in collaboration with

ComixNook

Nirajan Mahato

BSc. (Hons) Computing, Softwarica College of IT and E-commerce

ST5007CEM web Development

Hari Saran Shrestha

March 01, 2024

GitHub Link: NirajanMahato/ComixNook-Web (github.com)

Table of Contents

Table of Figures

Introduction

Comix**N**ook is an online platform that offers a comprehensive collection of comics for users to browse and download. With a focus on user experience and convenience, comixNook provides a user-friendly interface that allows users to explore a wide range of comics across various genres. *(Marvel Comics | Marvel Comic Books)*

COMIXNOOK

It is the ultimate destination for all comic enthusiasts, whether they are looking for classic favorites, emerging talent, or niche genres. ComixNook provides a centralized hub for discovering and accessing comics, enhancing the reading experience and fostering a community of comic enthusiasts. With its diverse selection, user-friendly interface, and thriving community, this platform offers comic enthusiasts an exciting journey of discovery and inspiration. (Viewcomics, n.d.)

Aim

This project aims to create an all-inclusive online platform that will allow users to easily browse and download a vast collection of comics. The platform will feature a user-friendly interface, strong backend functionality, and a seamless browsing experience across different genres and authors. (GetComics, n.d.)

Objectives

- Develop a user-friendly frontend interface that allows users to browse and search for comics efficiently.
- Implement a secure user authentication system to manage user accounts and access permissions.
- Design and develop a backend infrastructure using Spring Boot and PostgreSQL to store and manage comic data.
- Provide seamless integration between the frontend and backend components through RESTful APIs.
- Implement features for users to explore, discover, and download comics, including categorization, search functionality, and download options.

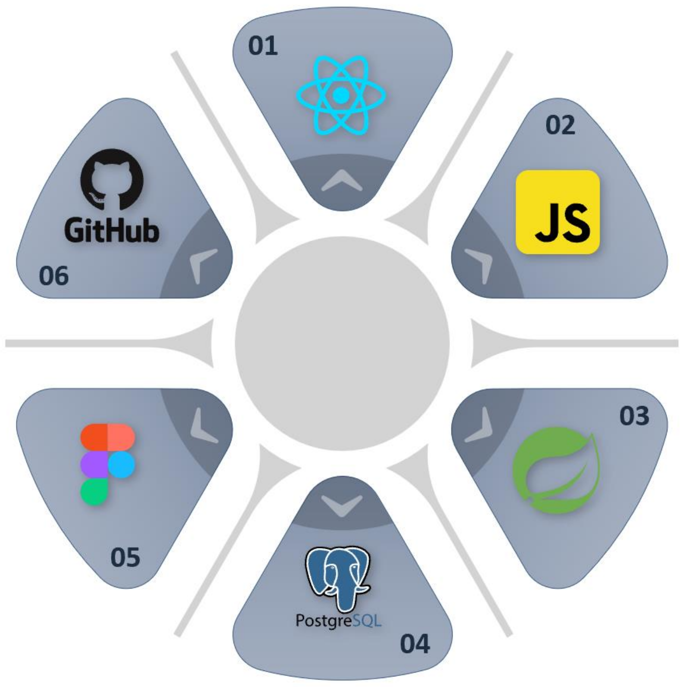
Features

- **User-friendly Interface:** Navigate the platform effortlessly with an intuitive interface for optimal user experience.
- **User Accounts:** Create personalized user accounts to manage preferences.
- **Featured Comics:** Explore hand-picked collections of popular comics, featuring new releases and trending series.
- **Browse by Genre:** Discover new comics with ease through our categorized genres.
- **Comic Details:** Get detailed information about each comic, including summary, release date, and cover art.
- **Download Comics:** Download comics and read them offline for maximum enjoyment at your convenience.
- **Search Functionality:** Quickly find desired comics using robust search functionality, allowing users to search by title, genre, or keywords.
- **Admin Dashboard:** Manage genre, Manage comics, user accounts, and platform settings efficiently through an intuitive admin dashboard.
- **Responsive Design:** Ensure a responsive design that adapts seamlessly to various screen sizes and resolutions, ensuring an optimal viewing experience on devices of all types, including desktops, laptops, tablets, and smartphones. (Comicextra, n.d.)

Learning Outcomes

During my participation in this project, I have had the privilege to embark on an inspiring journey of learning and growth in the field of web development. I honed my skills in both frontend and backend development by leveraging modern technologies such as React.js, Spring Boot, and PostgreSQL. Through hands-on activities, I gained practical experience that will serve me well in my future endeavors. I explored deep into user interface design, database management, and API integration, equipping myself with the tools needed to create responsive web applications with seamless user experiences and robust backend functionalities. Overcoming technical challenges along the way has been both rewarding and enlightening, fueling my passion for continuous learning and improvement. As I reflect on my active participation in this project, I am proud of the valuable skills, knowledge, and growth-oriented mindset that I have acquired. All of these will undoubtedly thrust me forward in the dynamic and ever-evolving field of web development.

Technologies Used

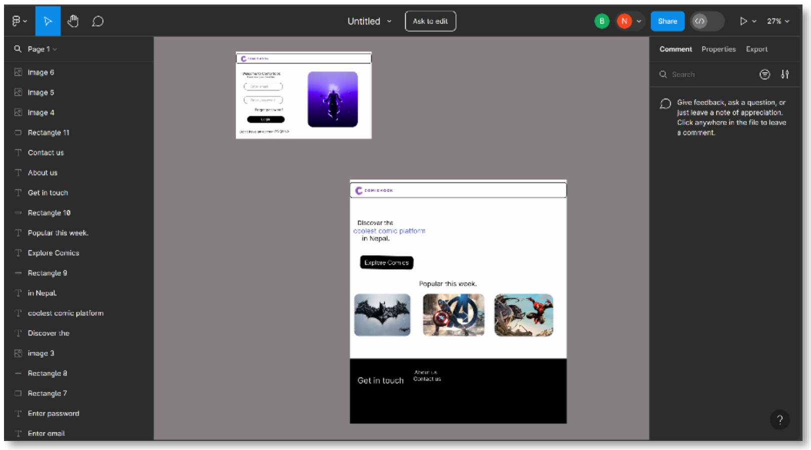


Frond-End Technologies

- **React.js:** A JavaScript library that enables fast and efficient creation of interactive frontend components.
- **CSS:** A style sheet language is used to describe the presentation of an HTML document, including its layout, fonts, and colors.
- **Tailwind CSS:** A CSS framework featuring pre-designed utility classes that quickly allows custom designs without the need for writing custom CSS.
- **JavaScript:** A programming language frequently used in web development to enhance web pages with interactive and dynamic features, supplementing HTML and CSS.

- **Prototyping:**

  Figma: A collaborative design tool for web and mobile apps. Figma allows real-time collaboration, efficient design iterations, and feedback.

