Actividad: Paso a Paso Pruebas API

Objetivo

El objetivo de este trabajo práctico es que los alumnos desarrollen las pruebas unitarias y de integración utilizando jest y supertest. Las pruebas deben aplicarse sobre la API de Biblioteca.

Consigna

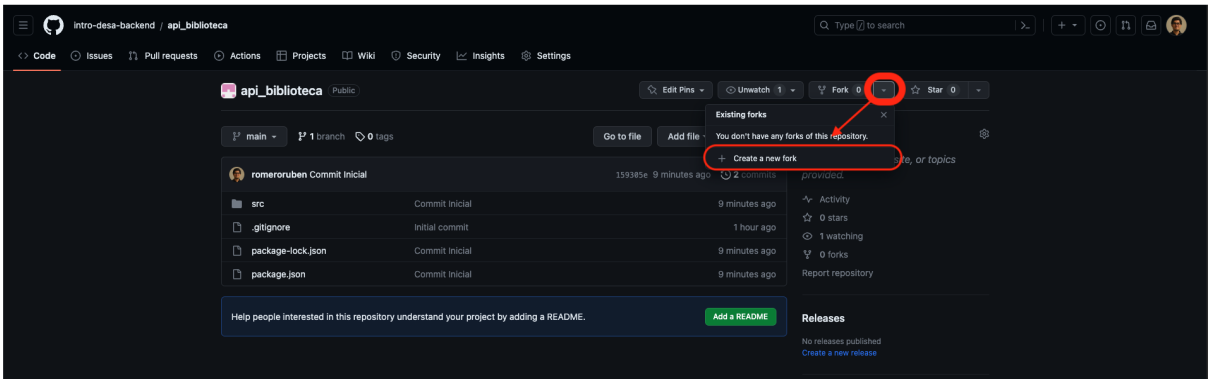
1. Hacer un Fork del repositorio base.
2. Agregar los test unitarios y de integración en la carpeta Tests.
3. Ejecutar los Tests.
4. Verificar el nivel de cobertura de los test.
5. Subir los cambios a GitHub.

