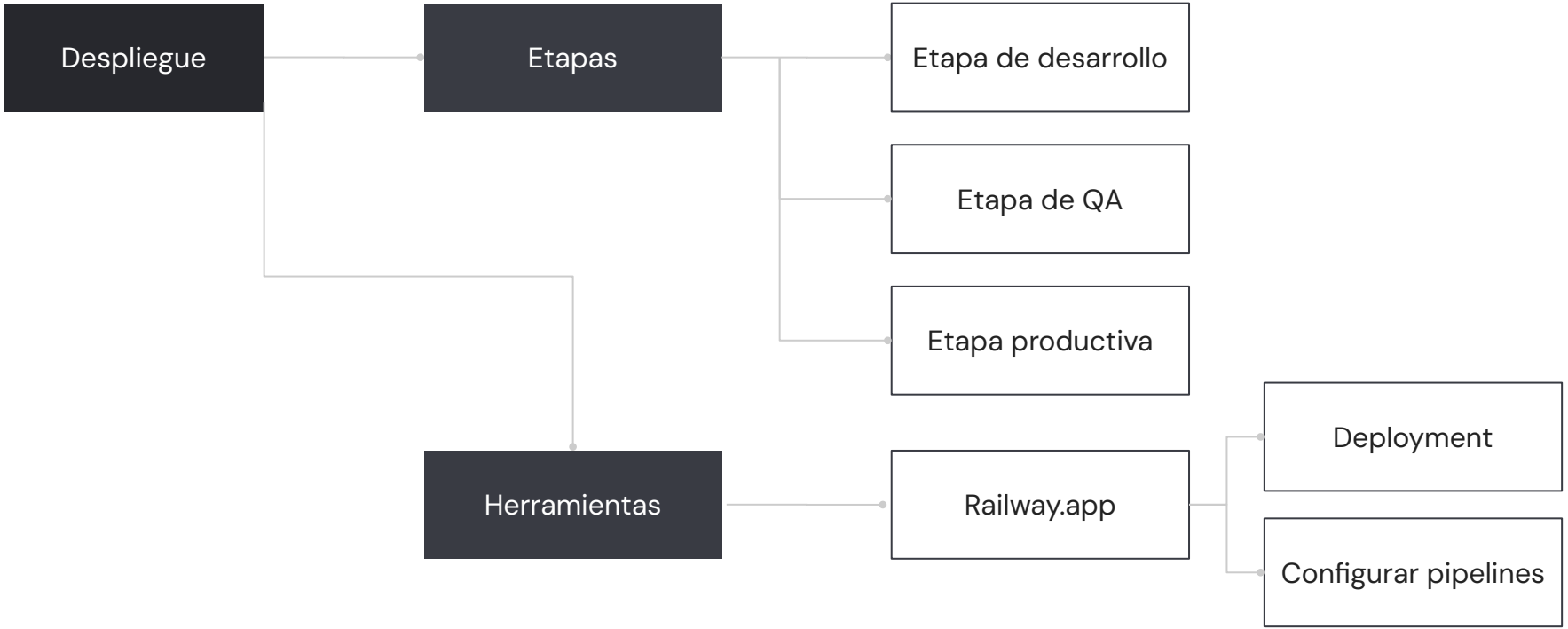
- ✓ Cerramos nuestra app

CODERHOUSE

Objetivos de la clase - Parte II

- Comprender sobre el proceso de deploy de un proyecto
- Manejar distintas etapas de desarrollo de aplicación

MAPA DE CONCEPTOS



Despliegue de nuestro aplicativo

CODERHOUSE

¿Nuestro aplicativo está listo para ser desplegado?