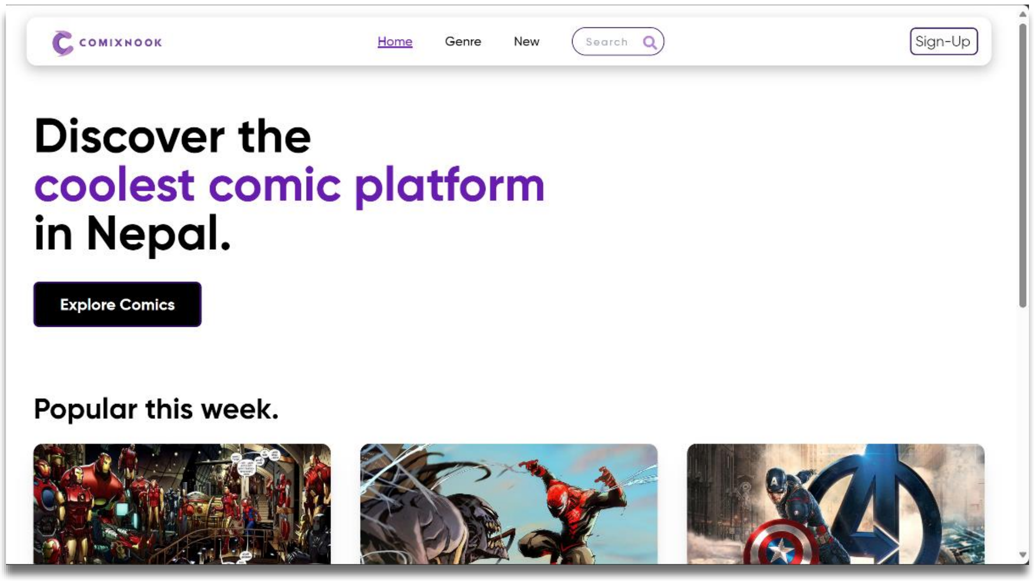


Responsive Design

In this project, I made sure to prioritize responsive design to create a seamless user experience on any device. I achieved this by using fluid layouts, media queries, and flexible media elements to ensure that the website's content adapted perfectly to different screen sizes. I utilized the viewport meta tag to have control over the viewport behavior on mobile devices, ensuring consistent rendering and functionality across various browsers. I decided to adopt a mobile-first approach, focusing on designing and developing the website for smaller screens before enhancing it for larger devices. I also used CSS Flexbox and Grid Layouts to create flexible and dynamic layouts that were intelligently adjusted to accommodate different screen orientations and resolutions. To ensure a smooth user experience on all devices, I tested the website using inspects, browser developer tools, and, fixing any layout or compatibility issues that I found. With all these efforts, I successfully delivered a website that was accessible, user-friendly, and visually appealing on any device, enhancing engagement and satisfaction for all users. (*Welcome: Login Page*, n.d.)

Snapshots of Frontend

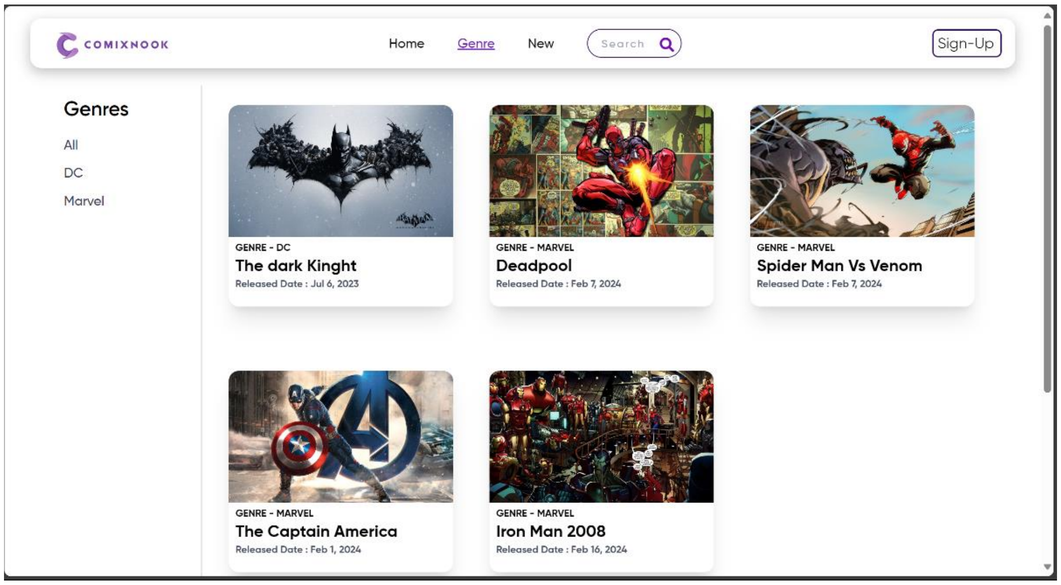
Figure 1:
ComixNook Homepage



Discover the
coolest comic platform
in Nepal.

Popular this week.

Figure 2:
Genre Page



Genres
All
DC
Marvel

GENRE - DC
The dark Kinght
Released Date : Jul 6, 2023

GENRE - MARVEL
Deadpool
Released Date : Feb 7, 2024

GENRE - MARVEL
Spider Man Vs Venom
Released Date : Feb 7, 2024

GENRE - MARVEL
The Captain America
Released Date : Feb 1, 2024

GENRE - MARVEL
Iron Man 2008
Released Date : Feb 16, 2024

Figure 3:
New Comics Page

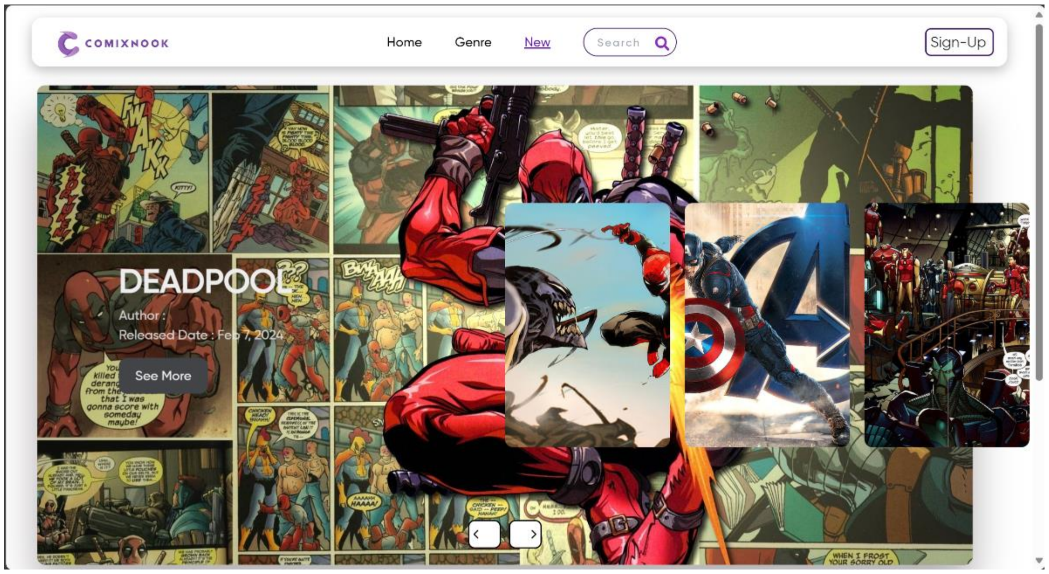


Figure 4:
Search Page

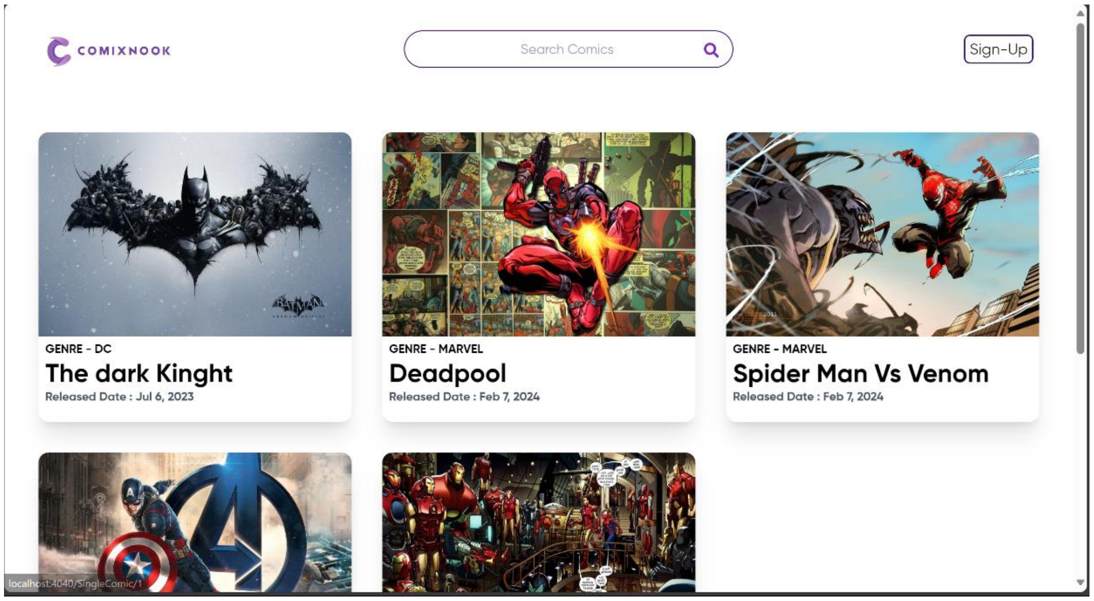
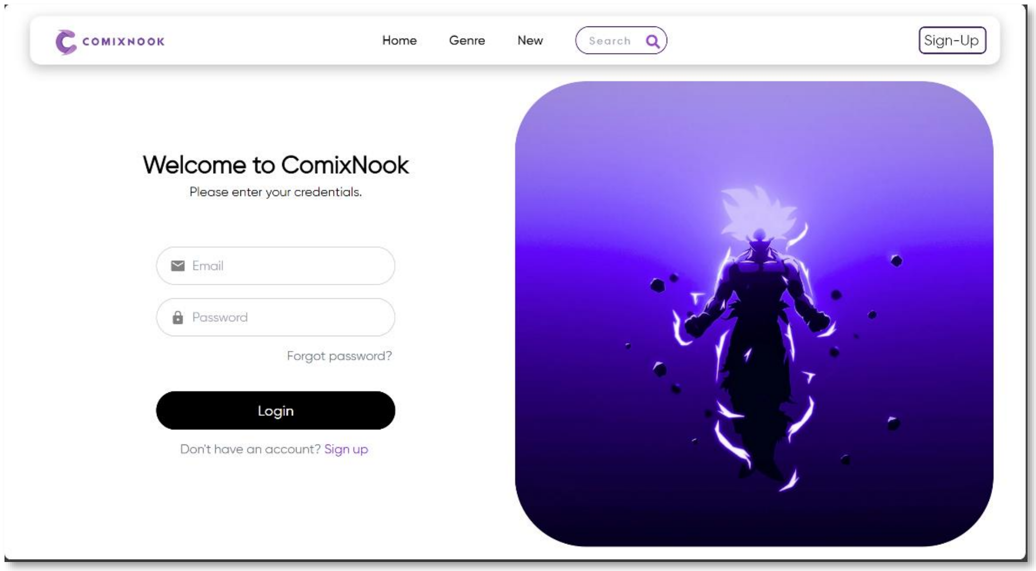