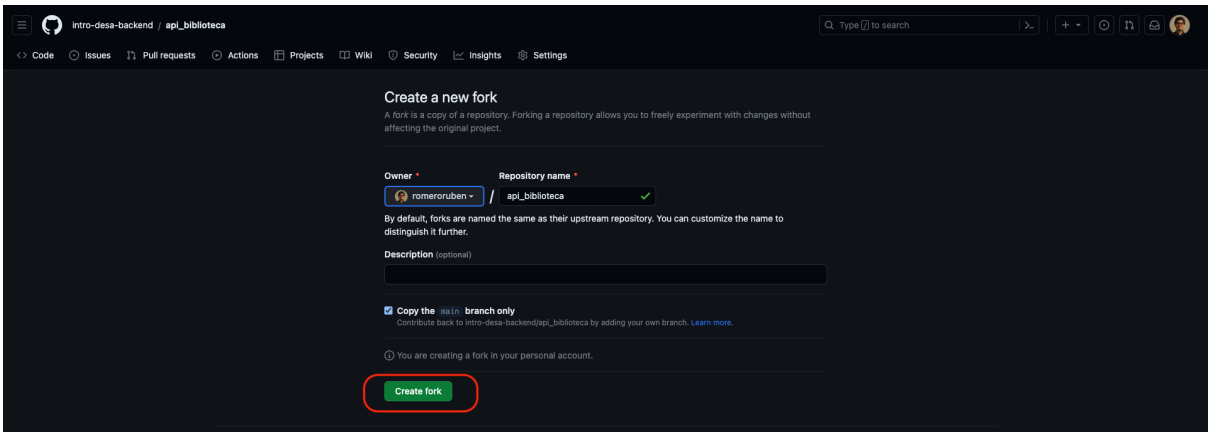
Desarrollo Paso a Paso

1. Primero vamos a ingresar al siguiente link que contiene el repositorio api_biblioteca de GitHub:

   https://github.com/intro-desa-backend/api_biblioteca

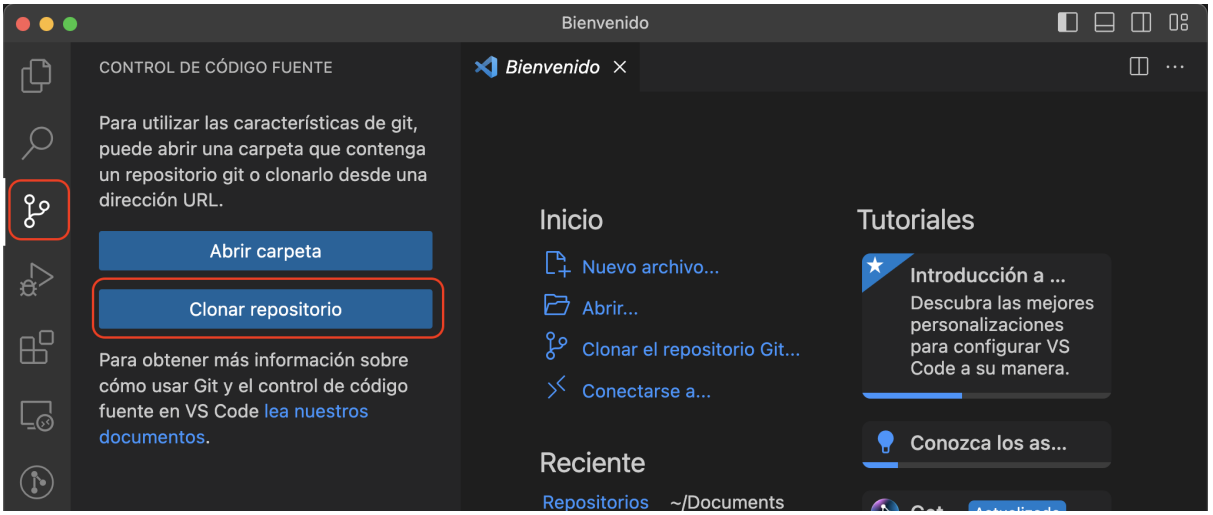
2. Vamos a crear un FORK de este repositorio para nuestra cuenta de GitHub. FORK en GitHub es una funcionalidad que permite a los usuarios hacer una copia de un repositorio ajeno en su cuenta de GitHub.



