Read My Novel - An Online Reading and Writing Platform

I. Project description

The most important thing for any writers is not how much money their book makes, but how many people have read it and truly enjoyed their work. If you are trying to write a novel and would like some unofficial "beta" readers or if you have published a short story, it is not a bad idea to post your work on a website that has a devoted readership. Reading and writing communities can be a great way to get feedback on your writing. They often have various criteria to evaluate each writing in order to find the most popular one, which are then publicized. For example, some of the large sites such as Wattpad and Authonomy have cooperative links with social media, publishers, or in the case of WEbook, there is a service that helps writers contact directly to agents. Each community offers different things, however, they all have in common is to provide an online platform for writer's works to get closer to the public, make them available to thousands, if not millions, of readers.

This project aims at developing an online reading and writing platform, called Read My Novel. Through this website, authors can write books by posting chapters and choose to publish their writing to public community. Each user can read some book for free or buy each chapter of a book to read if he/she is interested. Moreover, users can write book reviews, comment on chapters or vote for their favorite books.

II. Member assignment

- Lê Minh Hải Phong – 20170221: 40%
  - Usecase: Manage user's library, Evaluate book, Read book.
  - Database management
- Lê Hồng Minh – 20176820: 30%
  - Manage own account usecase
  - Manage user's book usecase
- Nguyễn Bình Long – 20176807: 30%
  - Browse book usecase
  - System and UI design
  - Develop frontend

