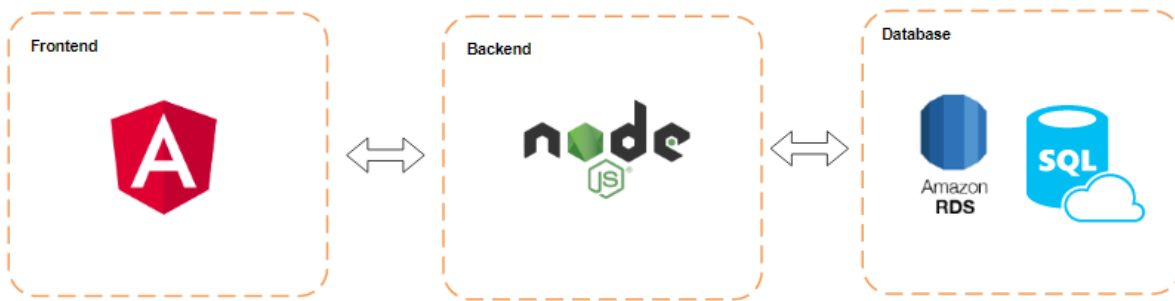


Manual Técnico

ONE BANK

Grupo # 1 - Abril 2021

Arquitectura de la aplicación:



Repositorio con Workflow:

<https://gitlab.com/cristelmedina122496/practica3-grupo1>

Casos de Prueba

Login

Se obtienen los datos “cuenta” y “password” y se mandan como parámetros al servidor para verificar que los datos se encuentren asociados a una cuenta registrada utilizando el siguiente endpoint:

Endpoint: <http://localhost:3000/users/users/login/cuenta&password>

“Debe permitir que un usuario ingrese”

Si los datos se encuentran registrados devuelve el resto de la información necesaria para ser mostrada y almacenada en el sitio web.

Registro

Se obtienen los datos del json que viene en el body del request, el cual contiene todos los datos del nuevo usuario a registrar ha ingresado en un formulario.

Endpoint: <http://localhost:3000/users/>

“Registro - Debe registrar un nuevo usuario”

Se valida que la respuesta sea el código único que identifica a cada usuario al momento de registrarse en la aplicación. De lo contrario devuelve false.

Perfil

Se obtienen los datos del local storage previamente cargados con el login al momento de loguearse el usuario.

Endpoint: <http://localhost:3000/users/login/14022020&123>

“Debe obtener los datos del usuario 6 para el perfil”

Se valida que la respuesta incluya el nombre, apellido, correo, dpi y número de cuenta correcto del usuario que está logueado, en esta prueba se utiliza el usuario 6, que es de un usuario previamente registrado en el sistema. Además se verifica que el estado de la respuesta sea 200.

Consultar Saldo:

Se realiza la consulta a la base de datos para obtener el id y el saldo del usuario registrado

Endpoint: http://localhost:3000/users/getIdUser/:no_cuenta

“Obtener saldo y id de un usuario - test”

Se valida que retorne un JSON con el saldo y el id del usuario que está logueado, en esta prueba se utiliza el no. de cuenta “14022020”, que es de un usuario previamente registrado en el sistema. Además se verifica que el estado de la respuesta sea 200.

Transacción:

Se crea un JSON, el cual contiene todos los datos de la nueva transferencia a registrar.



Endpoint: (POST) <http://localhost:3000/transferencias/>

“Debería insertar una transferencia”

Se valida que los usuarios estén creados y que el monto sea mayor o igual al saldo de la persona que hace la transacción para restarle el monto y a la persona que recibe la transacción sumarle el monto, en esta prueba se utiliza el usuario 1 y el usuario 3. Además se verifica que el estado de la respuesta sea 200.

Se consulta en la BD las transacciones realizadas.

Endpoint: (GET) <http://localhost:3000/transferencias/>

“Debería obtener todas las transferencias”

Se valida que retorne un JSON con todas las transferencias que ha realizado. Además se verifica que el estado de la respuesta sea 200.

Reporte:

Se consulta en la BD las transacciones realizadas por un usuario específico.