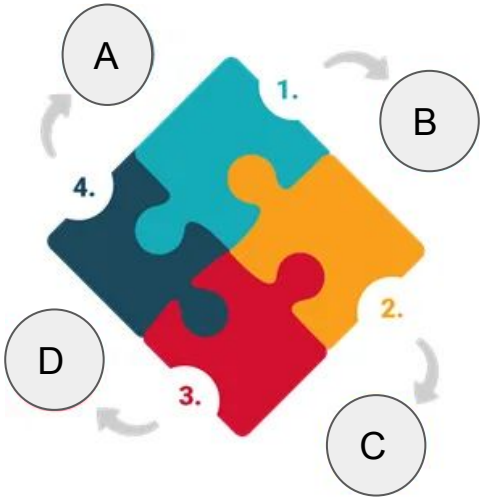
¿Cuándo podemos decir que nuestro aplicativo está finalizado? La realidad es que, dentro del mundo del desarrollo web, no podemos esperar a “terminar” un aplicativo para poder comenzar a desplegarlo.

La liberación de un proyecto es algo más complejo de lo que parece, y ocupa diferentes etapas que dependen de la empresa en la cual se está trabajando.

Etapas en el desarrollo de aplicaciones

Cuando se trata de sacar un proyecto al mercado, debe pasar por un conjunto de etapas, sobre las cuales se evalúan diferentes elementos para considerarse listo para el cliente. Las etapas varían enormemente según la forma de trabajo de cada empresa, por lo tanto, colocaremos en este punto las tres etapas principales por las cuales pasa un aplicativo:

- ✓ Etapa de desarrollo
- ✓ Etapa de QA
- ✓ Etapa productiva



CODERHOUSE

Etapa de desarrollo

Esta etapa está pensada para el flujo libre del desarrollador en cuanto a los cambios del aplicativo que éste pueda realizar.

En esta primera etapa, el desarrollador realiza y prueba su código en un **entorno demasiado amigable.** Se lo considera amigable, porque se desarrolla en el contexto que tiene en dicho momento el desarrollador, además de pasar por las pruebas elementales que el desarrollador pensó.

Sin embargo, cuando el desarrollador está listo para decir adiós a su aplicativo, tiene que enviarlo a las siguientes etapas

CODERHOUSE

Etapa de QA

Una vez que el desarrollador considera que su aplicativo está listo para ser presentado, este **“pasa”** la app a la siguiente etapa, que es QA.

Quality Assurance está pensado para poder ejecutar la feature del aplicativo que el desarrollador recién acaba de entregar. En su mayoría se tratan de **pruebas de blackbox** (pruebas sin saber del código, solo funcionalidad directa).

Si el aplicativo tiene el visto bueno por parte del equipo de QA, éste estará listo para salir a ser utilizado por el cliente. Es en este momento final cuando se dice que se hace un **despliegue a productivo**.

En caso de que no se dé el visto bueno, tendrá que regresar a etapa de desarrollo para hacer los respectivos cambios y presentar nuevamente.

Etapa productiva

Cuando QA autoriza liberación, entonces el aplicativo **“pasa”** a la etapa productiva. Éste es el punto final en el que el aplicativo vive en el mundo real, se enfrenta a clientes y comportamientos reales, con datos reales, con elementos cambiantes y con formas inusuales de usarse.

Cuando el aplicativo ha pasado por las etapas previas, el rango de error debería ser enormemente reducido, sin embargo, **el cliente es capaz de causar errores indescriptibles en el uso real**.

Es por ello que, siempre que se requiere algún cambio, algún bugfix, alguna nueva funcionalidad en general; se requiere regresar desde el punto 0 de las etapas, para volver a pasar por desarrollo, por QA, y volver a llegar aquí.

Importante

Como desarrollador, debemos limpiar la falsa idea de que tenemos al equipo de QA como “contendientes” solo porque son quienes enuncian los defectos de nuestros aplicativos.
Al final, recuerda que cada defecto que arreglemos es un peldaño más hacia un aplicativo más sólido.

¡Siempre se trata de un ganar-ganar!

¿Qué es “pasar” el aplicativo?

Cuando decimos que un aplicativo “pasa” de un lado a otro, hacemos referencia a cada uno de los stages que hayan sido definidos por el equipo. Un stage representa un checkpoint en el aplicativo, y habitualmente toma raíz de **una rama de GitHub**.