III. Usecase diagram

3.1 General usecase diagram

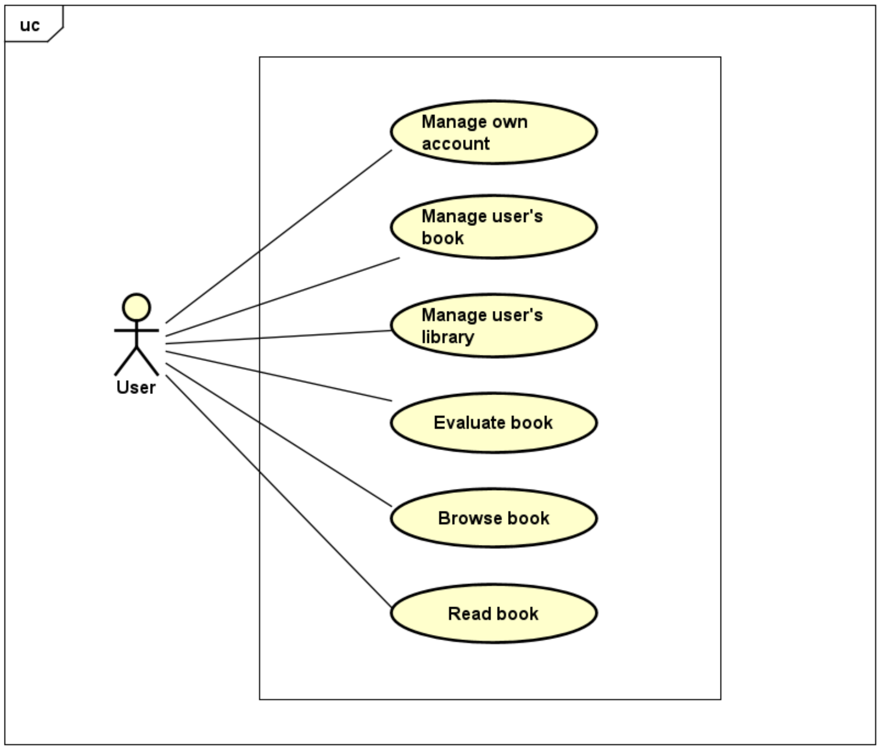


Figure 1: General usecase

3.2 Manage own account

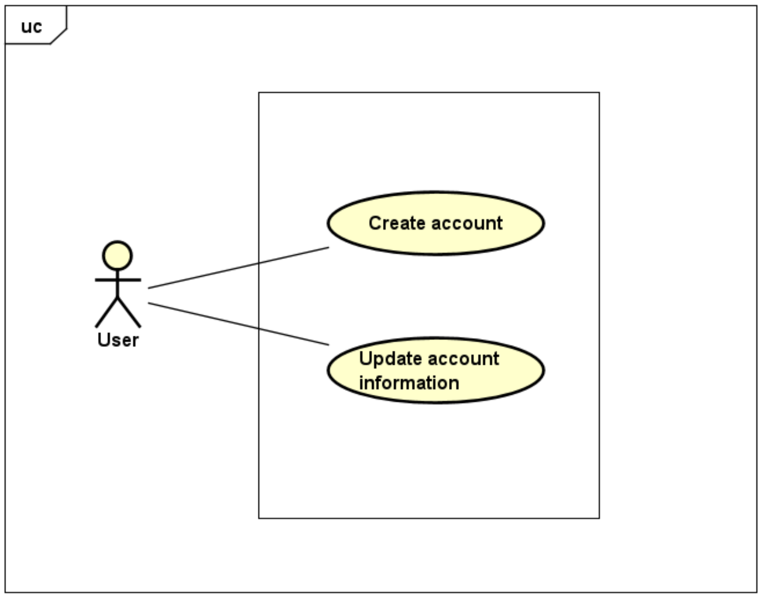


Figure 2: Manage own account usecase

3.3 Manage user’s book

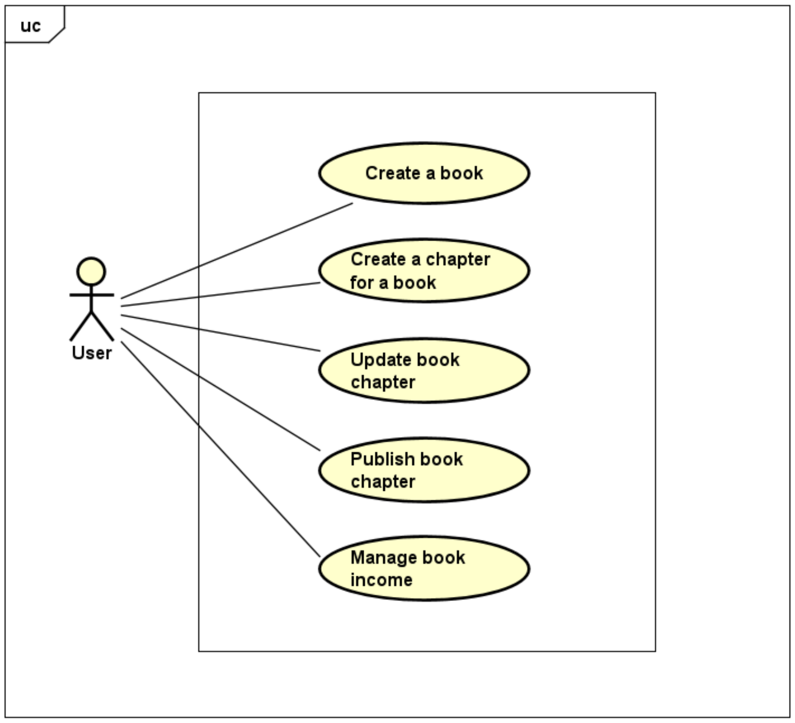


Figure 3: Manage user's book usecase

3.4 Manage user’s library

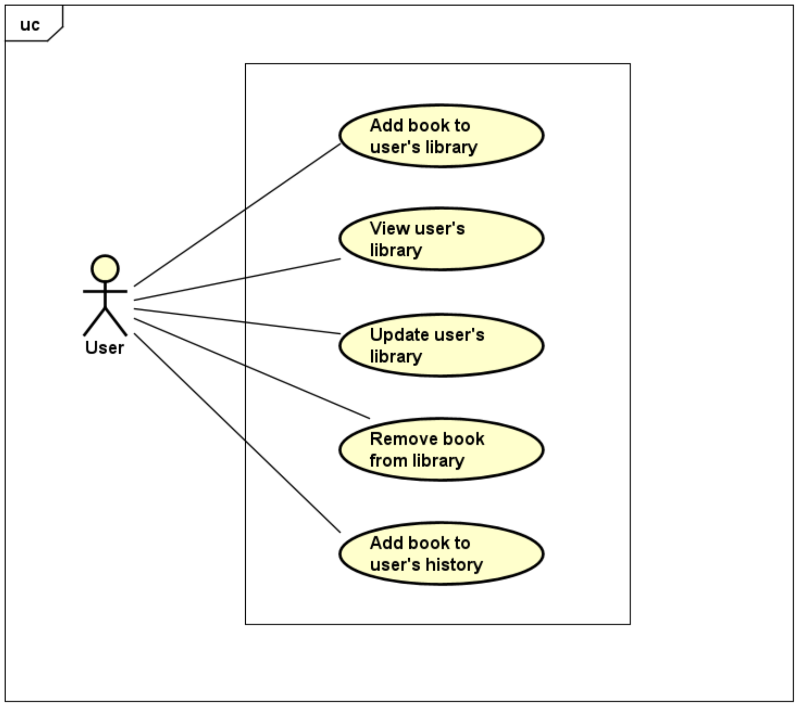


Figure 4: Manage user's library usecase

3.5 Read book

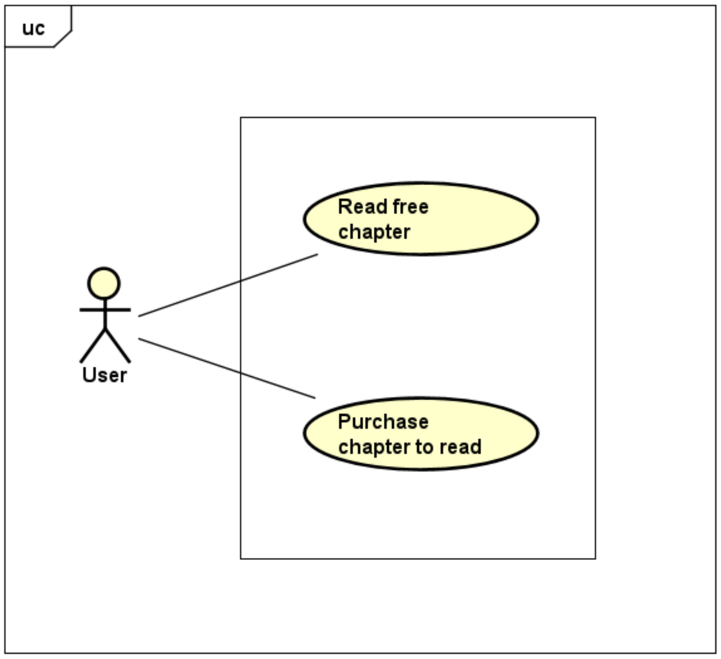


Figure 5: Read book usecase

3.6 Browse book

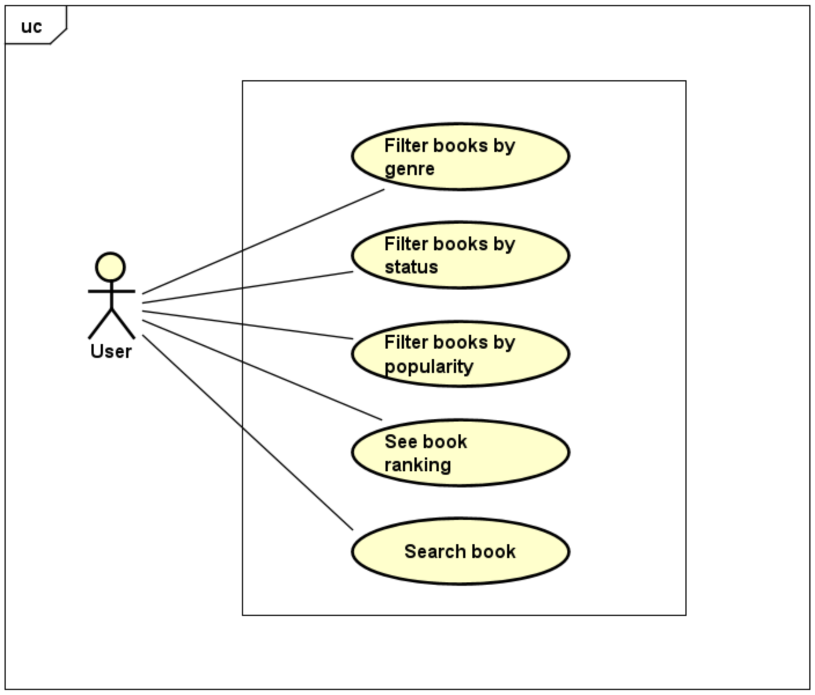


Figure 6: Browse book usecase

3.7 Evaluate book

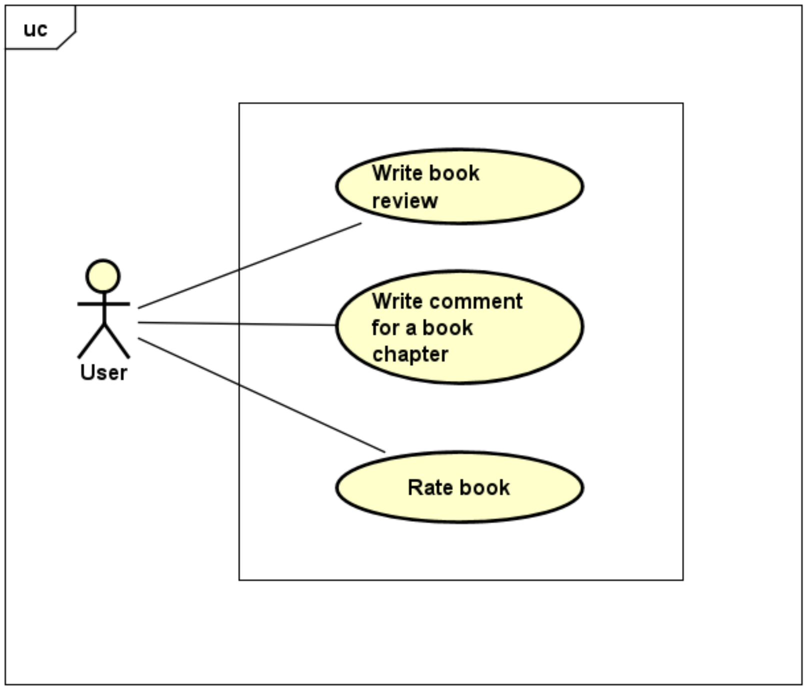


Figure 7: Evaluate book usecase

- There are only some decomposed usecases which have not been implemented such as
  - Update account information
  - Manage book income
  - Purchase chapter to read

IV. UI design

- The website is responsive, which can run on desktop screen (screens equal to or greater than 1200px wide) and display well on mobile screens (screens less than 768px wide)
- The design of UI is based on https://www.webnovel.com/
- The UI of homepage with header, footer is as follows:
  - For desktop screen:

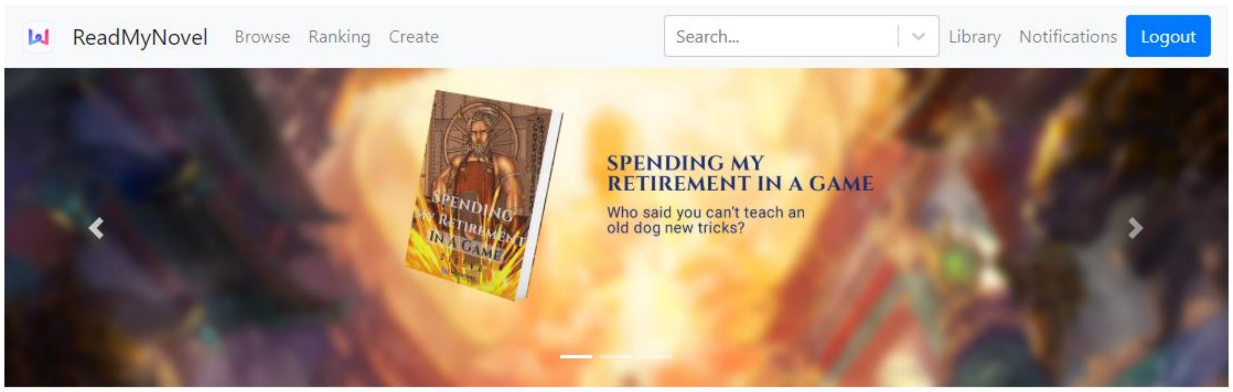

ReadMyNovel
Browse
Ranking
Create
Search...
Library
Notifications
Logout
SPENDING MY RETIREMENT IN A GAME
Who said you can't teach an old dog new tricks?