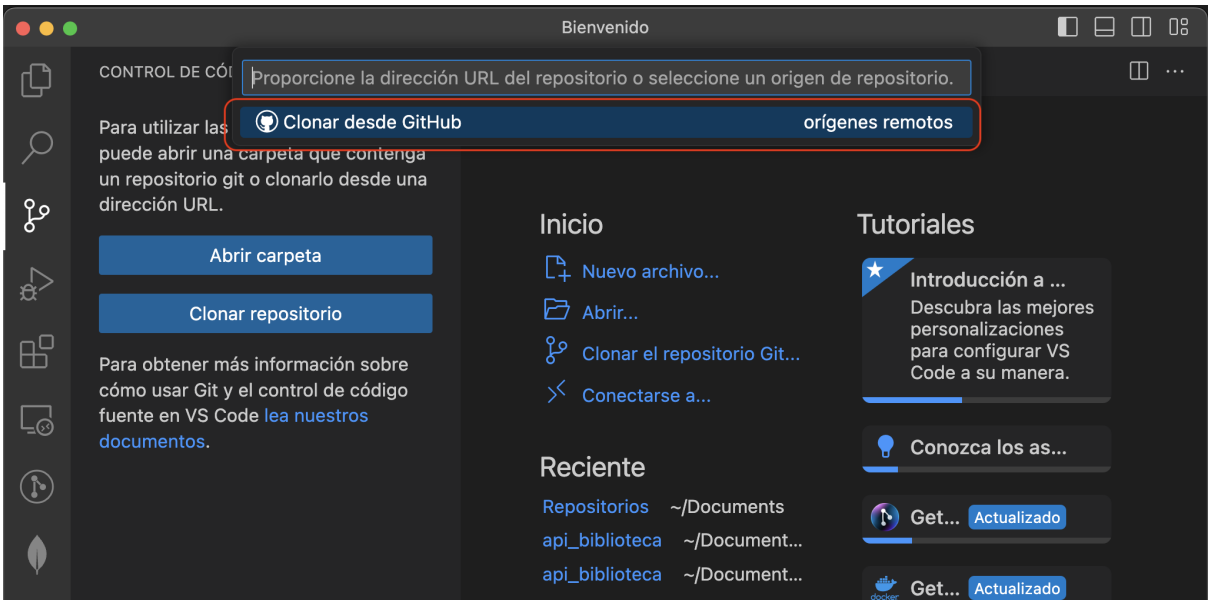


3. Una vez que hicimos el FORK ya podemos clonar el repositorio localmente. Iniciamos la aplicación VSCode.

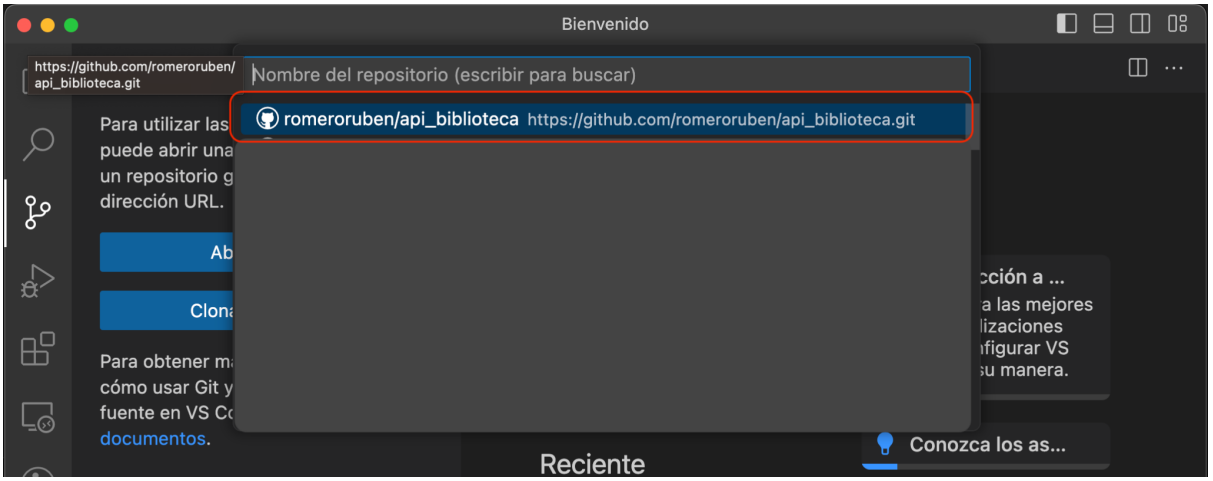
4. Vamos a la opción de “Control de código fuente”, y seleccionamos la opción “Clonar repositorio” y seguimos los pasos:



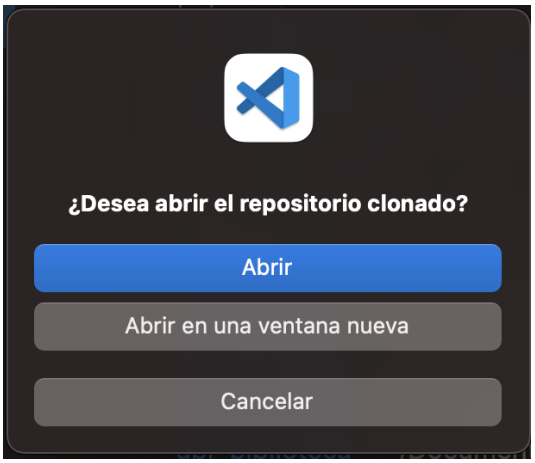
5. Hacemos click en “Clonar desde GitHub”



6. Seleccionamos el repositorio api_biblioteca que esta en nuestra cuenta:



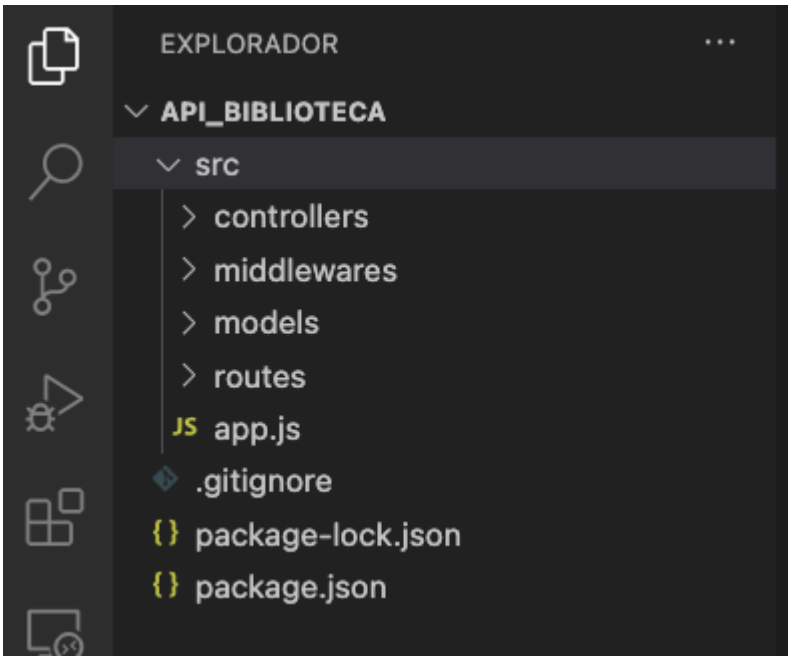
7. En el siguiente paso nos muestra el mensaje “Desea abrir el repositorio clonado?”, y hacemos click en **Abrir**