Así, el desarrollador puede estar trabajando en la rama **develop1,** la cual habrá sido derivada de la rama **development**, esta rama development más adelante se mezclará con la rama **QA**, y finalmente QA podrá mezclarse con **main**, cuando éste se considere listo.

Recuerda que al final, la rama main siempre se considerará como la rama principal, y nuestro objetivo como desarrolladores es que, los cambios que nosotros estamos haciendo desde **develop1**, siempre lleguen a estar presentes en **main**. ¡Entonces habremos llegado al entorno “productivo”!

Pero, ¿Qué tiene que ver esto de las ramas de git con el despliegue que se menciona en clase? Para ello primero tendremos que recordar la esencia de un **despliegue**

¿De qué se trata un despliegue?

¿Recuerdas el despliegue del chat que hicimos con **glitch.com**, o bien el seteo de pods que contenía el cluster de contenedores con **docker + kubernetes?** fueron despliegues del aplicativo.

Al finalizar estos procesos, **conseguimos que la aplicación estuviera ejecutándose en un servidor diferente**, logrando así que estuviera activo siempre que el servidor mantuviera vivo el proceso.

El servidor contaba ya con un dominio o un link particular que podía ser visitado por cualquier persona, consiguiendo así que el cliente pueda visualizar los cambios como si estuviera visitando una página real.

Herramientas de despliegue

El proceso de despliegue es **una de las partes más importantes dentro del flujo de vida de nuestro aplicativo**. Aquí decidiremos dónde terminará ejecutándose el producto final.

Hay múltiples alternativas para contar con un servidor en la nube. De manera que tenemos que elegir cuál puede sernos más útil acorde con nuestras necesidades.

Podemos optar por utilizar algún servicio robusto para aplicaciones de calibre enterprise (Azure, AWS, Google Cloud, etc.), o bien utilizar plataformas de calibre más simple.

Alternativas gratuitas

No es común que un desarrollador dé pauta a invertir en una plataforma para hacer despliegues más complejos, cuando recién está comenzando a entender el “qué es”. Por ello, se recomienda tomar alguna alternativa gratuita para que puedas experimentar cuál es el flujo de despliegue de un aplicativo.

Una vez que tengas control de los procesos de despliegue elementales, será buen momento para poder comenzar a utilizar plataformas más robustas. ¡Vamos paso a paso! En este caso, el aplicativo a utilizar será **Railway.app**

Deploy de nuestro aplicativo con Railway.app

Sobre Railway.app

Railway.app es una de las alternativas que podemos utilizar para realizar el despliegue de nuestra aplicación. Nos permitirá setear una app bastante sencilla para poder alojar nuestro código y ejecutarlo dentro.

Railway.app es de pago, sin embargo, nos brinda 5 USD cada mes para practicar nuestros despliegues. De manera que no tendremos de que preocuparnos.

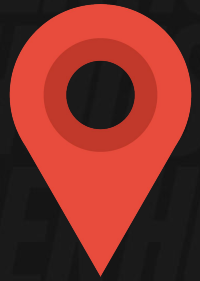
Antes de comenzar, necesitamos un aplicativo para desplegar

Seguramente sabemos de qué estamos hablando: **Adoptme**. El proyecto sobre el cual hemos estado trabajando este tiempo.

La idea será configurar el aplicativo hasta que esté listo para ser desplegado, posteriormente, se subirá a Railway.app para tener un link que permita que nuestro backend pueda ser visitado donde sea que se ocupe.

Lo primero será crear una cuenta de Railway.app. Además, sobre el proyecto de Adoptme, esta vez se te recomienda que hagas un forkeo del proyecto, ya que necesitarás manejar las branches de este proyecto.

Recuerda que clonar un proyecto es tener la referencia de ese proyecto, **forkearlo** es generar una copia personal del mismo.



Checkpoint: Espacio de preparación

Toma un tiempo para forkear y clonar el proyecto **aquí**. Revisa el código e instala las dependencias indicadas. Además, recuerda crear tu cuenta en Railway.app

Tiempo estimado: **10 minutos**

¡Comenzamos!