Figure 8: Header and navbar in desktop mode

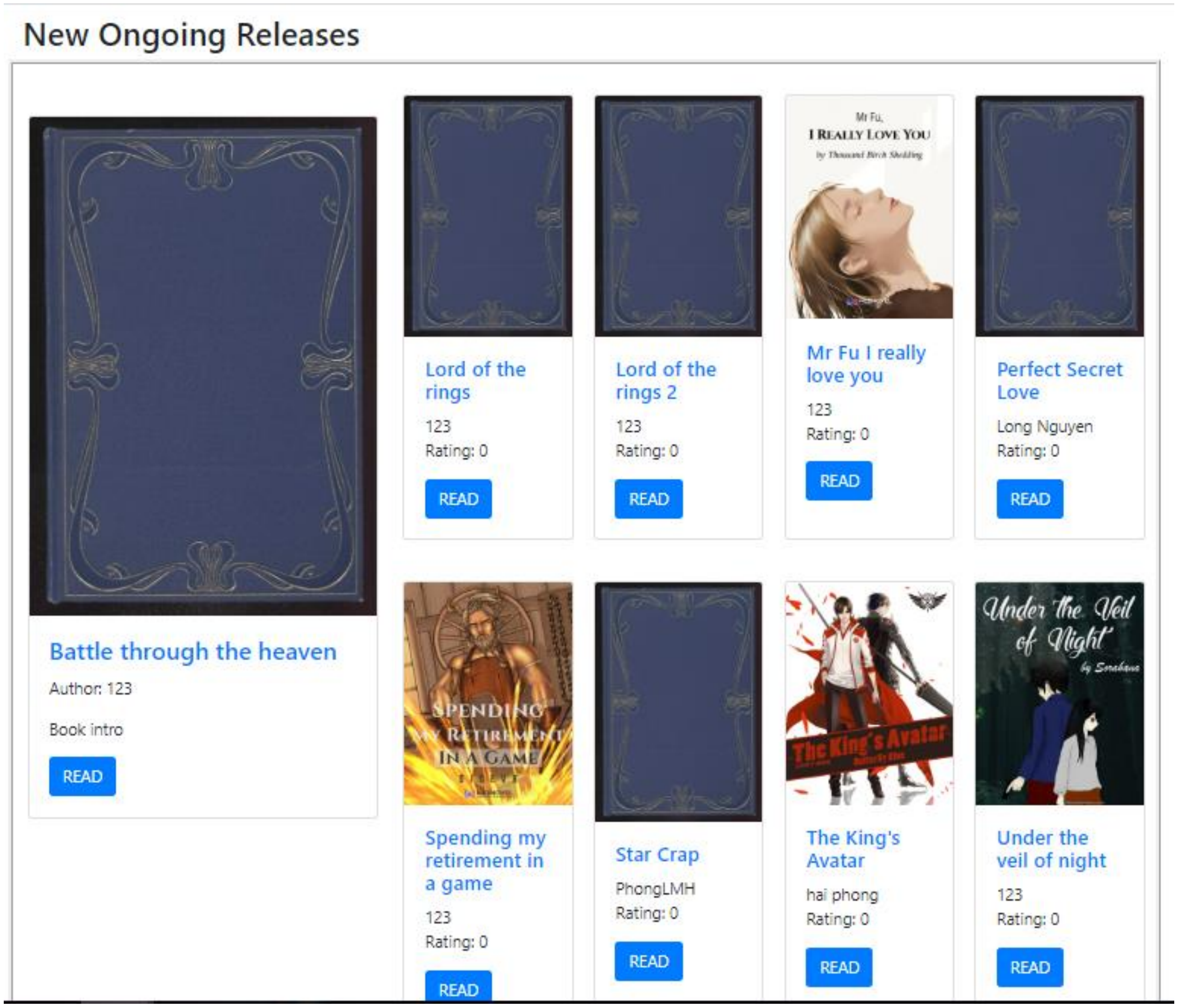
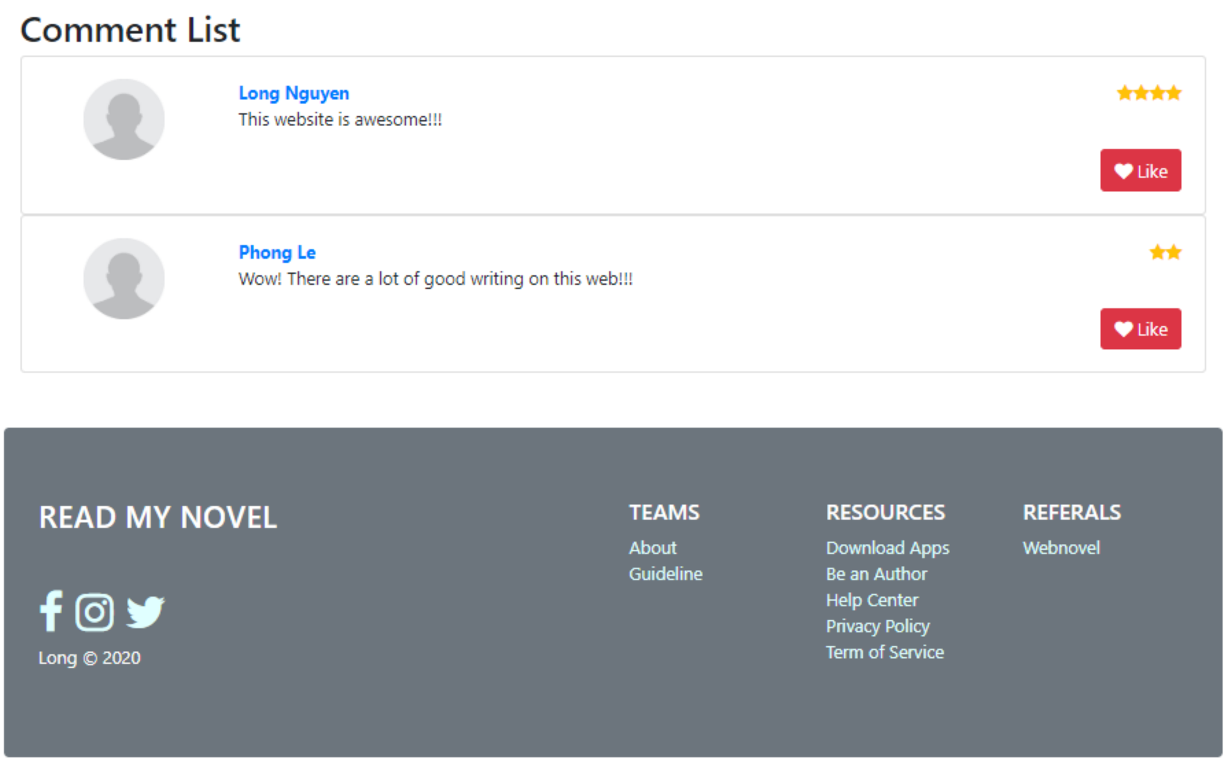


Figure 9: A part of main homepage in desktop mode


Comment List

Long Nguyen
This website is awesome!!!

Phong Le
Wow! There are a lot of good writing on this web!!!

READ MY NOVEL

Long © 2020

TEAMS	RESOURCES	REFERALS
About	Download Apps	Webnovel
Guideline	Be an Author	
	Help Center	
	Privacy Policy	
	Term of Service	

Figure 10: Footer in desktop mode

- For phone screen:

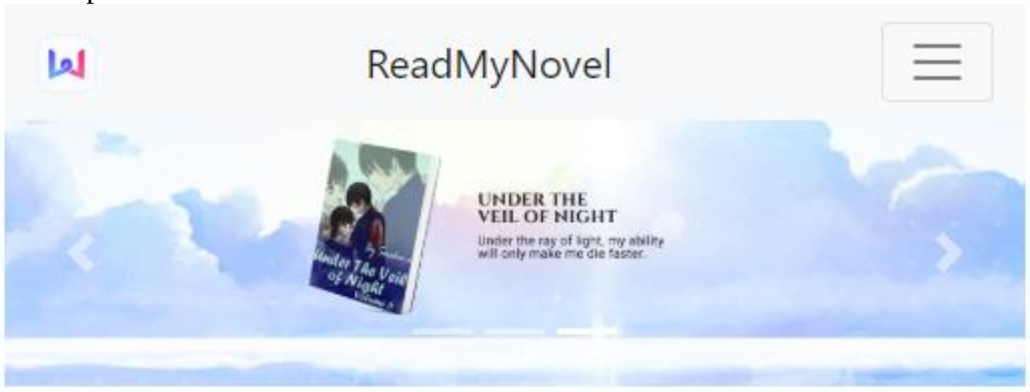

ReadMyNovel
UNDER THE VEIL OF NIGHT
Under the ray of light, my ability will only make me die faster.