COMIXNOOK
Search Comics
Sign-Up
GENRE - DC
The dark Kinght
Released Date : Jul 6, 2023
GENRE - MARVEL
Deadpool
Released Date : Feb 7, 2024
GENRE - MARVEL
Spider Man Vs Venom
Released Date : Feb 7, 2024
localhost:4040/SingleComic/1

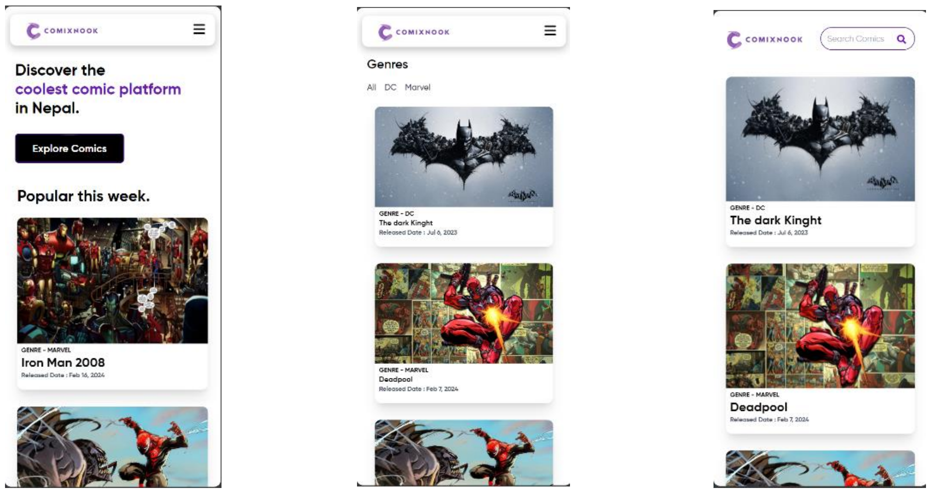
Figure 5:
Login Page



Welcome to ComixNook
Please enter your credentials.

Don't have an account? Sign up

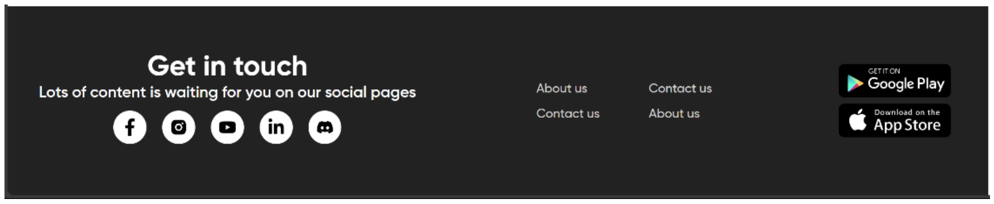
Figure 6:
Responsive Pages



Discover the coolest comic platform in Nepal.

Popular this week.

Figure 7:
Footer

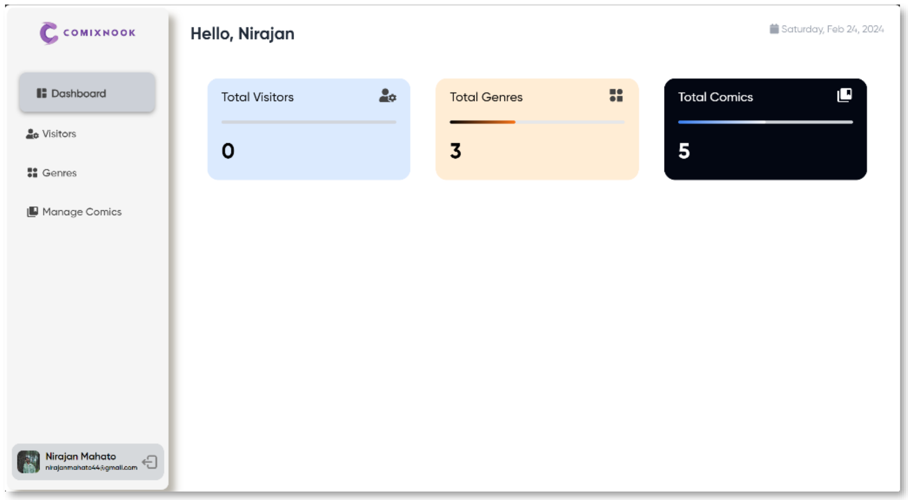


Get in touch
Lots of content is waiting for you on our social pages

Admin Dashboard

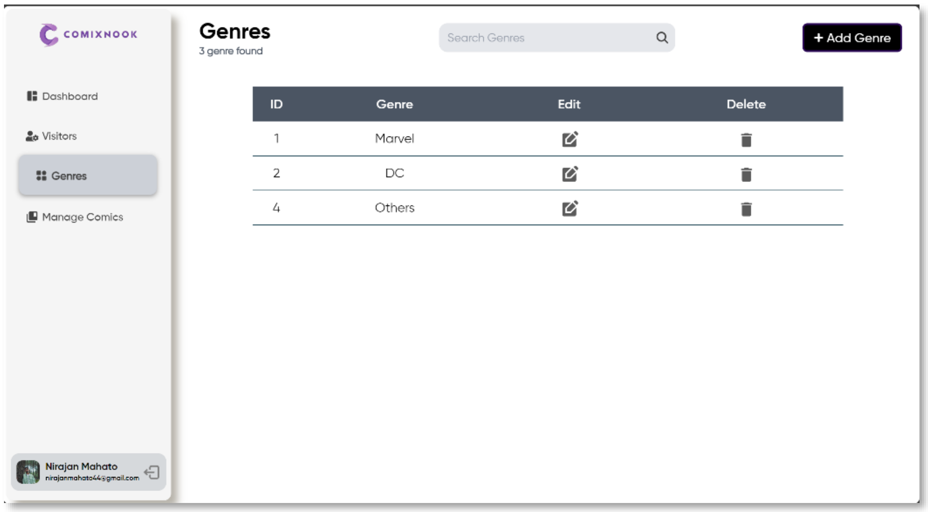
The admin dashboard serves as a central hub for managing various aspects of the application, providing administrators with tools to oversee and maintain the platform efficiently. The dashboard consists of multiple pages, each catering to specific administrative functions.