8. Esta es la estructura de archivos que deberían ver en su VSCode:

Nota: Tener en cuenta los siguientes puntos:

1. Se agregó una carpeta **src** donde colocamos el código que antes teníamos en la raíz del repositorio. Esto es para poder separar la carpeta del código fuente de los tests que vamos a hacer.
2. Agregamos un carpeta controllers donde tenemos el libroController.js, para poder separar la lógica de los routes y poder probar las funciones de manera aislada.



9. Ahora vamos a instalar los paquetes de npm localmente con el comando desde un terminal:

```
npm install
```

10. Debemos crear el archivo **.env** (de variables de entorno) con el siguiente contenido:

```
OAUTH_AUDIENCE=http://localhost:3000/api/biblioteca
OAUTH_URL=https://dev-utn-frc-iaew.auth0.com/
PORT=443
MONGO_DB=mongodb://127.0.0.1:27017/biblioteca
```

11. En el archivo package.json se definieron los scripts **start** (para iniciar la API) y **test** (para ejecutar los tests):

```
  "scripts": {
    "start": "node src/app.js",
    "test": "jest"
  },
```

12. Vamos a iniciar la API para probar que funcione todo bien:

```
npm start
```

13. Para probar la API necesitan crear un TOKEN, con el siguiente CURL, lo pueden importar desde Postman y crear un access_token:

```
curl --location 'https://dev-utn-frc-iaew.auth0.com/oauth/token' \
--header 'content-type: application/json' \
--header 'Cookie:
did=s%3Av0%3A76204e80-0ef1-11ee-8e12-b99ba3014b47.fKR174NkC7L0s9Z8riCysG
37mMeRVakoS%2BTtye2lC3g;
did_compat=s%3Av0%3A76204e80-0ef1-11ee-8e12-b99ba3014b47.fKR174NkC7L0s9Z
8riCysG37mMeRVakoS%2BTtye2lC3g' \
--data '{
    "client_id": "QiW8AlH9oykBg7ofBrHs6ToYvrdmhOeE",
    "client_secret":
"7kZPQqNnhRAuCXipSVSdHUsV9MzgfUzBB3AYbfemGPqxtpXI6j1GNxDBfYBSUume",
    "audience": "http://localhost:3000/api/biblioteca",
    "grant_type": "client_credentials"
```

```
}'
```

14. Importando este CURL en Postman pueden probar que la API funcione ok, reemplazando el TOKEN.

```
curl --location 'http://localhost:3000/api/libros' \
--header 'Authorization: Bearer TOKEN'
```