Figure 11: Header in phone mode

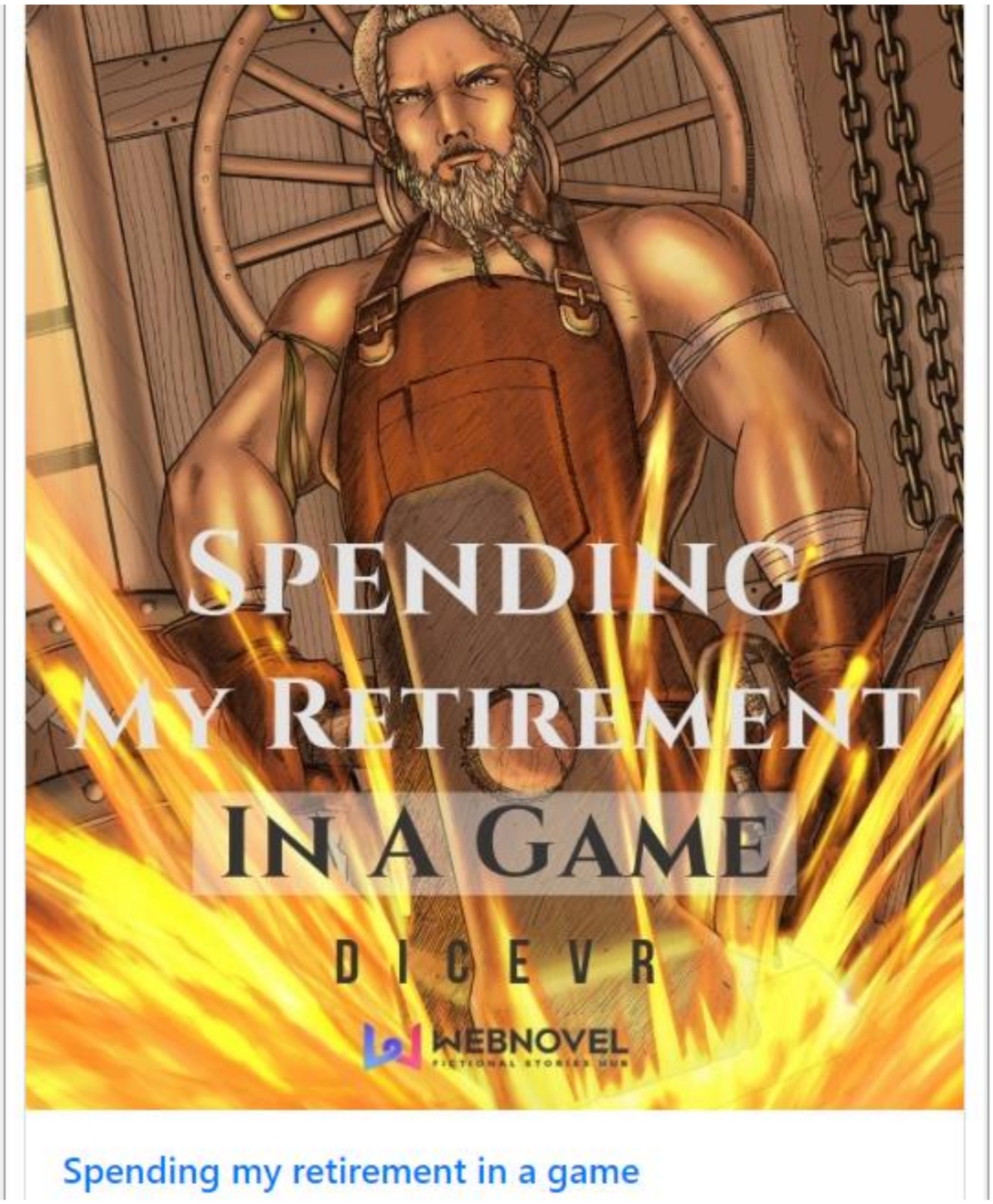
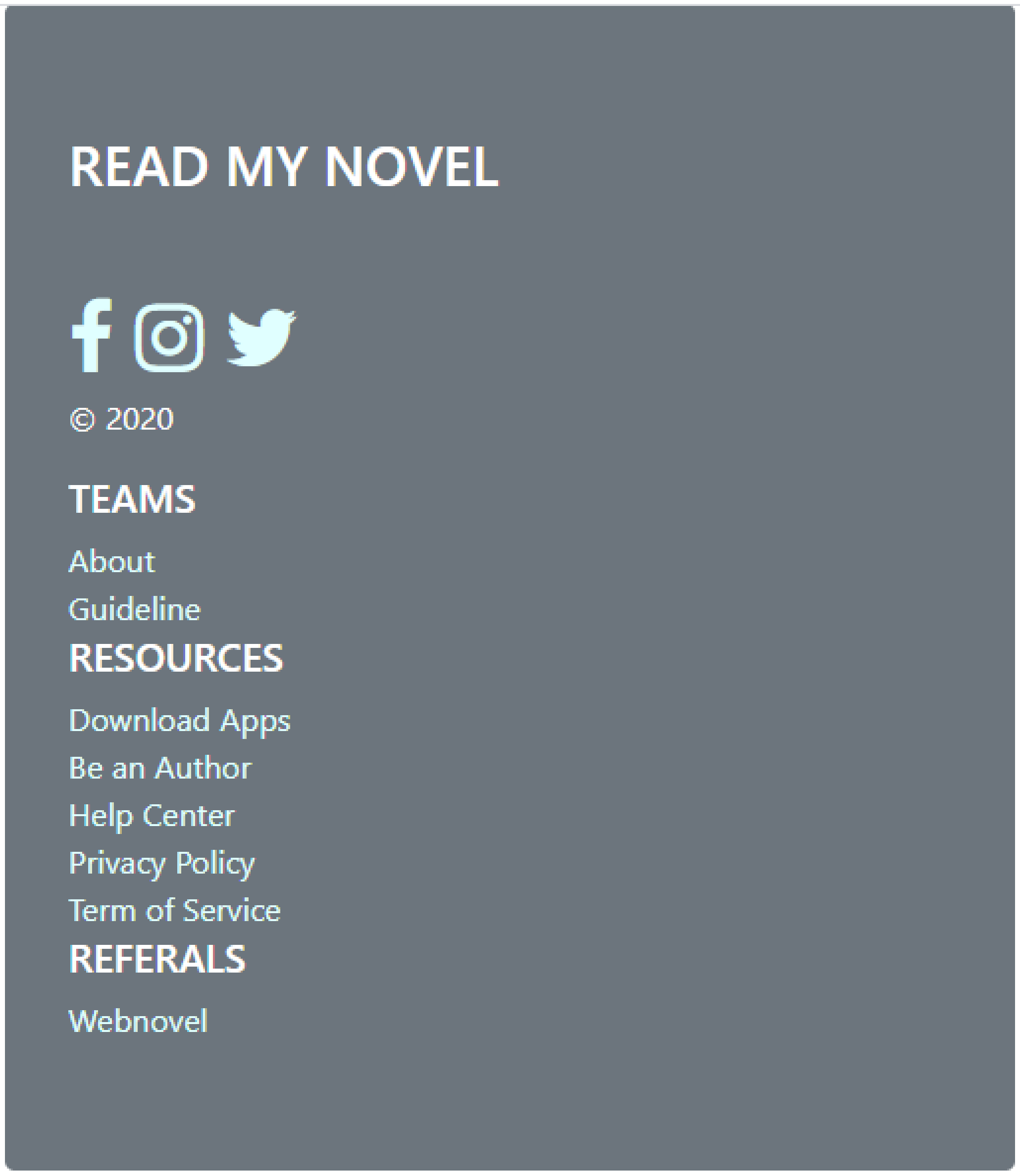


Figure 12: A part of home page in phone mode



READ MY NOVEL

© 2020

TEAMS

About

Guideline

RESOURCES

Download Apps

Be an Author

Help Center

Privacy Policy

Term of Service

REFERALS

Webnovel

Figure 13: Footer in phone mode

V. Database design

- Relational Schema:

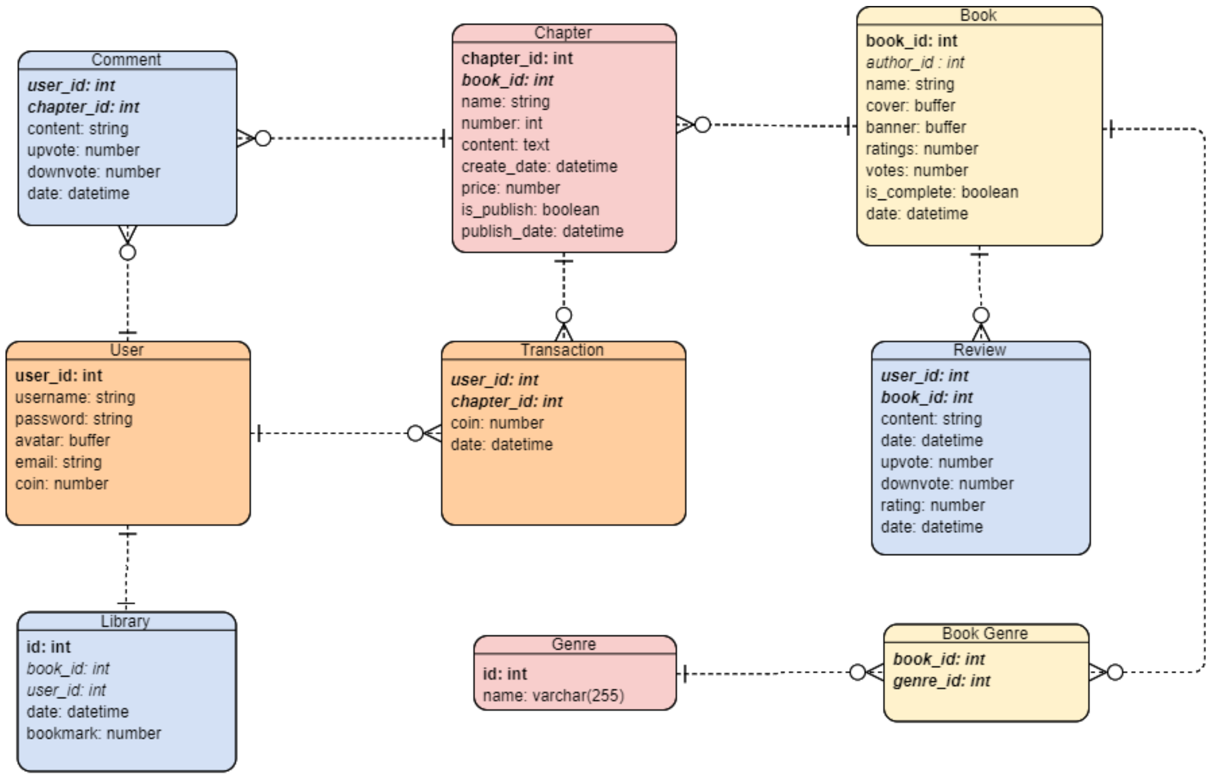
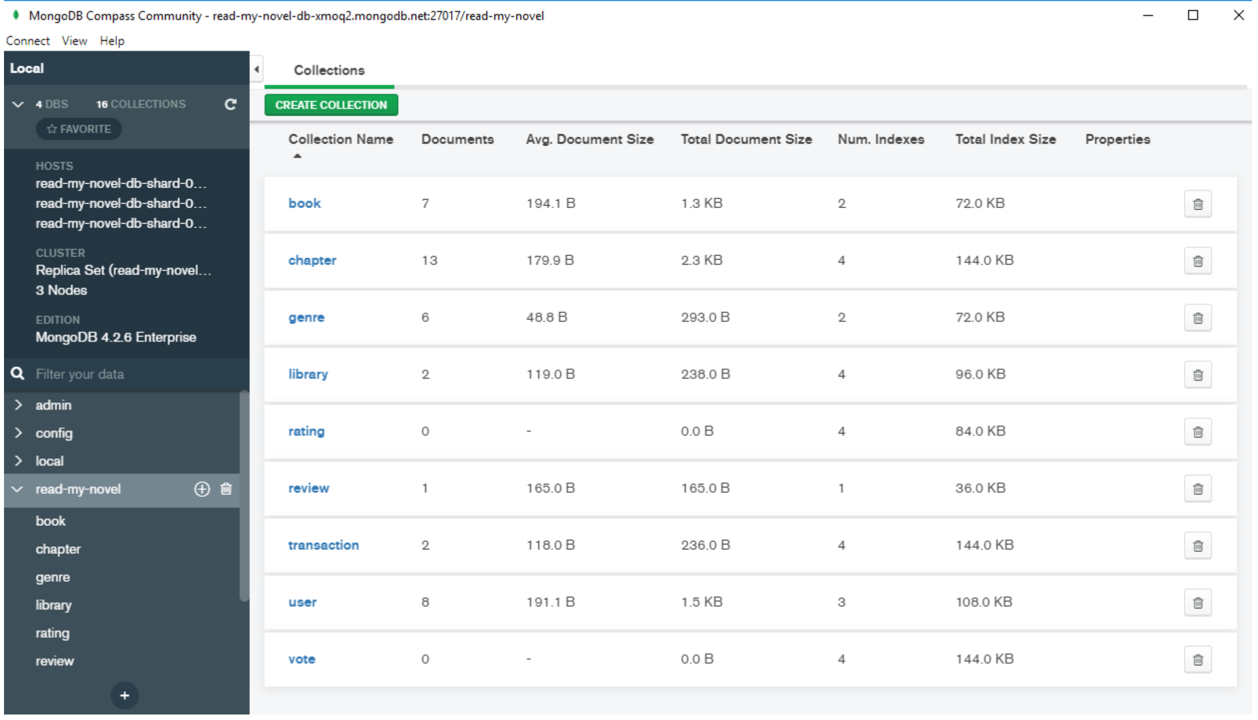