Figure 8:
Main Dashboard Page



The dashboard page provides an overview of essential metrics and statistics related to the application's functioning. It presents the current count of genres, users, and comics registered on the platform, giving administrators an instant look at the platform's activity. This page acts as a summary of the application's performance and user engagement, providing a bird's eye view of the platform's operations. [(*Bonita Dashboard*, n.d.)]

Figure 9:
Manage Genre Page

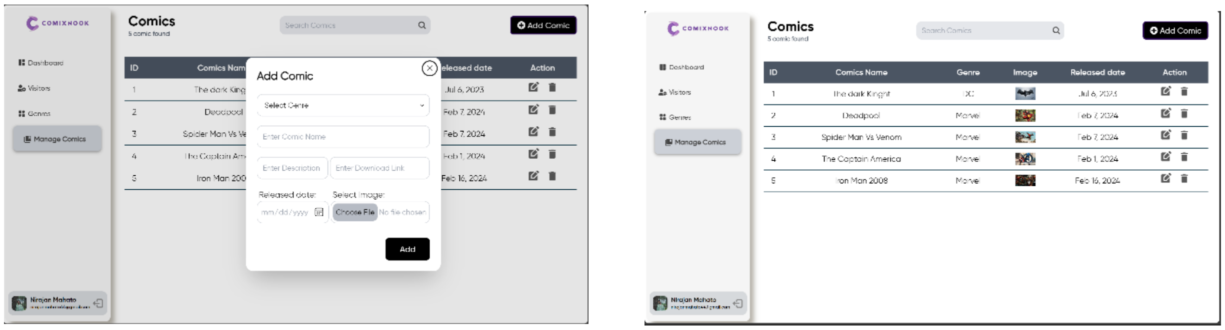


Genres
3 genre found

ID	Genre	Edit	Delete
1	Marvel		
2	DC		
4	Others		

The Managing Genre page is a useful tool for administrators to manage the genres available on the platform. This page enables administrators to add new genres, edit existing ones, and delete unnecessary genres. By having the ability to manage genres effectively, administrators can ensure that comics are organized and categorized in a user-friendly way, providing users with a streamlined browsing experience.

Figure 10:
Manage Comics Page

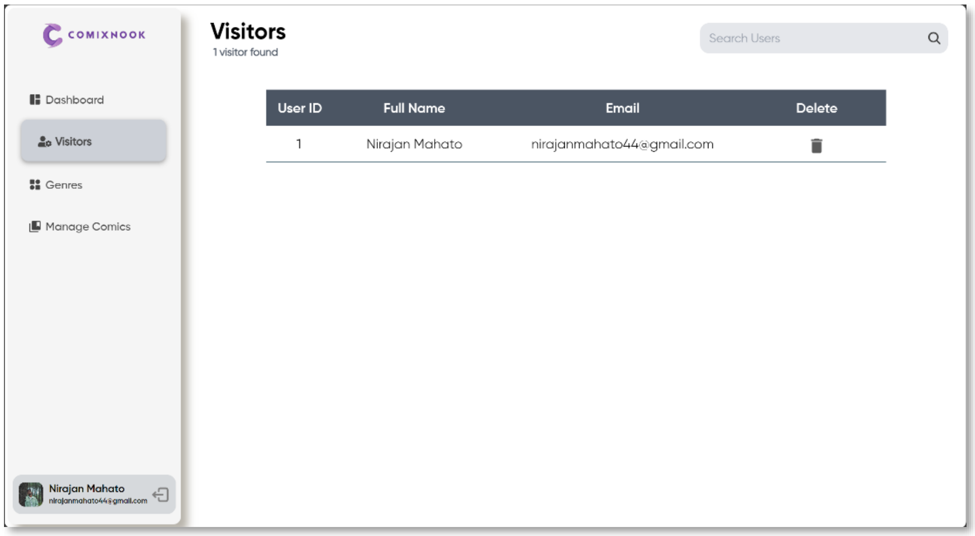


Comics
5 comic found

ID	Comics Name	Genre	Image	Released date	Action
1	The dark Knight	DC		Jul 6, 2023	
2	Deadpool	Marvel		Feb 7, 2024	
3	Spider Man Vs Venom	Marvel		Feb 7, 2024	
4	The Captain America	Marvel		Feb 1, 2024	
5	Iron Man 2008	Marvel		Feb 16, 2024	

The Manage Comics page enables administrators to supervise the comics accessible on the platform. They can upload new comics, modify comic details, and remove comics that may no longer be relevant or suitable. This page may also have features for categorizing comics, tagging them with relevant keywords, and managing comic licenses or copyrights.

Figure 11:
Manage User Page



Visitors
1 visitor found

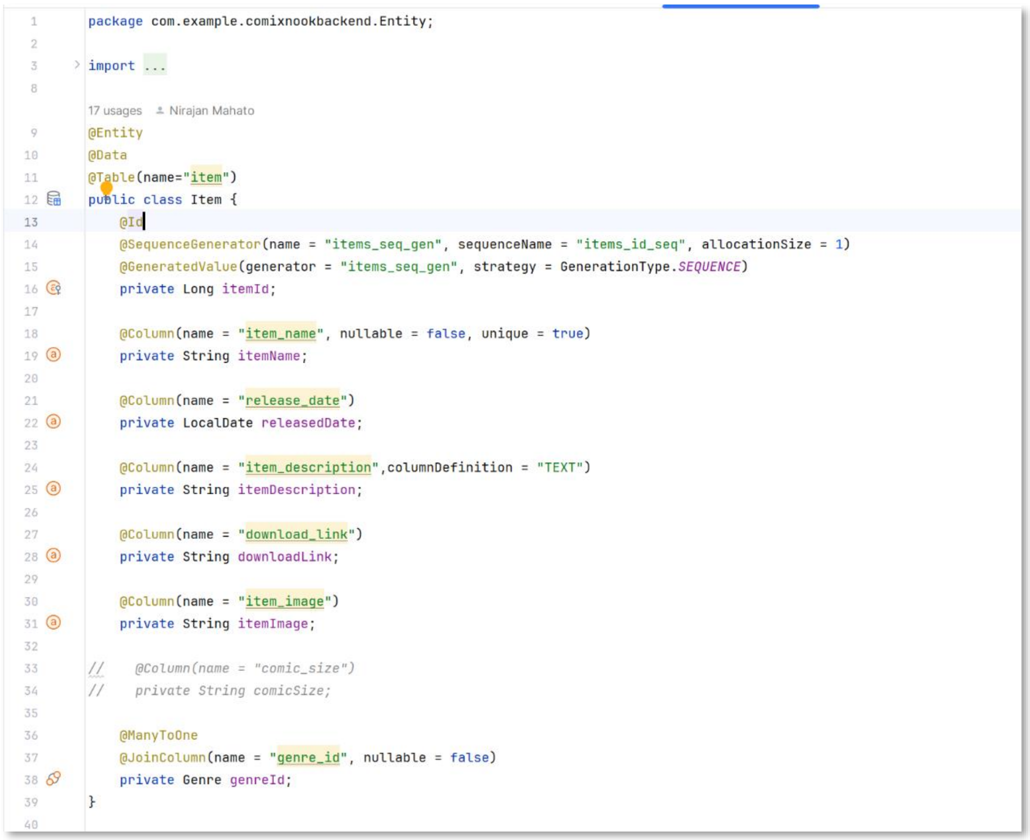
User ID	Full Name	Email	Delete
1	Nirajan Mahato	nirajanmahato44@gmail.com	🗑

On the Manage Users page, administrators can access user details and delete accounts as needed. This feature enables administrators to uphold the integrity of user data and address compliance or regulatory requirements in a streamlined manner. While other actions such as updating profiles or managing roles and permissions may be useful in the future, the current priority is to focus on essential user management tasks.