Con nuestro proyecto forkeado y nuestra cuenta de Railway.app creada, lo primero que haremos será comenzar a corroborar que los elementos fundamentales se encuentren en el aplicativo para poder ejecutarse arriba en Railway. Si no se cumplen estos criterios, se recomienda no comenzar nuestro deployment hasta que estén subsanados.

Los puntos a revisar son:

- ✓ **Variables de entorno seteadas**
- ✓ **Script de inicio**
- ✓ **Puerto de escucha**

Variables de entorno seteadas

Si inspeccionamos nuestro archivo app.js, notamos que están solicitándonos una URL de MONGO ATLAS, para poder conectar correctamente con la base de datos, sin embargo, sabemos que si subimos esta URL de manera limpia, Github se enojará por tener datos sensibles expuestos, por ello, colocaremos nuestra URL de la siguiente manera:

```
const connection = mongoose.connect(process.env.MONGO_URL)
```

También podríamos configurar nuestras variables de entorno del controlador de sessions (por toda la lógica de jwt y cookies). Sin embargo, de momento no es el objetivo principal, con la variable de entorno de la conexión a ATLAS estaremos bien.

Importante

¿Realmente necesitamos dotenv para nuestro proyecto?

Recuerda que, al desplegar nuestra aplicación en Railway, podremos colocar variables de entorno para dicho aplicativo, de manera que dotenv no será necesario para este ejemplo.
Si quisieramos ejecutar el proyecto sin dotenv, recuerda que siempre podemos ejecutar en modo **debug** nuestro aplicativo.

Script de inicio

¿Cuándo fue la última vez que corriste el script start en tu backend? Hemos estado tan acostumbrados a correr nuestro aplicativo a partir de **dev** (por el uso de nodemon), que en muchas ocasiones los desarrolladores llegan a olvidar colocar su script **start**

```
"scripts": {
  "start": "node src/app.js",
  "dev": "nodemon src/app.js",
```

Este script **start** es crucial para el servidor una vez deployado, debido a que éste ejecuta este comando, además, es importante que el script inicialice el aplicativo con node y no con nodemon, ya que la dependencia nodemon es algo externo que nosotros instalamos global, pero que el servidor no conoce.

Puerto de escucha

Por último, la forma en la que declaramos el puerto es importante también. Cuando ejecutamos el aplicativo desde nuestra computadora, abre el puerto que nosotros le indiquemos.

Contrario a esto, cuando nosotros desplegamos un aplicativo, el servidor es consciente de los puertos que tiene abiertos, por lo tanto, somos nosotros quienes deben acoplarse a este puerto, es por ello que la configuración de nuestro puerto debe verse así:

```
const PORT = process.env.PORT||8080;
```

¿Nuestro aplicativo ya cumplió con estos puntos?
¡Vamos a Railway!

Recuerda hacer push de los cambios realizados

Ya que hicimos un cambio de la variable de entorno de mongo, es importante que hagamos push de nuestros últimos cambios a la rama principal.

Sabiendo que Railway tomará el repositorio de GitHub como referencia de despliegue, es importante siempre tener actualizado el repositorio.

```
git add .
git commit -m "Added ENV VARIABLE for MONGO_URL"


git push origin main
```