Figure 14: Database Relational Schema

- Database management system used: MongoDB, which is an open-source, document database designed for ease of development and scaling. Mongoose is a client API for node.js which makes it easy to access our database from our Express application.



Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size	Properties
book	7	194.1 B	1.3 KB	2	72.0 KB	
chapter	13	179.9 B	2.3 KB	4	144.0 KB	
genre	6	48.8 B	293.0 B	2	72.0 KB	
library	2	119.0 B	238.0 B	4	96.0 KB	
rating	0	-	0.0 B	4	84.0 KB	
review	1	165.0 B	165.0 B	1	36.0 KB	
transaction	2	118.0 B	236.0 B	4	144.0 KB	
user	8	191.1 B	1.5 KB	3	108.0 KB	
vote	0	-	0.0 B	4	144.0 KB	

VI. Technology stacks

6.1 React

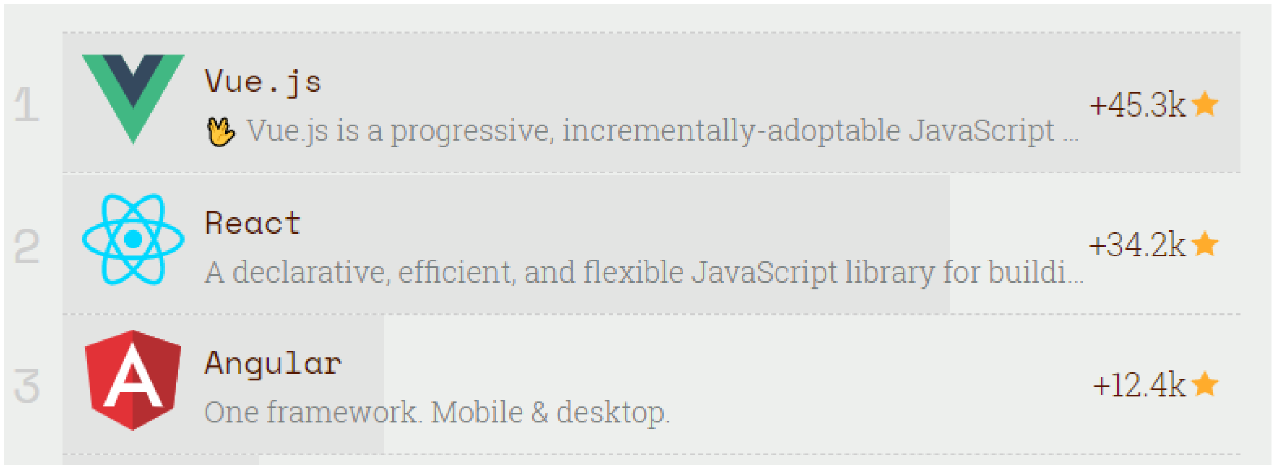
- React is a JavaScript library that aims to simplify development of visual interfaces. Developed by Facebook and released to the world in 2013, it drives some of the most widely used code in the world, powering Facebook and Instagram among many, many other software companies. Its primary goal is to make it easy to reason about an interface and its state in any point in time, by dividing the UI into a collection of components. React is used to build single-page web applications, among with many other libraries and frameworks that were available before React came into life.
- React is really popular nowadays and has taken the frontend web development world by storm. Firstly, React is less complex than other alternatives. At the time when React was announced, Ember.js and Angular 1.x were the predominant choices as a framework. Both these imposed too many conventions on the code that porting an existing app was not convenient at all. React made a choice to be very easy to integrate into an existing project, because that's how they had to do it at Facebook in order to introduce it to the existing codebase. Also, those 2 frameworks brought too much to the table, while React only chose to implement the View layer instead of the full MVC stack. React encourages the creation of reusable UI components, which present data that changes over time. React abstracts away the DOM from you, offering a simpler programming model and better performance. React can also render on the server using Node, and it can power native apps using React Native. React implements one-way reactive data flow, which reduces the boilerplate and is easier to reason about than traditional data binding

- Popularity of React:

According to the *2019 Stack Overflow Report*, JS is used by 67.8% of developers with "Big Three" of JS frameworks in 2019: React, Angular and Vue.

1. **GitHub Stars**

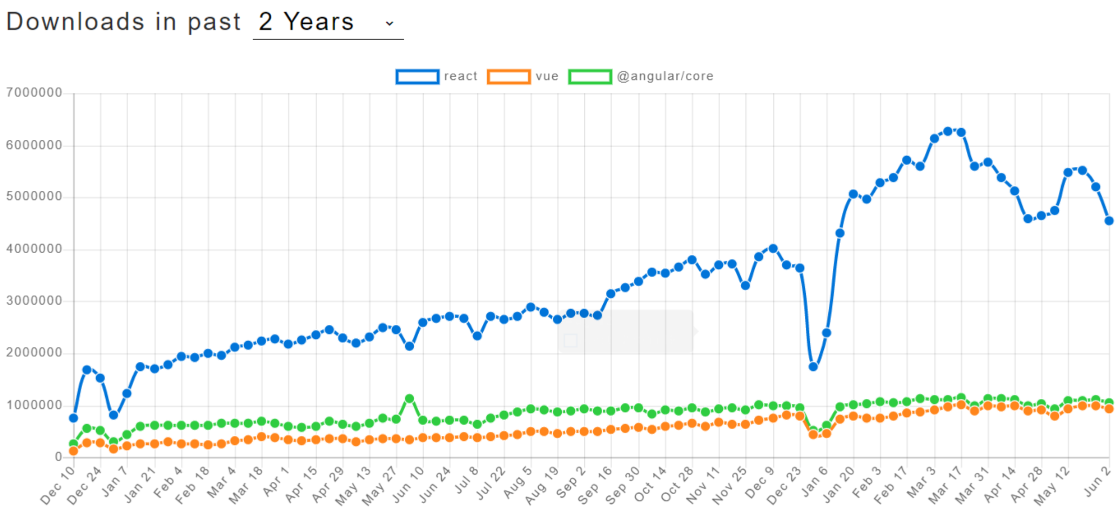
The *JavaScript Rising Stars report*, which tracks the number of stars added on GitHub, listed Vue, React and Angular as the most popular JS frameworks in 2018, outpacing the other frameworks by a lot:



1	Vue.js Vue.js is a progressive, incrementally-adoptable JavaScript ...	+45.3k ★
2	React A declarative, efficient, and flexible JavaScript library for buildi...	+34.2k ★
3	Angular One framework. Mobile & desktop.	+12.4k ★