Back-End Technologies

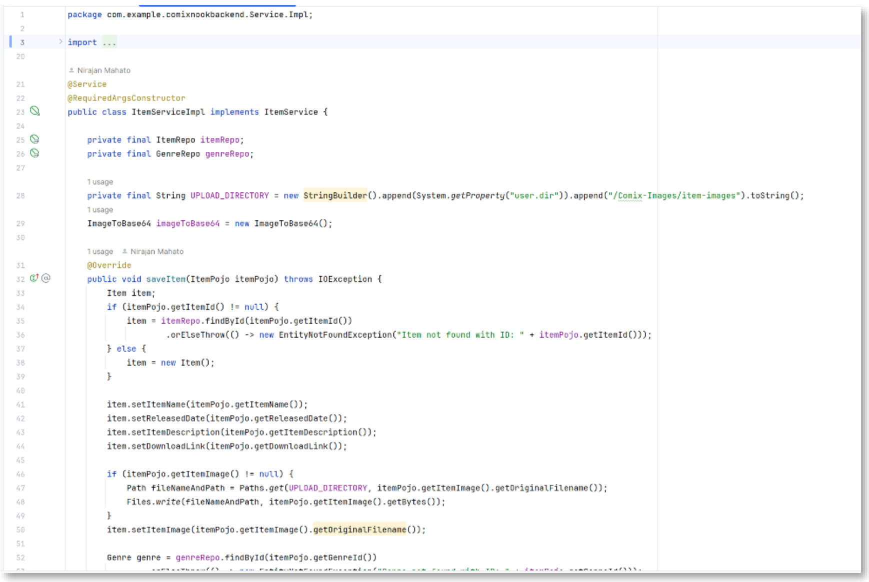
- **Spring Boot:** A framework for developing web applications in Java with a streamlined experience and simplified setup for Spring-based projects This application uses a structured and modular architecture with various components. Entity classes represent the domain model, controllers serve as the interface to external clients, POJOs encapsulate data, repositories abstract data access, and services contain business logic. Service implementations interact with repositories to manipulate data. Using these components, developers can build scalable, maintainable, and extensible backend applications in Spring Boot. *(Building Web Applications With Spring Boot and Kotlin)*

Figure 12:
Entity of Comic Item

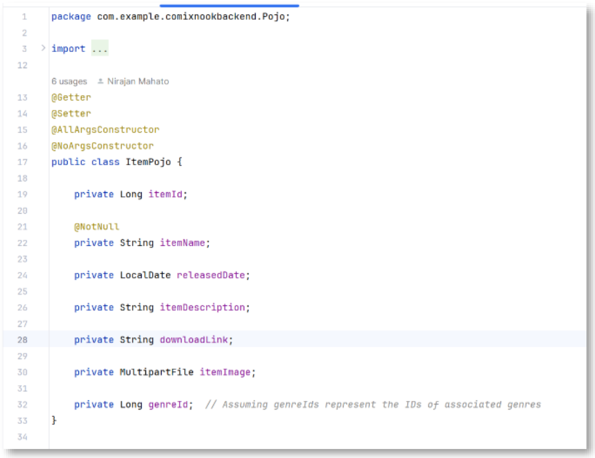


```
package com.example.comixnookbackend.Entity;

import ...

17 usages  Nirajan Mahato
@Entity
@Data
@Table(name="item")
public class Item {
    @Id
    @SequenceGenerator(name = "items_seq_gen", sequenceName = "items_id_seq", allocationSize = 1)
    @GeneratedValue(generator = "items_seq_gen", strategy = GenerationType.SEQUENCE)
    private Long itemId;

    @Column(name = "item_name", nullable = false, unique = true)
    private String itemName;

    @Column(name = "release_date")
    private LocalDate releasedDate;

    @Column(name = "item_description",columnDefinition = "TEXT")
    private String itemDescription;

    @Column(name = "download_link")
    private String downloadLink;

    @Column(name = "item_image")
    private String itemImage;

//    @Column(name = "comic_size")
//    private String comicSize;

    @ManyToOne
    @JoinColumn(name = "genre_id", nullable = false)
    private Genre genreId;
}
```

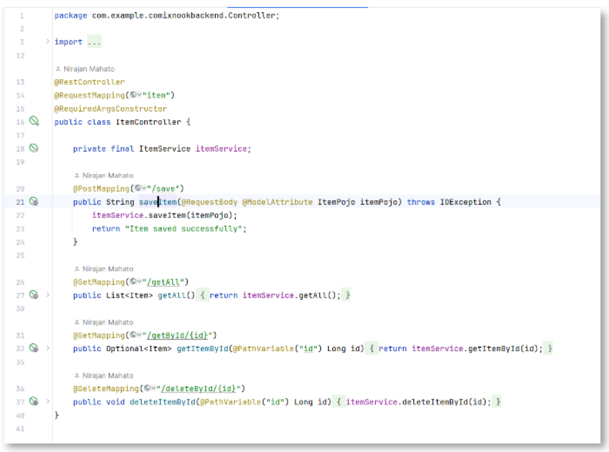
Figure 13:
Backend codes of Comic Item



```
package com.example.comixnookbackend.Service.Impl;

import ...

Nirajan Mahato
@Service
@RequiredArgsConstructor
public class ItemServiceImpl implements ItemService {

    private final ItemRepo itemRepo;
    private final GenreRepo genreRepo;

    1 usage
    private final String UPLOAD_DIRECTORY = new StringBuilder().append(System.getProperty("user.dir")).append("/Comix-Images/item-images").toString();
    1 usage
    ImageToBase64 imageToBase64 = new ImageToBase64();

    1 usage  Nirajan Mahato
    @Override
    public void saveItem(ItemPojo itemPojo) throws IOException {
        Item item;
        if (itemPojo.getItemId() != null) {
            item = itemRepo.findById(itemPojo.getItemId())
                    .orElseThrow(() -> new EntityNotFoundException("Item not found with ID: " + itemPojo.getItemId()));
        } else {
            item = new Item();
        }

        item.setItemName(itemPojo.getItemName());
        item.setReleasedDate(itemPojo.getReleasedDate());
        item.setItemDescription(itemPojo.getItemDescription());
        item.setDownloadLink(itemPojo.getDownloadLink());

        if (itemPojo.getItemImage() != null) {
            Path fileNameAndPath = Paths.get(UPLOAD_DIRECTORY, itemPojo.getItemImage().getOriginalFilename());
            Files.write(fileNameAndPath, itemPojo.getItemImage().getBytes());
        }
        item.setItemImage(itemPojo.getItemImage().getOriginalFilename());

        Genre genre = genreRepo.findById(itemPojo.getGenreId())
```



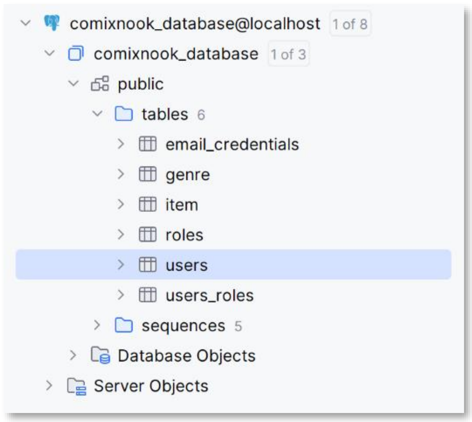
```
package com.example.comixnookbackend.Pojo;

import ...

6 usages  Nirajan Mahato
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class ItemPojo {

    private Long itemId;

    @NotNull
    private String itemName;

    private LocalDate releasedDate;

    private String itemDescription;

    private String downloadLink;

    private MultipartFile itemImage;

    private Long genreId;  // Assuming genreIds represent the IDs of associated genres
}
```



```
package com.example.comixnookbackend.Controller;

import ...

Nirajan Mahato
@RestController
@RequestMapping("item")
@RequiredArgsConstructor
public class ItemController {

    private final ItemService itemService;

    Nirajan Mahato
    @PostMapping("/save")
    public String saveItem(@RequestBody @ModelAttribute ItemPojo itemPojo) throws IOException {
        itemService.saveItem(itemPojo);
        return "Item saved successfully";
    }

    Nirajan Mahato
    @GetMapping("/getAll")
    public List<Item> getAll() { return itemService.getAll(); }

    Nirajan Mahato
    @GetMapping("/getById/{id}")
    public Optional<Item> getItemById(@PathVariable("id") Long id) { return itemService.getItemById(id); }

    Nirajan Mahato
    @DeleteMapping("/deleteById/{id}")
    public void deleteItemById(@PathVariable("id") Long id) { itemService.deleteItemById(id); }
}
```

- **PostgreSQL:** An open-source database management system for storing and managing data, providing scalability, reliability, and robust data integrity features.