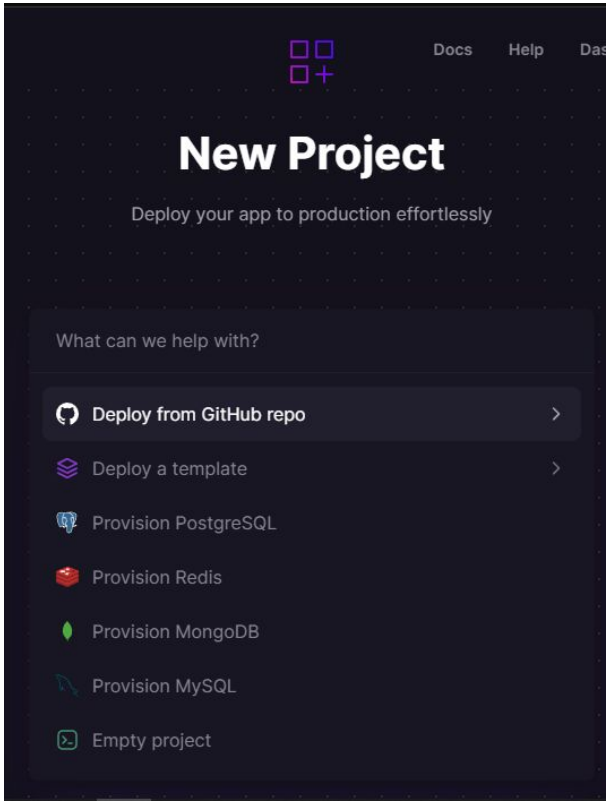
CODERHOUSE

Creando un proyecto

Ahora, desde Railway seleccionaremos la opción “New Project”

Nota cómo nos da la opción directamente de hacerlo desde Github, ahí es donde toma acción nuestro proyecto recién forkeado.

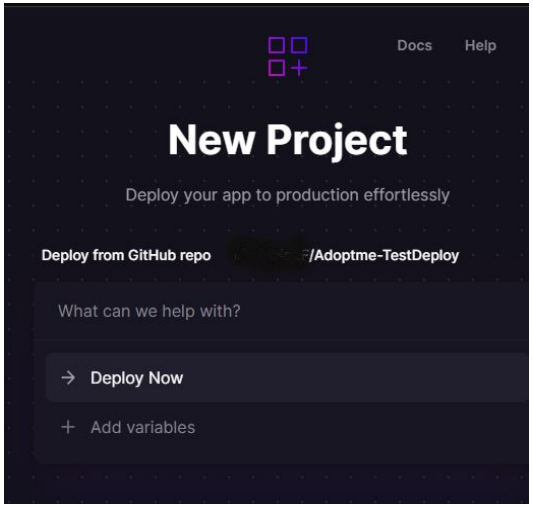
Daremos click a la opción de Github y seleccionaremos el repositorio que recién forkeamos.

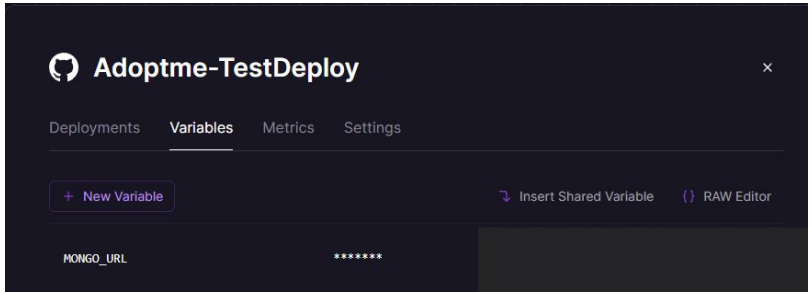


Configuración inicial

Antes de que se levante el aplicativo, Railway nos pregunta si tenemos intención de agregar variables de entorno, ¡Y nosotros tenemos una variable de entorno por agregar!