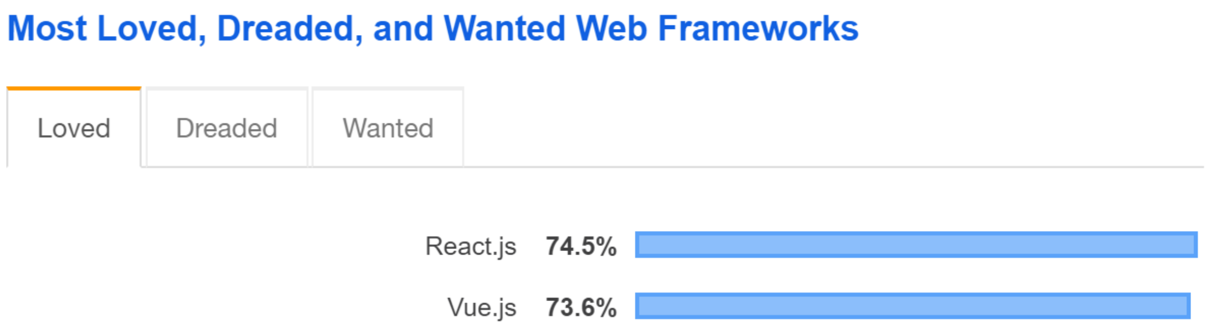
2. Program Downloads

Unlike GitHub stars, the number of downloads show React in the lead in terms of sheer bulk of use. These downloads are a good indicator of what developers are actually using, instead of hot trends. The most surprising part of this data? Vue comes in last, after React and Angular:

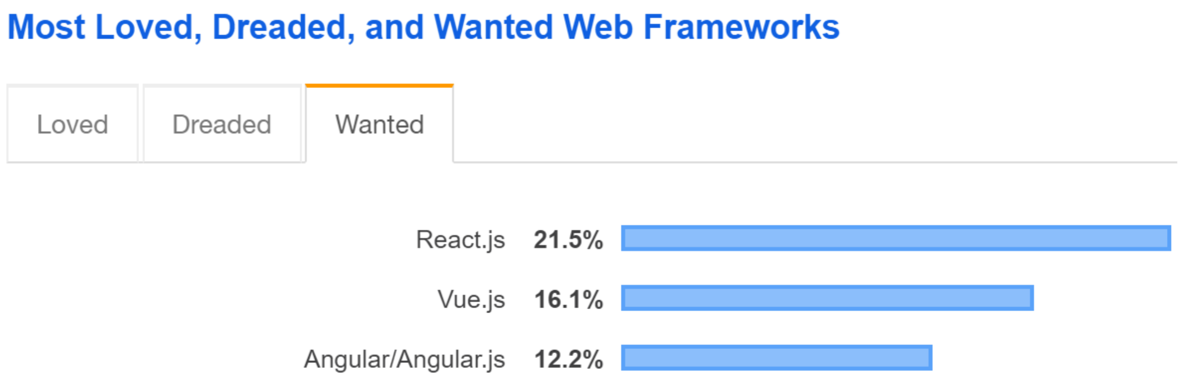


3. Most Loved and Wanted Frameworks

The 2019 Stack Overflow Developer Survey also measured what frameworks developers love versus which are most wanted. Like GitHub stars, this shows how developers feel about these frameworks. The results aren’t too surprising:



React and Vue were head-to-head in the most loved category, while all three appeared in the top most wanted frameworks, with React in the lead with 21.5%.



- React Features:

4. JSX − JSX is JavaScript syntax extension. It isn't necessary to use JSX in React development, but it is recommended.
5. Components − React is all about components. You need to think of everything as a component. This will help you maintain the code when working on larger scale projects.
6. Unidirectional data flow and Flux − React implements one-way data flow which makes it easy to reason about your app. Flux is a pattern that helps keeping your data unidirectional.
7. License − React is licensed under the Facebook Inc. Documentation is licensed under CC BY 4.0.

- React Advantages:

1. Uses virtual DOM which is a JavaScript object. This will improve apps performance, since JavaScript virtual DOM is faster than the regular DOM.
2. Can be used on client and server side as well as with other frameworks.
3. Component and data patterns improve readability, which helps to maintain larger apps.

- React Limitations:
  1. Covers only the view layer of the app, hence you still need to choose other technologies to get a complete tooling set for development.
  2. Uses inline templating and JSX, which might seem awkward to some developers.

6.2 Redux

- Redux is a predictable state container for JavaScript apps. As the application grows, it becomes difficult to keep it organized and maintain data flow. Redux solves this problem by managing application's state with a single global object called Store. Redux fundamental principles help in maintaining consistency throughout your application, which makes debugging and testing easier. More importantly, it gives you live code editing combined with a time-travelling debugger. It is flexible to go with any view layer such as React, Angular, Vue, etc.
- Principles of Redux: Predictability of Redux is determined by three most important principles as given below

1. Single Source of Truth: The state of your whole application is stored in an object tree within a single store. As whole application state is stored in a single tree, it makes debugging easy, and development faster.
2. State is Read-only: The only way to change the state is to emit an action, an object describing what happened. This means nobody can directly change the state of your application.
3. Changes are made with pure functions: To specify how the state tree is transformed by actions, you write pure reducers. A reducer is a central place where state modification takes place. Reducer is a function which takes state and action as arguments, and returns a newly updated state

6.3 Express

- ExpressJS is a web application framework that provides you with a simple API to build websites, web apps and back ends. Express is a minimalist and extensible web framework built for the Node.js ecosystem. It enables you to create a web server that is more readable, flexible, and maintainable than you would be able to create using only the Node HTTP library, which can get verbose and complicated for even the most basic web servers. With ExpressJS, you need not worry about low level protocols, processes, etc. Express is going to make creating a web server much easier!

- Express provides a minimal interface to build our applications. It provides us the tools that are required to build our app. It is flexible as there are numerous modules available on *npm*, which can be directly plugged into Express. Express was developed by TJ Holowaychuk and is maintained by the Node.js foundation and numerous open source contributors.
- Setup

- Make a new directory to hold this project, and then change into that directory. As most Node projects begin, open your terminal and start with initializing NPM to create a *package.json* file. Then install Express as a dependency.

```
npm init --yes
npm install --save express
```

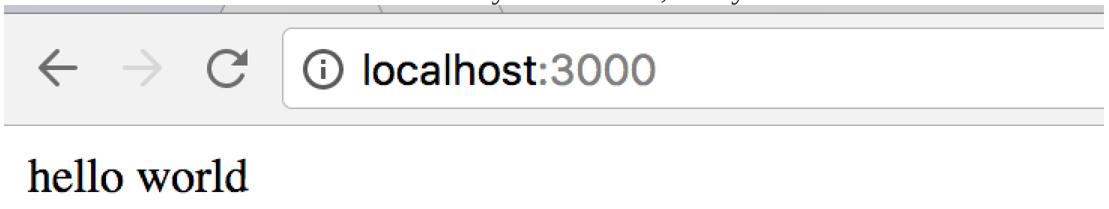
- We'll start with the classic "hello world" example, explain that, and then build from there. First, create a new file called *server.js* that will hold the code for, you guessed it, our Express server. In this file, write the following code:

```
const express = require('express');
const app = express();

app.get('/', (request, response) => {
  response.send('hello world');
});

app.listen(3000, () => {
  console.log('Express intro running on localhost:3000');
});
```

- In your terminal, start the server using the command:

```
node server.js
```

- Then head on over to *localhost:3000* in your browser, and you should see:



hello world

VII. Project structure

This project contains 7 folders/packages with the functionality of each folder is as follows:

1. Models: to store all model of the database

Book.js	Chapter.js
Comment.js	Genre.js
Library.js	Rating.js
Review.js	Transaction.js
User.js	Vote.js

2. Config: to store all configurations of the project such as database connenction, etc.
3. Middlewares:

- Express middleware are functions that run after a request is received but before the route handler function. Middleware functions have access to the request object, response object, and a function called *next* to pass control to the next middleware function or to the route handler.
- The "middlewares" folder contains 3 file: admin.js, auth.js and book.js

4. Images: contains all images of the system
5. Node modules: generated folder
6. Route: contains all API of the project. The API tree of the project can be illustrated as:

```
routes
|____   api
        |____   auth (GET)
        |____   users (POST)
        |____   books (GET, POST)
        |       |_____ :bookid (GET, PUT, DELETE)
        |               |____   chapters (GET, POST)
        |               |       |_____ :chapid (GET, PUT, DELETE)
        |               |               |____   comments
        |               |                       |____   :commentid
        |               |____   reviews (GET, POST)
        |                       |_____ :reviewid (GET, PUT, DELETE)
        |                               |____   upvote
        |                               |____   downvote
        |____   genres (GET)
        |____   library (GET, POST, PUT, DELETE)
        |____   ratings (GET, PUT)
```

```
|____   votes (GET, POST, DELETE)
|____   transactions (GET, POST)
```

7. Client

- Components: contains all components of the website according to React framework.
- Actions: contains all actions which are payloads of information that send data from your application to the store of React-redux.
- Reducers: This folder holds all reducer functions. After those actions in Actions folder get dispatched, they will be handled in reducers.