Pruebas Unitarias

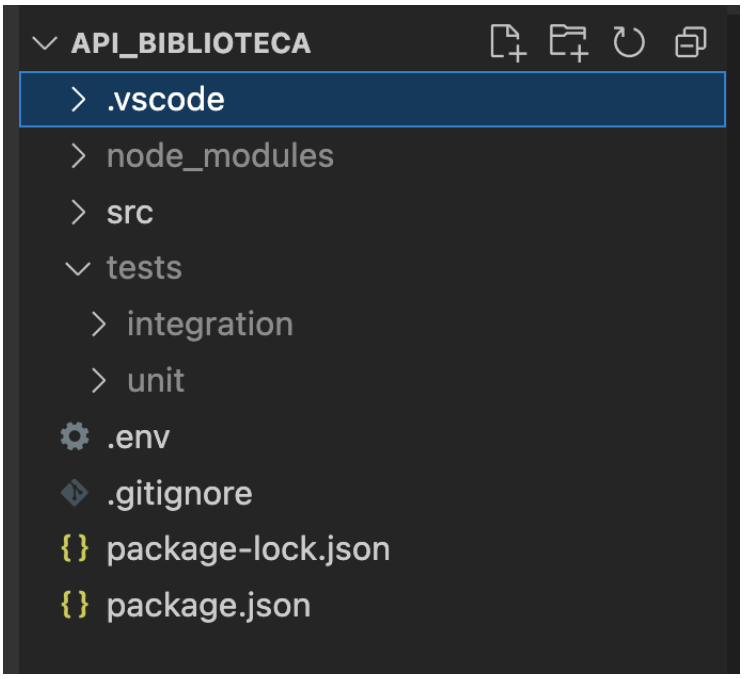
15. Vamos a instalar Jest en nuestra computadora (esta instalación se hace por única vez, el parámetro -g hace referencia a una instalación global en nuestra computadora.):

```
npm install -g jest
```

16. Ahora vamos a instalar la librería Jest en nuestro proyecto:

```
npm install --save-dev jest
```

17. Creamos la carpeta **/tests**, y dentro las carpetas **/unit** y la carpeta **/integration,** quedando de la siguiente forma:



18. Dentro de la carpeta **/test/unit**, vamos a crear el archivo **libroController.test.js** con el siguiente contenido:

Nota: En el siguiente código probamos las funciones definidas en el controlador libroController, pero para no afectar los datos de la base de datos utilizamos un mockup (simulación) de libroModel (que contiene la definición de mongoose para acceder a la base de datos).

```
const {
  getAllLibros,
  createLibro,
  updateLibro,
  deleteLibro,
  getLibroById,
} = require("../../src/controllers/libroController");

const libroModel = require("../../src/models/libroModel");

jest.mock("../../src/models/libroModel");

describe("Libro Controller", () => {
  let mockRes;

  beforeEach(() => {
    mockRes = {
      status: jest.fn().mockReturnThis(),
```

```
      json: jest.fn(),
    };
  });

  test("getLibros debería obtener todos los libros", async () => {
    const mockLibros = [
      { id: "1", title: "Libro 1" },
      { id: "2", title: "Libro 2" },
    ];

    libroModel.find.mockResolvedValue(mockLibros);

    const mockReq = {};

    await getAllLibros(mockReq, mockRes);

    expect(mockRes.status).toHaveBeenCalledWith(200);
    expect(mockRes.json).toHaveBeenCalledWith(mockLibros);
  });


  test("getLibroById debería obtener un libro", async () => {
    const mockLibro = { id: "1", titulo: "Libro Encontrado", autor:
"Juan Perez" };

    libroModel.findById.mockResolvedValue(mockLibro);

    const mockReq = { params: { id: "1" } };

    await getLibroById(mockReq, mockRes);

    expect(mockRes.status).toHaveBeenCalledWith(200);
    expect(mockRes.json).toHaveBeenCalledWith(mockLibro);
  });

  test("createLibro debería crear un nuevo libro", async () => {
    const mockLibro = { id: "1", titulo: "Nuevo Libro", autor: "Juan
Perez" };
    mockLibro.save = () => {};

    libroModel.create.mockResolvedValue(mockLibro);

    const mockReq = { body: mockLibro };
```

```
    await createLibro(mockReq, mockRes);

    expect(mockRes.status).toHaveBeenCalledWith(201);
    expect(mockRes.json).toHaveBeenCalledWith(mockLibro);
  });

  test("updateLibro debería actualizar un libro existente", async () =>
{
    const libroId = '1';
    const libroActualizado = { titulo: 'Libro Actualizado', autor:
'Autor Actualizado' };
    const libroActualizadoMock = { _id: libroId, ...libroActualizado };


libroModel.findByIdAndUpdate.mockResolvedValue(libroActualizadoMock);

    const mockReq = { params: { id: "1" }, body: libroActualizado };

    await updateLibro(mockReq, mockRes);

    expect(libroModel.findByIdAndUpdate).toHaveBeenCalledWith(libroId,
libroActualizado, { new: true });
    expect(mockRes.status).toHaveBeenCalledWith(200);
    expect(mockRes.json).toHaveBeenCalledWith(libroActualizadoMock);
  });

  test("updateLibro debería devolver un error si el libro no existe",
async () => {

    libroModel.findByIdAndUpdate.mockResolvedValue(null);

    const mockReq = {
      params: { id: "99" },
      body: { titulo: "Libro Actualizado" },
    };

    await updateLibro(mockReq, mockRes);

    expect(mockRes.status).toHaveBeenCalledWith(404);
    expect(mockRes.json).toHaveBeenCalledWith({ error: "Libro no
encontrado" });
  });

  test("deleteLibro debería eliminar un libro existente", async () => {
```

```
    const mockLibroEliminado = { titulo: 'Libro Eliminado', autor:
'Autor Eliminado' };

    libroModel.findByIdAndRemove.mockResolvedValue(mockLibroEliminado);

    const mockReq = { params: { id: "1" } };

    await deleteLibro(mockReq, mockRes);


expect(libroModel.findByIdAndRemove).toHaveBeenCalledWith(mockReq.params
.id);
    expect(mockRes.status).toHaveBeenCalledWith(200);
    expect(mockRes.json).toHaveBeenCalledWith(mockLibroEliminado);
  });
});
```