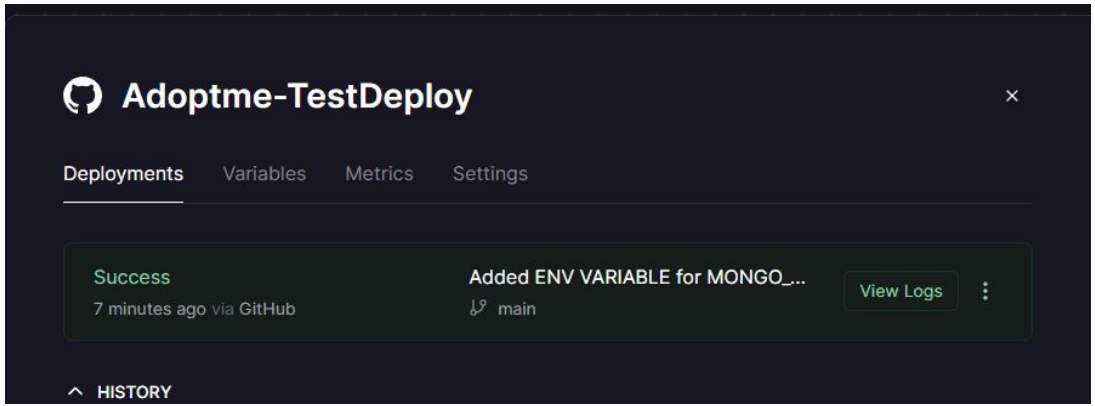
Daremos click en **Add variables**, y posteriormente añadiremos la MONGO_URL





Información del deploy

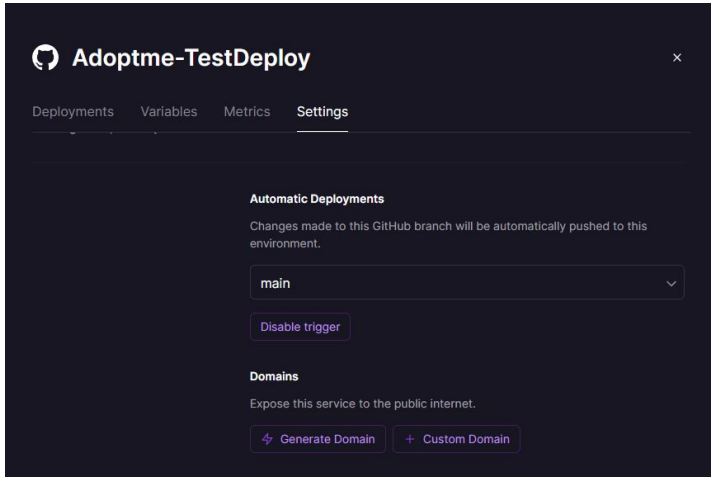
Podemos visualizar en un panel de información, en la pestaña de Deployments, el estatus actual del deploy que realizó la aplicación (Success en este caso). Siempre que un deploy salga mal, podemos dar click en **View Logs** para saber qué es lo que está saliendo mal

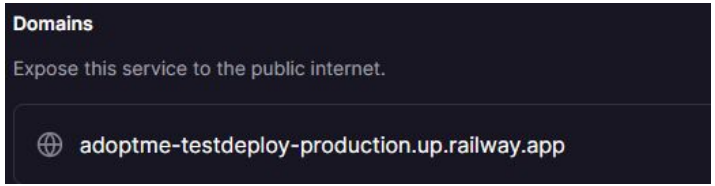


Generando un dominio

Por defecto, la aplicación se encuentra desplegada, pero no con un dominio especificado, para esto iremos a la pestaña de Settings. Luego, en el apartado **Domains**, seleccionaremos el botón **Generate Domain**

El link generado será el enlace para visitar nuestro backend.





Finalmente, utilizando nuestro backend

Si todos los pasos se siguieron correctamente, podemos visitar /api/users y obtendremos el resultado de usuarios de la base de datos de atlas que configuramos en nuestra MONGO_URL

Break

¡10 minutos y volvemos!

Configuración de pipeline en Railway.app

¿Qué es un pipeline?

Se entiende por pipeline a todo el flujo que comprende un proceso. Una característica importante de un pipeline es el uso de diferentes **stages**.

Estas stages están relacionadas con:

- ✓ **Diferentes entornos**: Cada entorno debe apuntar a diferentes bases de datos, por ejemplo, no es prudente mezclar mock users con usuarios reales de un entorno productivo.

- ✓ **Diferentes intenciones**: Una stage de desarrollo tiene la principal intención de corroborar que todas las ramas mergeadas de diferentes desarrolladores del equipo no hayan generado ningún conflicto funcional, sin embargo, una stage de QA no está interesada por la integración, sino directamente el carácter funcional del mismo.