Figure 14:
Tables of Backend

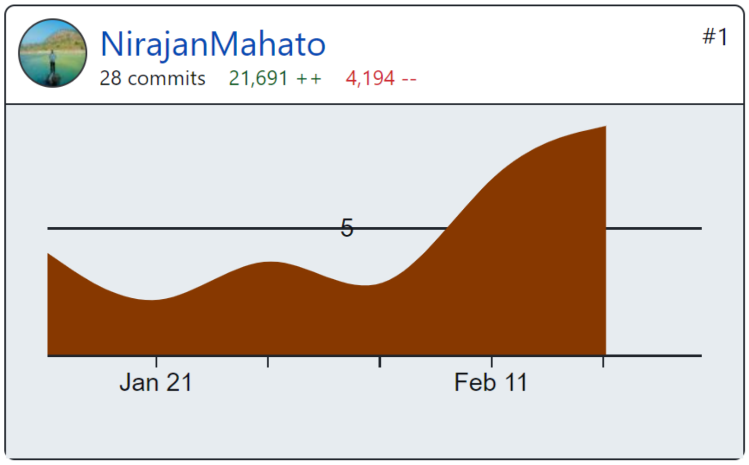


```
comixnook_database@localhost 1 of 8
  comixnook_database 1 of 3
    public
      tables 6
        email_credentials
        genre
        item
        roles
        users
        users_roles
      sequences 5
    Database Objects
  Server Objects
```

Version Control

- **GitHub:** A web-based platform that enables version control using Git, which facilitates collaborative development and code sharing among team members. This platform provides various features such as repository hosting, issue tracking, pull requests, and continuous integration, making it an essential tool for software development projects.

Figure 15:
GitHub

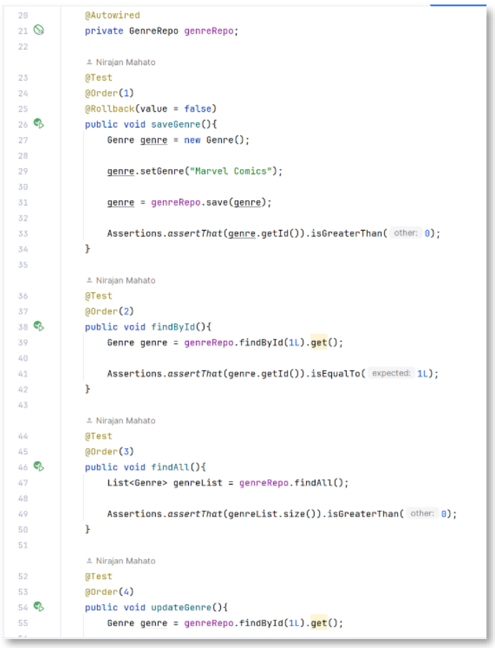


Testing

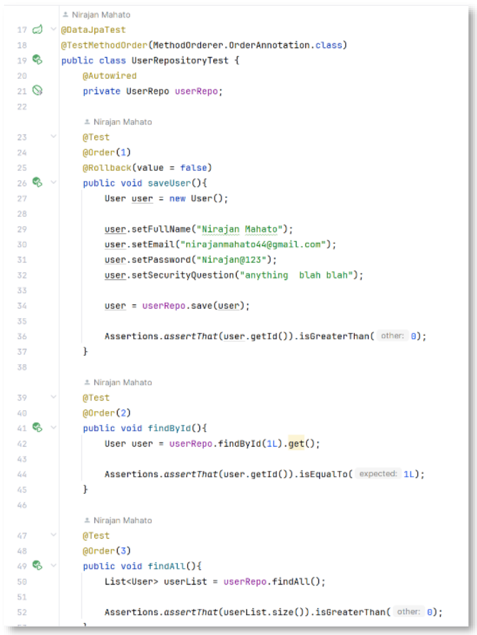
Unit Testing

Unit testing ensures individual code components work correctly in isolation. This test case makes use of JUnit 5 testing framework in conjunction with Spring Boot's testing annotations to ease integration testing with the underlying data layer. The @DataJpaTest annotation configures the test context to concentrate entirely on the JPA components such as repositories and entities, without loading the complete Spring application context. (Shah, 2023)

Figure 16:*jUnit Test*



```
    @Autowired
    private GenreRepo genreRepo;

    Nirajan Mahato
    @Test
    @Order(1)
    @Rollback(value = false)
    public void saveGenre(){
        Genre genre = new Genre();

        genre.setGenre("Marvel Comics");

        genre = genreRepo.save(genre);

        Assertions.assertThat(genre.getId()).isGreaterThan( other: 0);
    }

    Nirajan Mahato
    @Test
    @Order(2)
    public void findById(){
        Genre genre = genreRepo.findById(1L).get();

        Assertions.assertThat(genre.getId()).isEqualTo( expected: 1L);
    }

    Nirajan Mahato
    @Test
    @Order(3)
    public void findAll(){
        List<Genre> genreList = genreRepo.findAll();

        Assertions.assertThat(genreList.size()).isGreaterThan( other: 0);
    }

    Nirajan Mahato
    @Test
    @Order(4)
    public void updateGenre(){
        Genre genre = genreRepo.findById(1L).get();
```



```
Nirajan Mahato
@DataJpaTest
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
public class UserRepositoryTest {
    @Autowired
    private UserRepo userRepo;

    Nirajan Mahato
    @Test
    @Order(1)
    @Rollback(value = false)
    public void saveUser(){
        User user = new User();

        user.setFullName("Nirajan Mahato");
        user.setEmail("nirajanmahato44@gmail.com");
        user.setPassword("Nirajan@123");
        user.setSecurityQuestion("anything  blah blah");

        user = userRepo.save(user);

        Assertions.assertThat(user.getId()).isGreaterThan( other: 0);
    }

    Nirajan Mahato
    @Test
    @Order(2)
    public void findById(){
        User user = userRepo.findById(1L).get();

        Assertions.assertThat(user.getId()).isEqualTo( expected: 1L);
    }

    Nirajan Mahato
    @Test
    @Order(3)
    public void findAll(){
        List<User> userList = userRepo.findAll();

        Assertions.assertThat(userList.size()).isGreaterThan( other: 0);
```

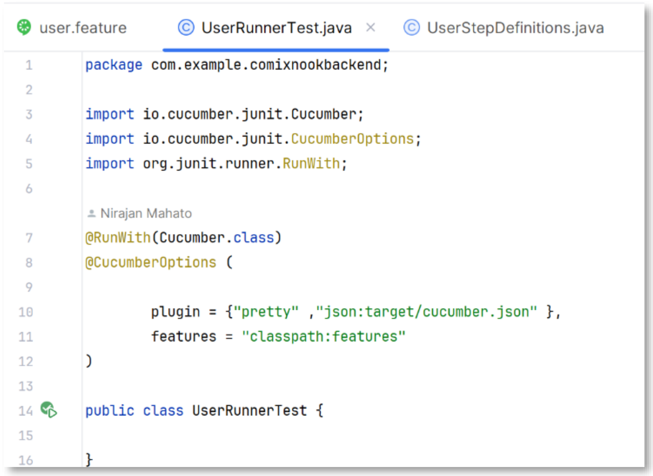
In each unit test, mock objects and dependency injection are utilized to isolate the component being tested from its dependencies, ensuring that tests are independent, reproducible, and deterministic. Assertions are then made to verify the expected behavior of the component under test. Overall, unit testing is an essential aspect of the development process, providing confidence in the correctness and robustness of the codebase, reducing the likelihood of bugs, and facilitating easier maintenance and refactoring in the future.

BDD Testing

The UserStepDefinitions is a suite of test cases implemented using Cucumber's BDD framework. These test cases are used to validate key user management functionalities in the backend system. The tests are executed in the Spring Boot application context using the SpringBootTest annotation, which simulates interactions with the backend API over HTTP.