19. Ejecutamos las pruebas con el siguiente comando:

```
npm test
```

Respuesta esperada:

```
> jest

 PASS  tests/unit/libroController.test.js
  Libro Controller
    ✓ getLibros debería obtener todos los libros (3 ms)
    ✓ getLibroById debería obtener un libro (1 ms)
    ✓ createLibro debería crear un nuevo libro (1 ms)
    ✓ updateLibro debería actualizar un libro existente (1 ms)
    ✓ updateLibro debería devolver un error si el libro no existe
    ✓ deleteLibro debería eliminar un libro existente (1 ms)

Test Suites: 1 passed, 1 total
Tests:       6 passed, 6 total
Snapshots:   0 total
Time:        0.737 s, estimated 2 s
Ran all test suites.
```

Pruebas Integración

20. Para las pruebas de integración vamos a usar la librería **supertest**. Para eso debemos instalar con el siguiente comando:

```
npm install --save-dev supertest
```

21. Dentro de la carpeta **/test/integration**, creamos el archivo **libroIntegration.test.js** con el siguiente código:

Nota: Aquí no solo tenemos que mockear la base de datos sino también la autenticación para que el test se ejecute correctamente:

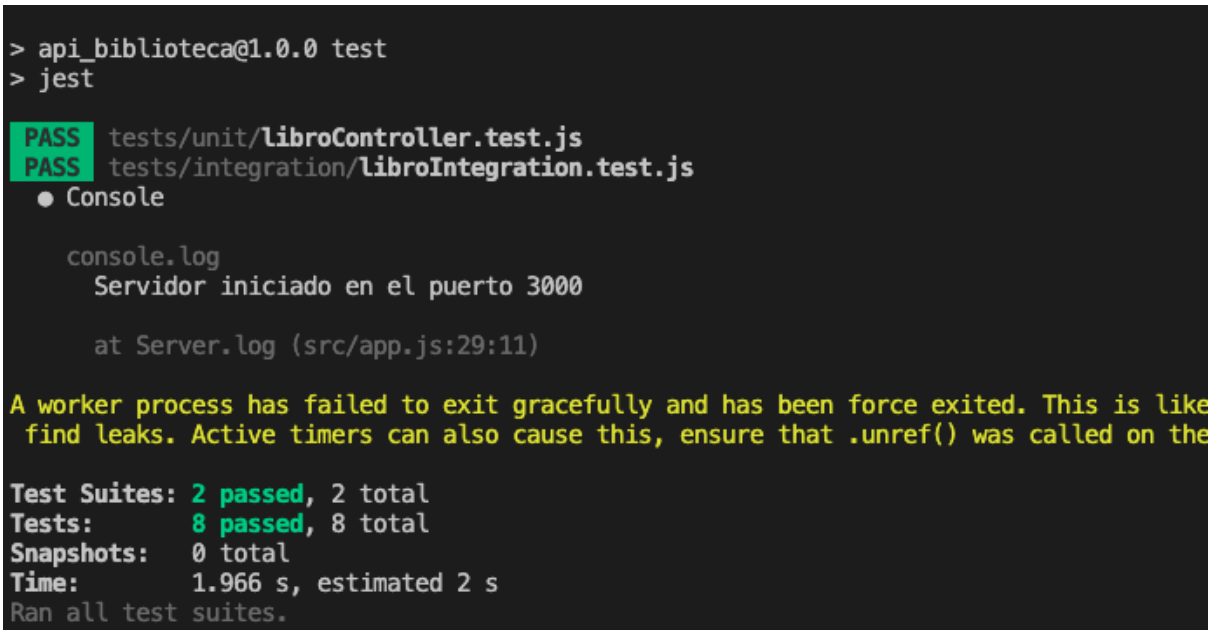
```
const request = require("supertest");
const app = require("../../src/app");
const libroModel = require("../../src/models/libroModel");

// Mockup de Autenticación
jest.mock("express-oauth2-jwt-bearer", () => {
  return {
    auth: jest.fn().mockImplementation(() => (req, res, next) =>
next()),
    requiredScopes: jest.fn().mockImplementation(() => (req, res, next)
=> next()),
  };
});

//Mockup de Mongoose
jest.mock("../../src/models/libroModel");

describe("Libro API", () => {
  test("GET /libros debería obtener todos los libros", async () => {
    const mockLibros = [
      { id: "1", title: "Libro 1" },
      { id: "2", title: "Libro 2" },
    ];

    libroModel.find.mockResolvedValue(mockLibros);

    const response = await request(app).get("/api/libros");

    expect(response.status).toBe(200);
    expect(response.body).toEqual(mockLibros);
```

```
    expect(libroModel.find).toHaveBeenCalledTimes(1);
  });

  test("POST /libros debería crear un nuevo libro", async () => {
    const libroCreado = { id: "1", titulo: "Nuevo Libro", autor: "Juan
Perez" };
    const libroMock = {
      ...libroCreado,
      save: () => {}
    };

    libroModel.create.mockResolvedValue(libroMock);

    const response = await
request(app).post("/api/libros").send(libroMock);

    expect(response.status).toBe(201);
    expect(response.body).toEqual(libroCreado);
    expect(libroModel.create).toHaveBeenCalledTimes(1);
    expect(libroModel.create).toHaveBeenCalledWith(libroCreado);
  });
});
```