Paso a paso

1

Desarrollo (preproductivo)

Probamos que todos los subdesarrollos de todos los devs estén correctamente integrados

2

QA (testing)

Corroboramos que los módulos no tengan defectos y mantengan una correcta funcionalidad.

3

Productivo

El aplicativo llega al cliente, está listo para poder ser utilizado en contextos reales.

Para trabajar con stages, también debemos pensar en branches

```
git checkout -b development

git push origin development
```

```
git checkout -b QualityAssurance

git push origin QualityAssurance
```

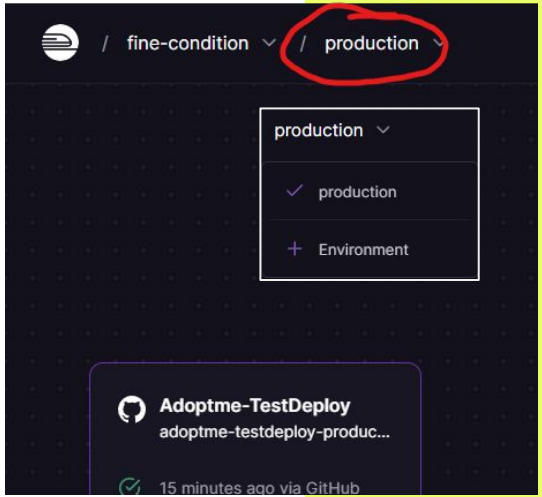
Hasta este momento solo hemos trabajado sobre la rama main del proyecto de Adoptme. La idea de haber forkeado el proyecto, también es que podamos separarlo en las branches que nosotros deseemos. Generaremos otras dos branches adicionales: development y QualityAssurance

Generaremos ambas ramas tanto en nuestro repositorio local, como en el repositorio remoto.

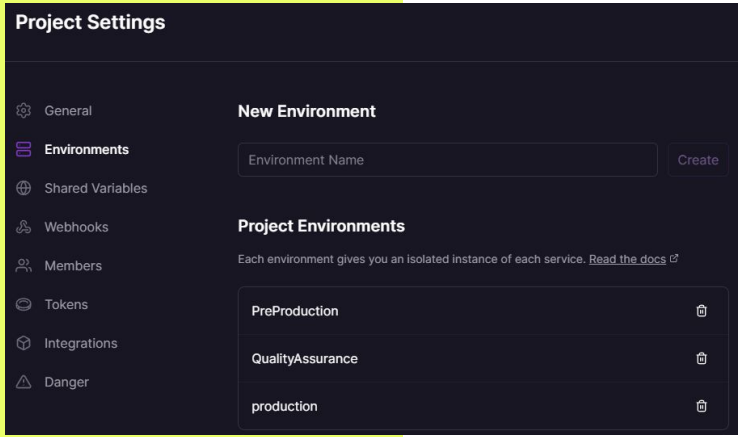
Configurando el pipeline en Railway.app

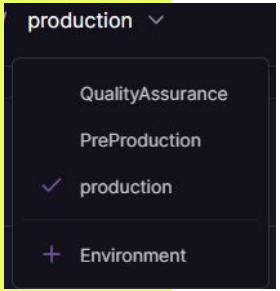
Al haber creado nuestra aplicación de Adoptme en Railway, esta se creó por defecto en un entorno de producción. Sin embargo, existe la forma de colocar múltiples entornos.

Daremos click en la opción “production” y seleccionaremos la opción **+ Environment**.



CODERHOUSE





Creación de entornos

Al dar click a la opción indicada, nos llevará a la configuración de environments, aquí es donde crearemos los dos environments diferentes: PreProduction y QualityAssurance

Ahora podremos visualizar los diferentes entornos disponibles.

CODERHOUSE

¡Cuidado, entorno inestable!

Al crear un entorno, el mismo intentará desplegarse automáticamente. Sin embargo:

- ✓ Sigue en la misma stage que el entorno principal, por lo que hay que cambiarlo para que escuche por uno diferente.
- ✓ Un entorno vuelve a crearse vacío en sus variables de entorno, por lo que hay que reconfigurar nuestra variable MONGO_URL nuevamente, apuntando a una base distinta.