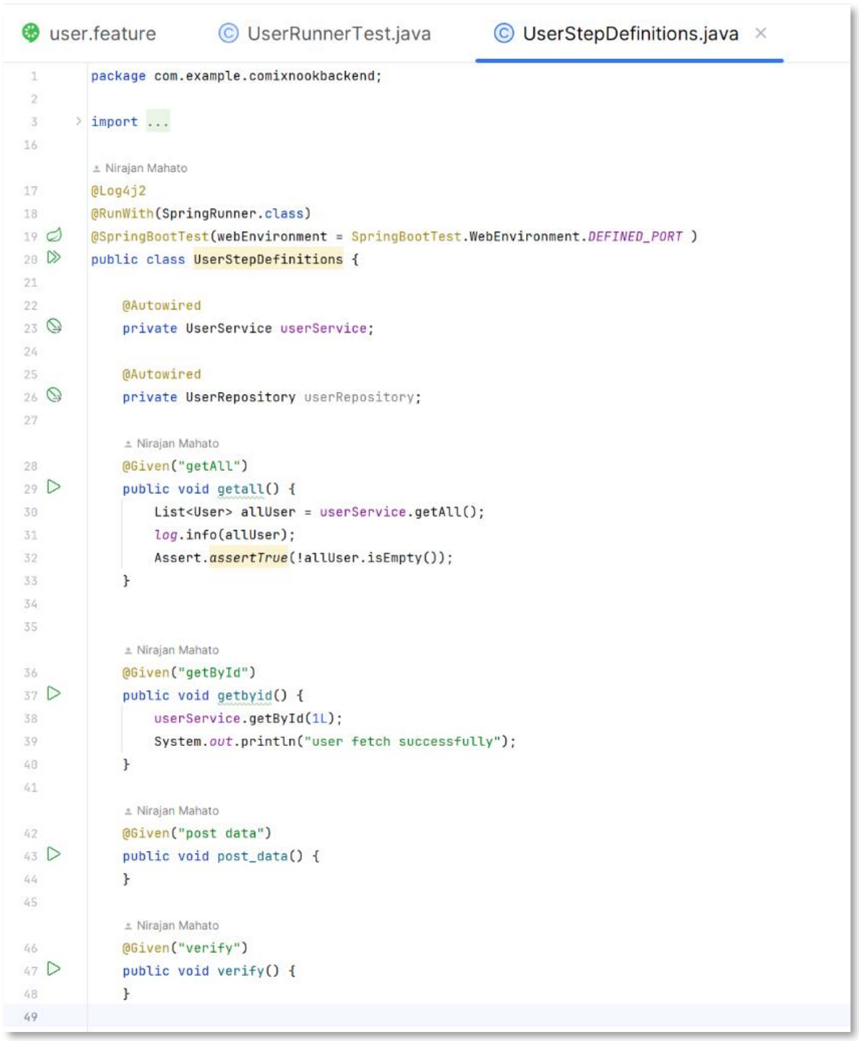
The UserStepDefinitions class relies on the UserService and UserRepository components to access user-related data and perform requisite operations. Each test case is designed to verify specific functionalities, such as validating the retrieval of all users from the database and fetching a user by their unique ID. These tests interpret Given-When-Then scenarios to provide comprehensive insights into the system behavior and ensure that user management functionalities meet the specified requirements. UserStepDefinitions is a pivotal component of the BDD testing strategy, as it strengthens the reliability and robustness of the backend system by rigorously validating user-related operations. (Christoph, 2021)

Figure 17:
UserRunnerTest

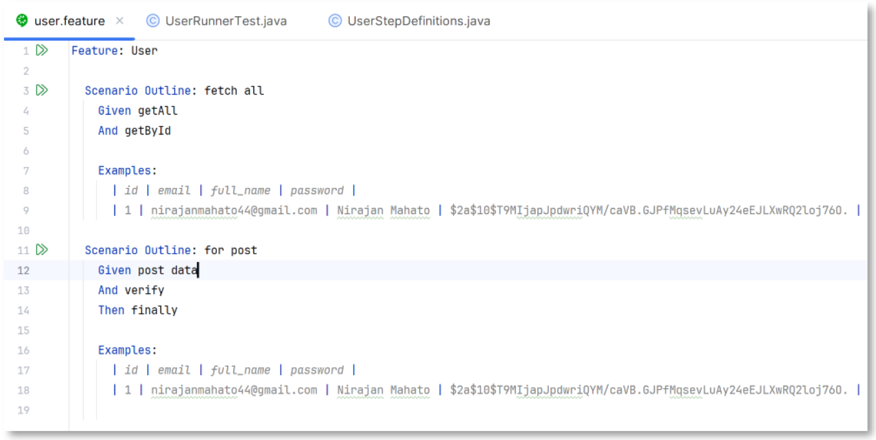


```
package com.example.comixnookbackend;

import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;
import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions (

        plugin = {"pretty" ,"json:target/cucumber.json" },
        features = "classpath:features"
)

public class UserRunnerTest {

}
```

Figure 18:
UserStepDefinitions


user.feature   UserRunnerTest.java   UserStepDefinitions.java

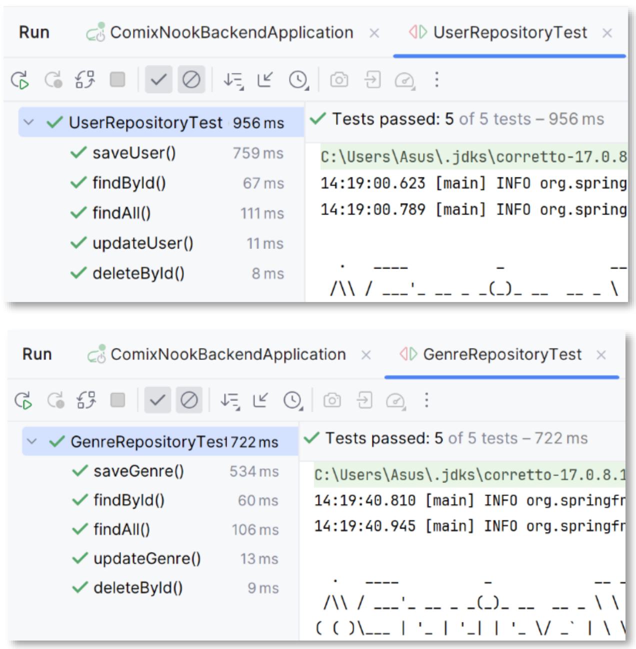
```java
package com.example.comixnookbackend;

import ...

Nirajan Mahato
@Log4j2
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.DEFINED_PORT )
public class UserStepDefinitions {

    @Autowired
    private UserService userService;

    @Autowired
    private UserRepository userRepository;

    Nirajan Mahato
    @Given("getAll")
    public void getall() {
        List<User> allUser = userService.getAll();
        log.info(allUser);
        Assert.assertTrue(!allUser.isEmpty());
    }


    Nirajan Mahato
    @Given("getById")
    public void getbyid() {
        userService.getById(1L);
        System.out.println("user fetch successfully");
    }

    Nirajan Mahato
    @Given("post data")
    public void post_data() {
    }

    Nirajan Mahato
    @Given("verify")
    public void verify() {
    }
```


user.feature   UserRunnerTest.java   UserStepDefinitions.java

```gherkin
Feature: User

  Scenario Outline: fetch all
    Given getAll
    And getById

    Examples:
      | id | email | full_name | password |
      | 1 | nirajanmahato44@gmail.com | Nirajan Mahato | $2a$10$T9MIjapJpdwriQYM/caVB.GJPfMqsevLuAy24eEJLXwRQ2loj760. |

  Scenario Outline: for post
    Given post data
    And verify
    Then finally

    Examples:
      | id | email | full_name | password |
      | 1 | nirajanmahato44@gmail.com | Nirajan Mahato | $2a$10$T9MIjapJpdwriQYM/caVB.GJPfMqsevLuAy24eEJLXwRQ2loj760. |
```

Test Result:

Figure 19:
Junit Test Result



```
UserRepositoryTest   956 ms
    saveUser()       759 ms
    findById()        67 ms
    findAll()        111 ms
    updateUser()      11 ms
    deleteById()       8 ms
```

Tests passed: 5 of 5 tests – 956 ms

```
C:\Users\Asus\.jdks\corretto-17.0.8
14:19:00.623 [main] INFO org.spring
14:19:00.789 [main] INFO org.spring
```

```
GenreRepositoryTest 722 ms
    saveGenre()      534 ms
    findById()        60 ms
    findAll()        106 ms
    updateGenre()     13 ms
    deleteById()       9 ms
```

Tests passed: 5 of 5 tests – 722 ms

```
C:\Users\Asus\.jdks\corretto-17.0.8.1
14:19:40.810 [main] INFO org.springfr
14:19:40.945 [main] INFO org.springfr
```

The above-mentioned unit test has passed, which means that it executed successfully without encountering any failures. To perform this test, JUnit 5 was used in conjunction with Spring Boot's testing annotations. The test focused solely on JPA components and ensured their correct behavior without loading the entire Spring context. By utilizing mock objects and dependency injection, the component under test was isolated, making testing independent, reproducible, and deterministic. Through assertions, expected behavior was confirmed, instilling confidence in the code's correctness and robustness. This successful outcome highlights the importance of unit testing in maintaining code quality and facilitating future maintenance efforts.