VIII. Individual contribution

1. Le Minh Hai Phong

a) Contribution

- In charge of 3 usecases: Manage user's library, Evaluate book, Read book.
- Create database in MongoDB of all system.

b) Database design:

- Manage user's library:

  + User: user_id

  + Book: book_id

  + Library: id, book_id, user_id, date, bookmark
- Evaluate book:

  + User: user_id

  + Book: book_id

  + Review: user_id, book_id, content, date, rating

  + Comment: user_id, chapter_id, content, upvote, downvote, date
- Read book:

  + Book: book_id

  + Chapter: chapter_id, book_id, name, content, create_date. is_publish, publish_date

  + Transaction: user_id, chapter_id, coin, date

c) Explanation of the code components

Take the usecase "Read free chapter" in "Read book" usecase for example. First, we need to get the list of chapters in a given book. This work has be done by using the API:

```
// @route   GET api/books/:bookid/chapters?published=true
// @desc    get chapters of a book
// @access  Semi
router.get('/', findBookById, async (req, res, next) => {
    try {
        if (req.query.published === "false") {
            next();
            }
            else {
                const chapters = await Chapter.find({ book: req.book._id, publish
ed: true }, '-content').sort('number');
                return res.status(200).json({ chapters, success: true });
            }
        }
        catch (err) {
            console.error(err);
            res.status(500).send('Server Error when get chapters');
        }
    }, auth, async (req, res) => {
        try {
            if (req.book.author.equals(req.user.id)) {
                const chapters = await Chapter.find({ book: req.book._id }, '-
content').sort('number');
                return res.status(200).json({ chapters, success: true });
            }
            return res.status(400).send('Get chapters unauthorized');
        }
        catch (err) {
            console.error(err);
            res.status(500).send('Server Error when get chapters');
        }
    })
```

And then, get a specific chapter:

```
    // @route   GET api/book/:bookid/chapters/:chapid
    // @desc    get a chapter of a book
    // @access  Public  if chapter.price = 0
    //          Private otherwise
    router.get('/:chapid', findBookById, findChapterById, async (req, res, next)
=> {
        try {
            const { price, published } = req.chapter;
            if (price === 0 && published === true) {
                return res.status(200).json({chapter: req.chapter, success: true}
);
            }
            next();
        }
        catch (err) {
            console.log(err);
            res.status(500).send('Server Error');
        }
    }, auth, async (req, res) => {
        try {
            const userid = req.user.id;
            const chapid = req.params.chapid;
            if (req.book.author.equals(req.user.id) || (await Transaction.findOne
({ 'user': userid, 'chapter': chapid }))) {
                return res.status(200).json({chapter: req.chapter, success: true}
);
            }
            else {
                return res.status(400).json({ errors: [{ msg: 'User need to pay f
or chapter' }], book: req.book, user: req.user });
            }
        }
        catch (err) {
            console.log(err);
            res.status(500).send('Server Error');
        }
    })
```

In components/book/Chapter.jsx, we will display the content of that chapter:

```
class Chapter extends Component {
    constructor(props) {
        super(props);
        this.state = {
            bookid: '',
            chapid: '',
            chapters: null,
            book: null,
            chapter: null,
            comments: null,
            needRedirect: false,
            loading: false
        }
        this.componentDidMount = this.componentDidMount.bind(this);
        this.onChange = this.onChange.bind(this);
        this.onClickPrevChap = this.onClickPrevChap.bind(this);
        this.onClickNextChap = this.onClickNextChap.bind(this);
    }

    loadChapter = async (bookid, chapid) => {
        if (!bookid || !chapid) return null;
```

```
            const res_chapter = await axios.get('api/books/' + bookid + '/chapter
s/' + chapid);
            const res_book = await axios.get('api/books/' + bookid);

            if (res_chapter.data.success && res_book.data.success) {
                return {
                    book: res_book.data.book,
                    chapter: res_chapter.data.chapter
                };
            }
            return null;
        }

        getChapters = async (bookid) => {
            try {
                if (!bookid) return null;
                const { data } = await axios.get('api/books/' + bookid + '/chapte
rs', {
                    params: {
                        published: true
                    }
                });
                if (!data.success) return null;
                return data.chapters;
            }
            catch (err) {
                console.error(err);
                throw err;
            }
        }

        getComments = async (bookid, chapid) => {
            try {
                const res = await axios.get('api/books/' + bookid + '/chapters/'
+ chapid + '/comments', {
                    params: {

                    }
                });

                console.log(res.data);
                if (res.data.success) return res.data.comments;
                return null;
            } catch (err) {
                console.log(err);
                return null;
            }
        }

        async componentDidMount() {
            await this.setState({
                bookid: this.props.match.params.bookid,
                chapid: this.props.match.params.chapid
            })
            await this.setState(await this.loadChapter(this.state.bookid, this.st
ate.chapid));
            await this.setState({ chapters: await this.getChapters(this.state.boo
kid) });
            await this.setState({ comments: await this.getComments(this.state.boo
kid, this.state.chapid) });
        }

        displayContent = (content) => {
            const paragraphs = content.split("\n");
            return paragraphs.map(p => (
                <p>{p}</p>
            ))
        }

        changeChapter = async (chapid) => {
            this.setState({ loading: true });
            await Promise.all(
                [await this.setState({ chapid }),
                await this.setState(await this.loadChapter(this.state.bookid, thi
s.state.chapid)),
                await this.setState({ chapters: await this.getChapters(this.state
.bookid) })],
                await this.setState({ comments: await this.getComments(this.state
.bookid, this.state.chapid) })
            ).then(() => {
                this.setState({ loading: false });
            })
        }

        onChange = async (e) => {
            e.preventDefault();
            const newchap = e.target.value;
            if (newchap !== this.state.chapid) {
                this.changeChapter(newchap);
            }
        }

        onClickPrevChap = async () => {
            let num = this.state.chapter.number;
            const newchap = this.state.chapters.find(chap => chap.number === num
- 1);
            this.changeChapter(newchap._id);
        }

        onClickNextChap = async () => {
            let num = this.state.chapter.number;
            const newchap = this.state.chapters.find(chap => chap.number === num
+ 1);
            await this.changeChapter(newchap._id);
        }

        chaptersNavBar = () => {
            return (
                <Navbar style={{ justifyContent: "center", alignItems: "stretch"
}} noValidate>
                    <Button
                        disabled={this.state.chapter._id === this.state.chapters[
0]._id}

                        onClick={this.onClickPrevChap}
                    >
                        Prev Chapter
                    </Button>

                    <select defaultValue={this.state.chapter._id} onChange={this.
onChange}>
                        {this.state.chapters.map(chapter => (
                            <option key={chapter._id} value={chapter._id}>
                                {'Chapter ' + chapter.number + ": " + chapter.nam
e}
                            </option>
                        ))}
                    </select>

                    <Button
                        disabled={this.state.chapter._id === this.state.chapters[
this.state.chapters.length - 1]._id}
                        onClick={this.onClickNextChap}
                    >
                        Next Chapter
                    </Button>
                </Navbar>
            )
        }

        render() {
            if (!this.props.isAuthenticated) {
                return <Redirect to={{ pathname: '/login', state: { from: this.pr
ops.location } }} />;
            }

            if (!this.state.book || !this.state.chapter || !this.state.chapters |
| this.state.loading) {
                return <Spinner />;
            }

            return (
                <>
                    <div>
                        {this.chaptersNavBar()}
                        <h2>{this.state.book.name}</h2>
                        <h2>Chapter {this.state.chapter.number}: {this.state.chap
ter.name}</h2>
                        <h5>Author: {this.state.book.author.username}</h5>
                        <div>
                            {this.displayContent(this.state.chapter.content)}
                        </div>
                        {this.chaptersNavBar()}
                    </div>

                    <Fragment>
                        <CommentForm bookid={this.state.bookid} chapid={this.stat
e.chapid} />
                        <hr />
                        <hr />
                        <hr />
                        <CommentList comments={this.state.comments} />
                    </Fragment>
                </>
            )
        }
    }

    Chapter.propTypes = {
        isAuthenticated: PropTypes.bool
    };
```

```
const mapStateToProps = state => ({
    isAuthenticated: state.auth.isAuthenticated,
});

export default withRouter(connect(
    mapStateToProps,
    // { loadChapter }
)(Chapter));
```

2. Nguyen Binh Long

a) Contribution

- In charge of 1 usecase: Browse book
- System and UI design
- Develop frontend for the web

b) Database design:

- Browse book:

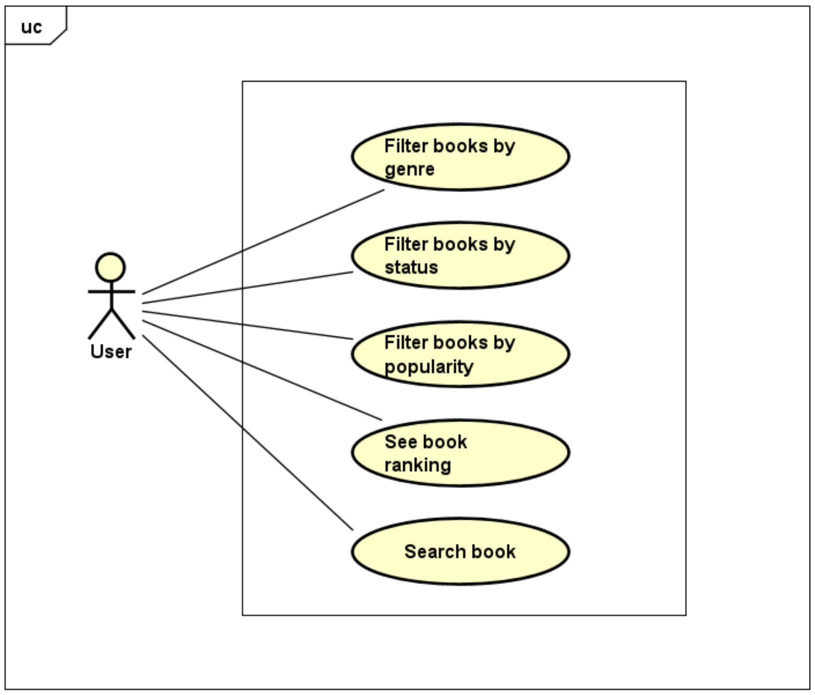
  + Book: book_id, author_id, name, cover, ratings, date

  + Genre: id, name

  + Book Genre: book_id, genre_id

c) Explanation of the code components

The "Browse book" usecase can be decomposed into smaller usecases which are:



Those decomposed usecases are implemented as follows:

- The Homepage of the website displays list of books sorted in created date (New ongoing releases) and popularity (Recommended). The source code of Homepage frontend is in components/home/Home.jsx
- Filter books by genre/status: This functionality is displayed after clicking on Browse in the Navbar of Header with the source code of frontend in components/browse folder.
- See book ranking: This functionality is displayed after clicking on Rankings in the Navbar of Header with the source code of frontend in components/rankings folder.
- User can search for books by using the search bar in the Navbar of Header.

Example of Filter books by genre usecase:

- In routes/api/books folder, you can see a file index.js which contains the API to get a list of books:

```
// @route   GET api/books
// @desc    get books user want to browse (depending on {genre, completed/ong
oing})
// @access  Public
/*
request: req.params: {
    author: author's id
    searchStr: ''
    genres: [genres]/'all'
    status: completed'/'ongoing'/'all'
    page: Number || 1
    perPage: Number || 10
    sortBy: 'alphabet'/'ratings'/'votes'/'popularity'/'onCreated' (default: '
alphabet')
    projection: 'name'
}
*/
router.get('/', async (req, res, next) => {
    try {
        let author = req.query.author;
        let genres = req.query.genres || 'all';
        let status = req.query.status || 'all';
        let page = parseInt(req.query.page || 1);
        let perPage = parseInt(req.query.perPage || -1);
        let sortBy = req.query.sortBy || 'alphabet';
        let projection = req.query.projection;

        if (author) next(); else {
            const books = await findBooksBy({ genres, status, page, perPage,
sortBy, projection: req.query.projection });
            res.status(200).json({ books, success: true });
        }
    } catch (err) {
        console.error(err.message);
        res.status(500).send('Server Error');
    }
}, auth, async (req, res) => {
    try {
        let author = req.query.author;
        let genres = req.query.genres || 'all';
        let status = req.query.status || 'all';
        let page = parseInt(req.query.page || 1);
        let perPage = parseInt(req.query.perPage || -1);
        let sortBy = req.query.sortBy || 'alphabet';
        let projection = req.query.projection;

        if (req.user.id != author) {
            return res.status(400).json({ success: false });
        }
        const books = await findBooksBy({
            author,
            genres,
            status,
            page,
            perPage,
            sortBy,
            projection
        });
        res.status(200).json({ books, success: true });
    }
    catch (err) {
        console.error(err);
        res.status(500).send('Server Error');
    }
})
```

- In components/browse/Browse.jsx, we will display the list of books in a specific order in the form of a table:

```
class Browse extends Component {
    constructor() {
        super();
        this.state = {
            books: null,
            genre: '',
            status: '',
            sortBy: ''
        }

        this.onChange = this.onChange.bind(this);
        this.componentDidMount = this.componentDidMount.bind(this);
    }

    loadBooksBy = async (genre, status, sortBy) => {
        try {
            const { data } = await axios.get('api/books', {
                params: {
                    genres: genre,
                    status: status,
                    sortBy: 'alphabet'
                }
            });
            if (data.success) return data.books;
            return null;
        }
        catch (err) {
            console.error(err);
            return null;
        }
    }

    async componentDidMount() {
        await this.setState({books: await this.loadBooksBy(this.state.genre,
this.state.status, this.state.sortBy)});
    }

    async onChange(e) {
        await this.setState({ [e.target.id]: e.target.value });
        await this.setState({books: await this.loadBooksBy(this.state.genre,
this.state.status, this.state.sortBy)});
    }

    render() {
        if (!this.state.books) {
            return <Spinner />
        }
        return (
            <>
                <h2 style={{ textAlign: 'center', marginTop: '20px' }}>Novels
 Genre</h2>
                <Container>
                    <Row>
                        <Col xs={12} md={4}>
                            <Form noValidate>
                                <Form.Group>
                                    <Form.Label>Genre</Form.Label>
                                    <Form.Control as="select" id="genre" onCh
ange={this.onChange} value={this.state.genre}>
                                        <option value='all'>All</option>
                                        {this.props.genres.map(genre => <opti
on key={genre.id} value={genre.name}>{genre.name}</option>)}
                                    </Form.Control>
                                </Form.Group>
                            </Form>

                        </Col>
                        <Col xs={12} md={4}>
                            <Form noValidate>
                                <Form.Group>
                                    <Form.Label>Status</Form.Label>
                                    <Form.Control as="select" id="status" onC
hange={this.onChange} value={this.state.status}>
                                        <option key='all' value='all'>All</op
tion>
```

```
                                        <option key='completed' value='comple
ted'>Completed</option>
                                        <option key='ongoing' value='ongoing'
>Ongoing</option>
                                    </Form.Control>
                                </Form.Group>
                            </Form>

                        </Col>
                    </Row>
                </Container>

                <Fragment>
                    <BookList books={this.state.books} />
                </Fragment>

            </>
        )
    }
}

const mapStateToProps = state => ({
    genres: state.genres
});

export default connect(
    mapStateToProps,
)(Browse);
```

3. Le Hong Minh

a) Contribution

- In charge of 2 usecases: Manage own account, Manage user's book

b) Database design:

- Manage own account:

  + User: user_id, username, password, avatar, email, coin

- Manage user's book

  + User; user_id

+ Book: book_id, author_id, name, cover, ratings, votes, is_complete, date

c) Explanation of the code components

For the usecase Manage user's book, let's consider the usecase "Create a book" for example. We can create a book by using the API:

```
// @route   POST api/books
// @desc    create new book
// @access  Private
/*
    const res = axios.post('api/books', {
        name: book name,
        genres: genres' names,
        cover: link img,
        sypnosis: String
    })
*/
router.post('/', auth, async (req, res) => {
    try {
        const author = req.user.id;
        const name = req.body.name;
        const genrenames = req.body.genres;
        const cover = req.body.cover;
        const sypnosis = req.body.sypnosis;
        const genres = await Genre.find().where('name').in(genrenames).select
('id').exec();
        if (genres.length === 0 || genres.length < genrenames.length) {
            return res.status(400).send('Invalid genres');
        }
        book = new Book({ author, name, genres, cover, sypnosis });
        await book.save();
        res.status(201).json(book);
    }
    catch (err) {
        console.error(err.message);
        res.status(500).send('Create book failed');
    }
})
```

The Component for create book is in component/create folder:

```
class Create extends Component {
    constructor() {
        super();
        this.state = {
            userid: null,
            books: null
        }

        this.componentDidMount = this.componentDidMount.bind(this);
    }

    getBooksCreated = async (userid) => {
        try {
            const { data } = await axios.get('api/books', {
                params: {
                    author: userid
                }
            });
            if (data.success) return data.books;
            return null;
        }
        catch (err) {
            console.error(err);
            this.props.history.push('/');
            return null;
        }
    }

    async componentDidMount() {
        await this.setState({
            userid: await this.props.user._id
        });

        await this.setState({
            books: await this.getBooksCreated(this.state.userid)
        })

    }

    render() {
        if (!this.props.isAuthenticated) {
            return <Redirect to='/login' />;
        }

        if (!this.state.books) {
            return <Spinner />
        }
        return (
            <>
                <Fragment>
                    <BookList books={this.state.books} />
                </Fragment>
                <Link to='/create/book'><Button>Create book</Button></Link>
            </>
        )
    }
}
const mapStateToProps = state => ({
    isAuthenticated: state.auth.isAuthenticated,
    user: state.auth.user
});

export default withRouter(connect(
    mapStateToProps,
)(Create));
```

References

1. https://reactjs.org/tutorial/tutorial.html
2. https://redux.js.org/basics/basic-tutorial
3. https://expressjs.com/
4. https://www.tecla.io/blog/2019-stats-on-top-js-frameworks-react-angular-and-vue/
5. https://www.webnovel.com/