Endpoint: <http://localhost:3000/transferencias/:id>

“Debería obtener todas las transferencias relacionadas al usuario con id 7”

Se valida que retorne un JSON con todas las transferencias que ha realizado el usuario relacionado al id enviado en la ruta, en este caso se validan las transferencias del usuario con id 7. De no producirse algún error en la consulta se verifica que el estado de la respuesta sea 200.

Endpoint: <http://localhost:3000/transferencias/:id>

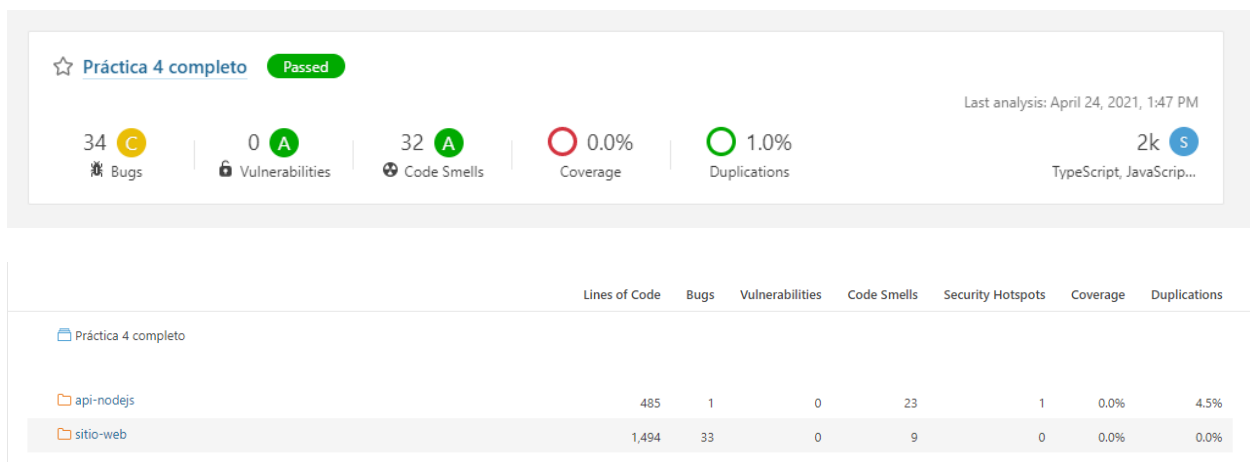
“Debería obtener una respuesta de tipo objeto”

Se valida que retorne un objeto JSON con todas las transferencias que ha realizado el usuario relacionado al id enviado en la ruta, en este caso se validan las transferencias del usuario con id 6. De no producirse algún error en la consulta se verifica que lo retornado sea de tipo ‘object’

SonarQube:

Permite hacer análisis estático del código y obtener métricas que permiten mejorar la calidad del código y descubrir evidencias de los problemas existentes en el código.

Al realizar el análisis a nuestra práctica obtenemos el siguiente resultado



Errores: a continuación se explicaran a detalles los tipos de errores que se encontraron.

Type			
Bug			34
Vulnerability			0
Code Smell			32
Security Hotspot			1
Severity			
Blocker	4	Minor	34
Critical	0	Info	0
Major	28		

Bugs: Un bug es un error o un defecto en el software que hace que un programa funcione de forma incorrecta, en este caso se detectaron bug's en el front-end, para ser mas especifico se detectaron errores simples en el HTML, como por ejemplo:

- Uso incorrecto de algunas propiedades dentro de etiquetas
- Etiquetas que ya no se utilizan en la versión del framework utilizado.

Además, esta herramienta nos muestra sugerencias para corregir dichos errores.

The image shows a bug tracking tool interface. The top section displays a list of bugs with filters on the left. The filters include Type (Bug, Vulnerability, Code Smell, Security Hotspot) and Severity (Blocker, Critical, Major, Minor, Info). The main list shows bugs with details like file path, description, severity, and status. The bottom section shows a detailed view of a bug in the file `api-nodejs/public/index.html`, highlighting the issue with the `<html>` tag and providing a suggestion to insert a `<!DOCTYPE>` declaration.