Issues Faced During Development

Issues Faced:

- **Frontend Design Challenges:** Creating an intuitive and visually appealing user interface for the frontend was challenging. Consistency across pages, responsive design, and balancing aesthetics with functionality were key considerations.
- **Prototyping in Figma:** Difficulties were also faced in using Figma to create interactive prototypes and refining design elements based on development requirements.
- **Login and Security:** Developing login and security features for this backend posed various challenges, such as implementing authentication mechanisms, data validation and sanitization, integrating with external systems, adhering to regulatory compliance standards, managing sessions securely, and implementing encryption techniques. These challenges were addressed by experts in security principles through meticulous testing and collaboration with cybersecurity professionals to ensure robust and secure features.

Constraints:

- **Time Constraints for Frontend Design:** Limited timeframes for frontend design limited the depth and breadth of design iterations. Prioritization and efficient utilization of design resources were necessary to balance design perfection with project deadlines.
- **Technical Constraints for Backend Security:** Ensuring comprehensive data security was challenging due to technical constraints in implementing secure authentication mechanisms and encryption techniques in the backend. Best practices, existing frameworks, and expert guidance were necessary to address these constraints.
- **Resource Constraints for Prototyping:** Limitations in expertise and resources hindered the exploration of advanced features in prototyping tools like Figma. Effective resource

utilization and streamlined prototyping workflows were crucial in overcoming these constraints.

Upcoming Features in the Future

Favorite Comics: Introduce a feature that allows users to mark comics as favorites, enabling them to easily access and revisit their preferred content. Users can create personalized collections of favorite comics for quick reference and enjoyment.

Notifications: Introduce a notification system that allows users to opt-in for updates on new comic releases and other important announcements via email or push notifications.

Read Comics Online: Implement a built-in comic reader with user-friendly features for an immersive reading experience, directly on the website without downloading.

Recommendation Engine: Introduce a recommendation engine that suggests comics based on users' preferences, reading history, and behavior patterns. Utilize machine learning algorithms to analyze user data and provide personalized recommendations, helping users discover new comics tailored to their interests.

User Profiles: Users can enhance their profiles with avatars, bios, background images, and track their activity. They can also view their reading history, favorite comics, and interactions with others.

Diagrams

Figure 20:
ER Diagram

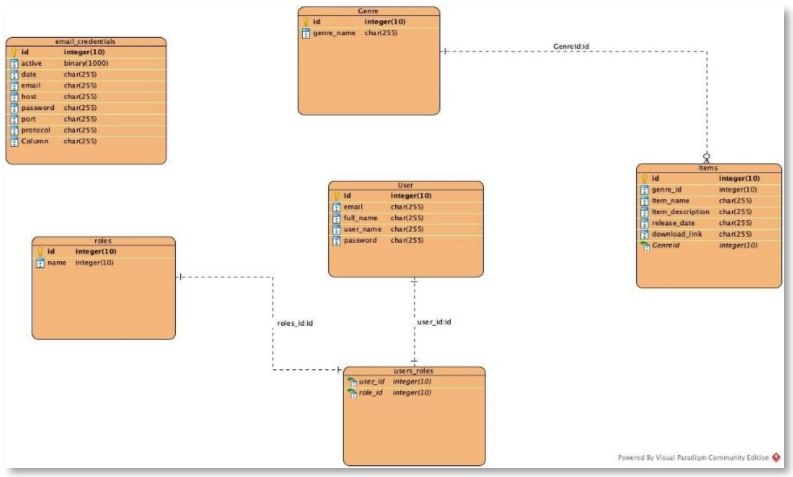
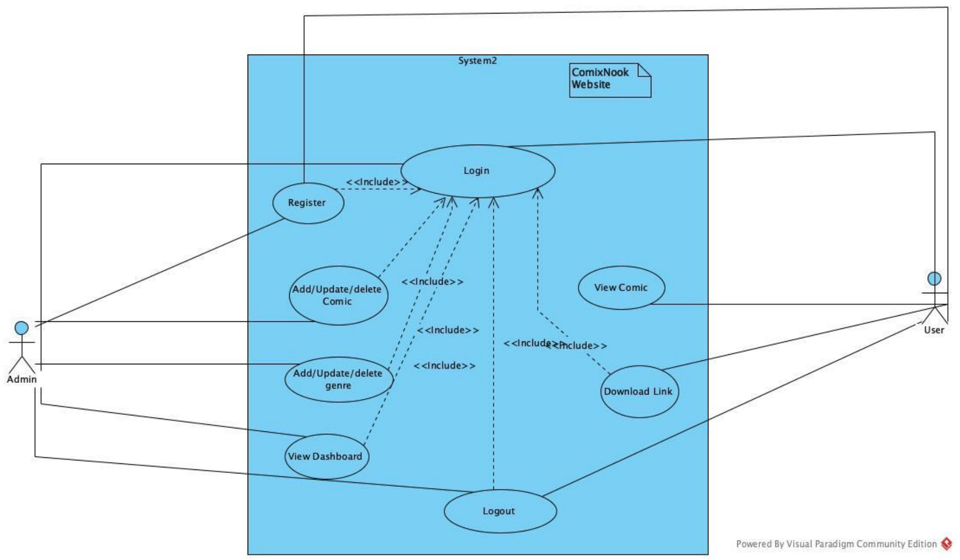


Figure 21:
Use Case Diagram



Conclusion

ComixNook is an easy-to-use online platform that offers a wide range of comics from different genres. It utilizes modern technologies such as React.js, Spring Boot, and PostgreSQL to provide a seamless browsing and downloading experience. With a responsive design and extensive testing, it ensures access across all devices. The project has provided valuable opportunities to learn frontend and backend development, unit testing, and database management. ComixNook aims to become the ultimate destination for comic enthusiasts worldwide by providing an inclusive and engaging experience.

Link

- **GitHub Link:** NirajanMahato/ComixNook-Web (github.com)
- **YouTube Link:** ComixNook(Online Comic Platform) | React.Js, Tailwind, SpringBoot, PostgreSQL (youtube.com)
- **Figma Link:** Untitled – Figma

References

- GetComics. (n.d.). *GetComics*. https://getcomics.org/
- *Marvel Comics | Marvel Comic Books | Marvel*. (n.d.). Marvel Entertainment. https://www.marvel.com/comics?&options%5Boffset%5D=0&totalcount=12
- Viewcomics. (n.d.). *New comic releases | Read comics online for free*. Viewcomics. https://readcomics.top/new-comics
- Comicextra. (n.d.). *Read Comics Online - Read comics online in high quality*. Viewcomics. https://comicextra.me/
- *Welcome: Login page*. (n.d.). Dribbble. https://dribbble.com/shots/20512066-Welcome-Login-Page
- *Bonita dashboard*. (n.d.). Dribbble. https://dribbble.com/shots/23653591-Bonita-Dashboard
- *Getting Started | Building web applications with Spring Boot and Kotlin*. (n.d.). Getting Started | Building Web Applications With Spring Boot and Kotlin. https://spring.io/guides/tutorials/spring-boot-kotlin
- Shah, H. (2023, June 16). How to do Unit Testing in Test Driven Development(TDD)? *Simform - Product Engineering Company*. https://www.simform.com/blog/unit-testing-tdd/
- Christoph. (2021, July 7). *Behavior driven development – from End-User to Unit Tests*. BDD Framework for NET. https://specflow.org/bdd/end-user-unit-tests/