22. Ejecutamos las pruebas con el siguiente comando:

```
npm test
```

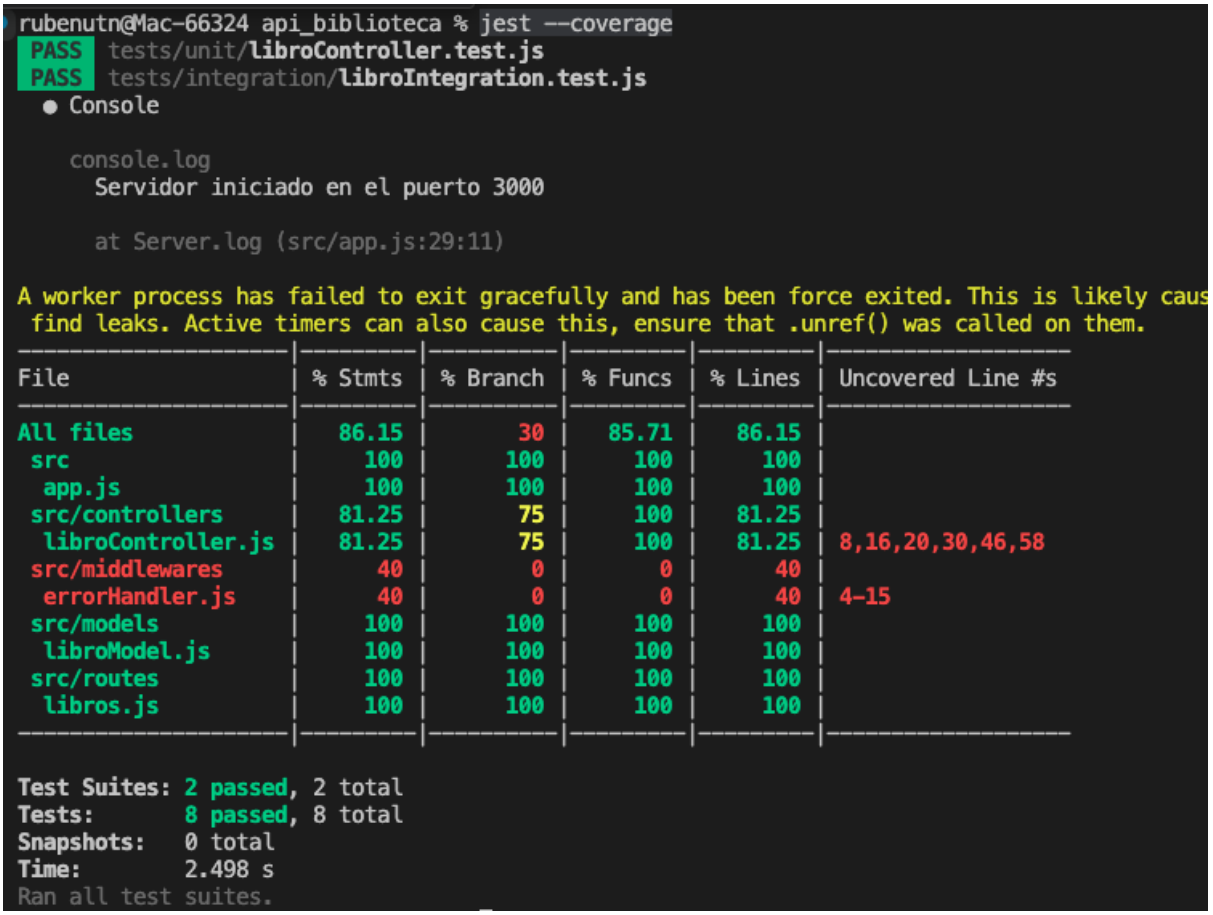
Respuesta esperada:


> api_biblioteca@1.0.0 test
> jest
PASS tests/unit/libroController.test.js
PASS tests/integration/libroIntegration.test.js
● Console
console.log
Servidor iniciado en el puerto 3000
at Server.log (src/app.js:29:11)
A worker process has failed to exit gracefully and has been force exited. This is like
find leaks. Active timers can also cause this, ensure that .unref() was called on the
Test Suites: 2 passed, 2 total
Tests: 8 passed, 8 total
Snapshots: 0 total
Time: 1.966 s, estimated 2 s
Ran all test suites.

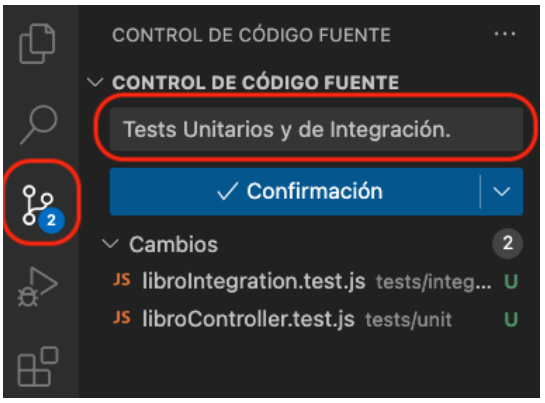
23. Podemos verificar el % de cobertura de test en nuestro código ejecutando el siguiente comando:

```
jest --coverage
```

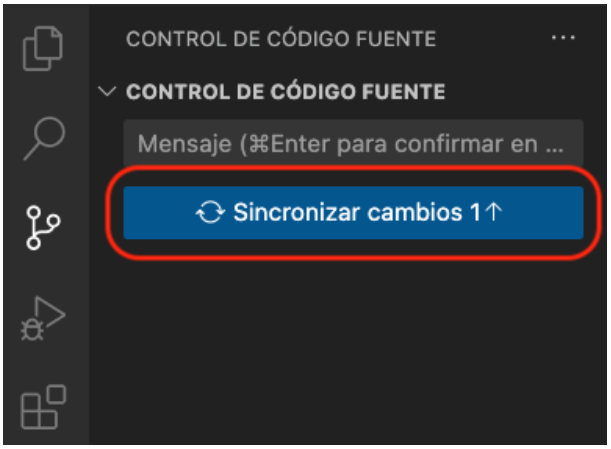
Resultado esperado:



24. Ahora vamos a subir los cambios a GitHub.
    a. Primero vamos a crear el Commit, en VSCode buscamos la opción de “Control de Código Fuente”, ingresamos un comentario “Tests Unitarios y de Integración” y luego hacemos click en Confirmar, como se ve a continuación:



    b. Luego hacemos el PUSH de los archivos a GitHub



25. IMPORTANTE! Para dar por finalizada la actividad compartir la url de GitHub del repositorio en su cuenta de GitHub en la actividad del aula virtual en https://uve.frc.utn.edu.ar/.