Filters:

- Type: Bug (34), Vulnerability (0), Code Smell (32), Security Hotspot (1)
- Severity: Blocker (0), Critical (0), Major (2), Minor (32), Info (0)

Bugs List:

File Path	Description	Severity	Status	Effort	Comment
api-nodejs/public/index.html	Insert a <code><!DOCTYPE></code> declaration to before this <code><html></code> tag.	Major	Open	5min	26 days ago, L1, user-experience
sitio-web/src/app/app.component.html	Add an "alt" attribute to this image.	Minor	Open	5min	21 days ago, L3, accessibility
sitio-web/src/app/app.component.html	Replace this <code></code> tag by <code></code> .	Minor	Open	2min	26 days ago, L10, accessibility
sitio-web/src/app/components/consulta/consulta.component.html	Replace this <code><i></code> tag by <code></code> .	Minor	Open	2min	25 days ago, L11, accessibility
sitio-web/src/app/components/consulta/consulta.component.html	Replace this <code><i></code> tag by <code></code> .	Minor	Open	2min	25 days ago, L17, accessibility
sitio-web/src/app/components/consulta/consulta.component.html	Replace this <code><i></code> tag by <code></code> .	Minor	Open	2min	22 days ago, L27, accessibility

Bug Detail View:

File: `api-nodejs/public/index.html`

Line 1: `<html>`

Suggestion: Insert a `<!DOCTYPE>` declaration to before this `<html>` tag.

Severity: Major, Status: Open, Effort: 5min, Comment: 26 days ago, L1, user-experience

Duplicados: hace referencia al código fuente que aparece más de una vez, lo cual puede generar una dificultad mayor al mantener el código y generar más costos, ya que un buen desarrollo está más asociado a la reutilización del mismo.

En nuestra aplicación se encontró este error únicamente en el archivo de rutas(back-end), ya que los métodos son similares y podrían compartir algunas variables y código entre ellas, sin embargo esto hace que dicho archivo sea más simple de entender entre los distintos desarrolladores del equipo.

Duplications Overview	
Overview	
Overall	
Density	1.0%
Duplicated Lines	26
Duplicated Blocks	2
Duplicated Files	1

<div>Reliability</div> <div>Security</div> <div>Maintainability</div> <div>Coverage</div> <div>Duplications</div> <div>Overview</div> <div>Overall</div> <div>Density</div> <div>Duplicated Lines</div> <div>Duplicated Blocks</div> <div>Duplicated Files</div>	Duplicated Lines 26			
			Duplicated Lines	Duplicated Lines (%)
	api-nodejs		26	4.5%
	sitio-web		0	0.0%
	2 of 2 shown			

Práctica 4 completo / api-nodejs / routes / users.js

```
29     }
30   });
31
32   //Envia true si el usuario de acuerdo al id recibido existe
33   router.get('/:no_cuenta', function (req, res, next) {
34     sql.connect(config, function (err) {
35       if (err) console.log(err);
36       var request = new sql.Request();
37       query = `
38         SELECT * FROM usuario u
39         WHERE u.no_cuenta = '${req.params.no_cuenta}';
40       `;
41       request.query(query, function (err, data) {
42         if (err) {
43           res.json(false);
44         } else {
45           if (data.recordset.length > 0) {
46             res.json(false);
47           } else {
48             res.json(true);
49           }
50         }
51       });
52     });
53   });
54 }
```

Complejidad ciclomática: Es una métrica de calidad software basada en el cálculo del número de caminos independientes que tiene nuestro código. Cuanto más compleja sea la lógica de un código, más difícil será de entender, mantener y probar.

[McCabe](#) propuso el siguiente límite:

Complejidad Ciclométrica

1-10

11-20

21-50

50

Evaluación del Riesgo

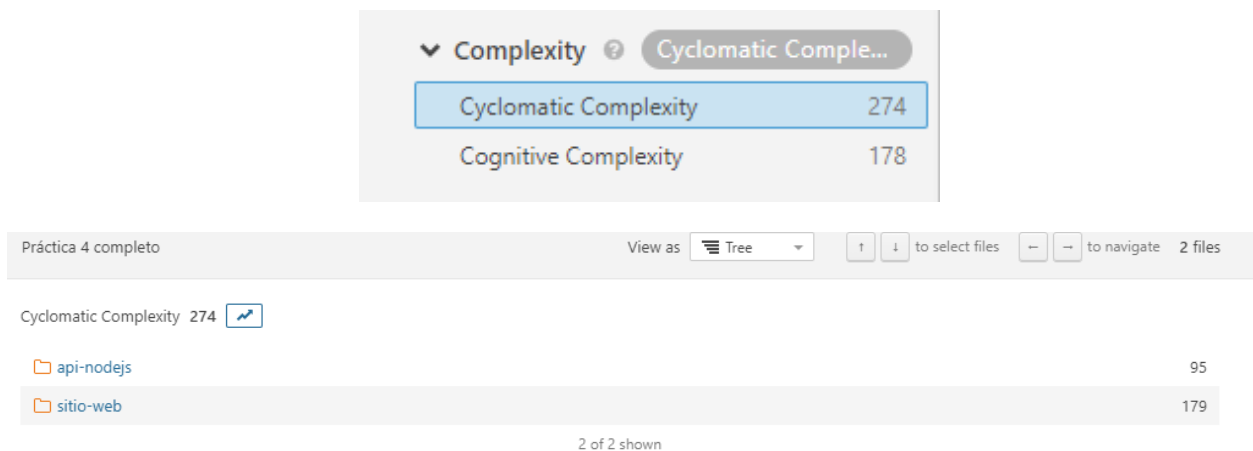
Programa Simple, sin mucho riesgo

Más complejo, riesgo moderado

Complejo, Programa de alto riesgo

Programa no testable, Muy alto riesgo

Entre más grande es un proyecto más complejo se vuelve, por lo cual nuestra aplicación marca 274 de complejidad, este valor es la suma de la complejidad individual de los archivos/funciones.



En las siguientes imágenes se muestra que individualmente los archivos de nuestra aplicación entran en el rango de “riesgo moderado”, con la excepción del archivo users.js que es el que maneja todas las rutas y posee código duplicado, por eso excede el límite de 50.

Práctica 4 completo / sitio-web / src / app / components		View as	Tree	↑	↓	to select files	←	→	to navigate	7 files
Cyclomatic Complexity 98										
consultas										10
home										7
login										12
photo-upload										14
register										20
reporte										17
transferencia										18
7 of 7 shown										

Práctica 4 completo / api-nodejs / routes		View as	Tree	↑	↓	to select files	←	→	to navigate	2 files
Cyclomatic Complexity 67										
transferencias.js										15
users.js										52
2 of 2 shown										

Pruebas Funcionales

Pruebas de Humo

Subconjunto de pruebas que cubren la funcionalidad principal de un sistema, pero sin preocuparse por los detalles finos

Puppeteer:

Es una librería de Node.js que proporciona una API de alto nivel que permite automatizar acciones sobre los navegadores de Google

Mocha :

Es un framework de pruebas para Node JS que puede ejecutarse desde la consola o desde un navegador. Al ejecutar los test, permite la presentación de informes flexibles y precisos.

Chai:

Es un librería de aserciones, la cual se puede emparejar con cualquier marco de pruebas de Javascript

Login

URL: <http://localhost:4200/>

Se carga la pagina de login y se verifica que existan los inputs de cuenta y contraseña. Se ingresan los valores para cuenta y contraseña y al final se captura la página con los datos ingresados.


Registro

URL: <http://localhost:4200/register/>

Se carga la página de registro y se verifica que existan los campos para que el usuario se pueda registrar, se ingresan los datos. Y luego se captura la página con los datos previamente ingresados.

Perfil

URL: <http://localhost:4200/home/>



Se carga la página de perfil, y se toma una captura de la página para visualizar que se encuentren todos los campos que deberían.

Transferencia

URL: <http://localhost:4200/transferencia/>

Se carga la página de transferencia y se verifica que existan los inputs de cuenta destino y monto, se ingresan los datos. Y luego se captura la página con los datos previamente ingresados. La imagen se guarda con el nombre transferencia.

Consulta de Saldo

URL: <http://localhost:4200/consulta/>

Se carga la página de consulta de saldo, y se toma una captura de la página para visualizar que se encuentren todos los campos que deberían.