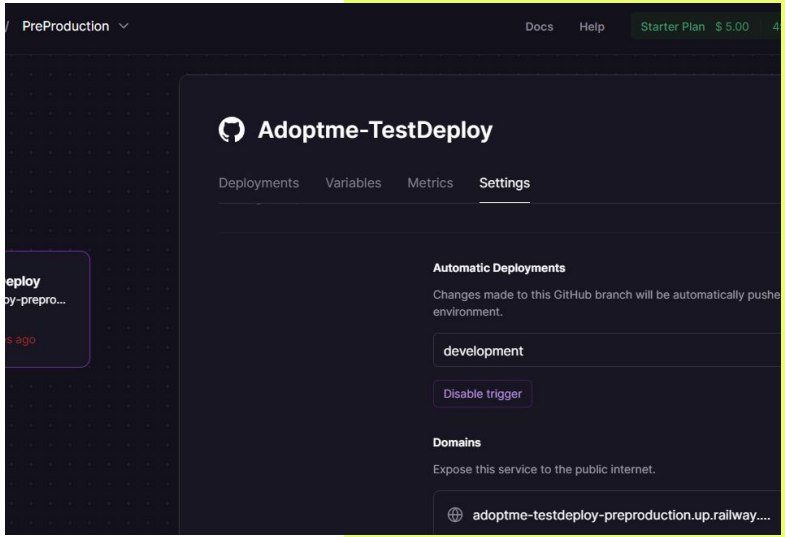
CODERHOUSE

¿Cómo reconfigurar los nuevos entornos?

Primero tenemos que ir a la configuración del respectivo entorno, y cambiaremos en ***Automatic Deployments***, la rama por la cual queremos escuchar en dicho entorno. Colocaremos la rama development.

Además volveremos a colocar la URL de la base de datos en MONGO_URL, apuntando a una base dev, con el fin de que no afecte al entorno productivo.

Realizar el mismo proceso para el entorno QualityAssurance.



CODERHOUSE

Hands on lab

En esta instancia de la clase **repasaremos** algunos de los conceptos vistos en clase con una aplicación

¿De qué manera?
El profesor demostrará cómo hacerlo y tú lo puedes ir replicando en tu computadora. Si surgen dudas las puedes compartir para resolverlas en conjunto de la mano de los tutores.

Tiempo estimado: **20 minutos**

CODERHOUSE

HANDS ON LAB

Creación de una vista a partir de un proceso de desarrollo completo.

Se nos ha solicitado crear vistas para el proyecto de Adoptme, el cual debe contar con:

- ✓ Una vista de bienvenida a partir de handlebars en la ruta base
- ✓ Una vista que permita visualizar a todos los usuarios registrados hasta el momento.
- ✓ Una vista que permita visualizar a todas las mascotas registradas hasta el momento.

HANDS ON LAB

Creación de una vista a partir de un proceso de desarrollo completo.

Comenzar desde la etapa de desarrollo, contemplado que los cambios en desarrollo se visualicen solo en el entorno de desarrollo, mas no en QA ni en Productivo.

Posteriormente, realizar un Pull Request entre development y QualityAssurance con el fin de que ahora los cambios se visualicen en el nuevo link, sin embargo, se necesitarán llenar los campos nuevamente, debido a que se estará apuntando a una nueva base de datos.

Finalmente, realizar el mismo proceso de QA a Productivo y corroborar que la vista coincida en productivo, llenar nuevamente la base con usuarios válidos para el entorno productivo.
Al final, tendremos tres entornos, con diferentes datos, pero funcionales por sí solos

¿Preguntas?

Muchas gracias.

Resumen
de la parte II

- ✓ Despliegue de aplicación
- ✓ Despliegue con Railway.app
- ✓ Pipeline de despliegue

Opina y valora esta clase

#DemocratizandoLaEducación