KAUNAS UNIVERSITY of TECHNOLOGY

Faculty of Informatics

Philipp Schlaus

Study module:

T120B165 Web Application Design

Kaunas, 2024

CONTENT

1. DESCRIPTION OF THE PROBLEM TO BE SOLVED

1.1 PURPOSE OF THE SYSTEM

The reason to create this system is to make looking for books online more interesting. While people are searching for a book online, they often get distracted by popular titles or well-known authors, instead of reading the books descriptions. In some bookstores, readers can buy books hidden underneath wrapping paper. The only available information about the book is a description written by the employees of the store. This system is supposed to be a digital version of this. Users describe their impression of a book they like, list some similar books from the same genre and add a link for a website to be the book for interested people. Other users can read descriptions and decide whether to buy the book or look for another one. The admin is responsible for checking the book descriptions and deleting those which include information like title and author.

1.2. FUNCTIONAL REQUIREMENTS

1.2.1 UNREGISTERED SYSTEM USER CAN:

1. View the platform representative page.
2. Log in to the online application.

1.2.2 A REGISTERED SYSTEM USER CAN:

1. Disconnect from the online application.
2. Log in (register) to the platform.
3. Add a book description
4. Read about books from other users.
5. Search for books with certain genre/topic

1.2.3 THE ADMINISTRATOR CAN:

1. Remove users.
2. Remove descriptions of books that do not follow the rules.

2. SYSTEM ARCHITECTURE

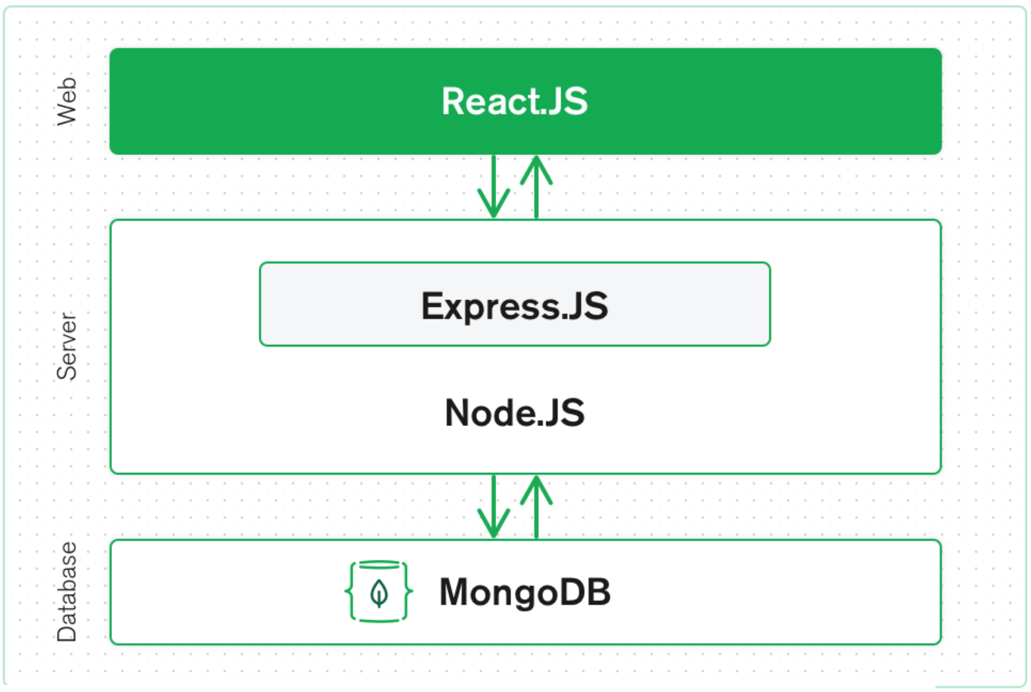
System components:

• Client side (Front-End) – will use React.js.

• Server side (Back-End) - will use Node.js and its web framework Express.js.

• Database - MongoDB.

In the following diagram, the deployment diagram of the system under development is shown.



Source: MERN Stack Explained | MongoDB

3. API SPECIFICATION

3.1 API METHODS FOR GENRES

The following tables describe the five API methods for genres:

API method	GET genre
Purpose	Receive information about one genre
Route	/api/genre/:genre_id
Request structure	-
Header	token: JWT-Token creatorID: ID of the current user
Response structure	{ "message": "genre found", "genre": { "_id": "…", "title": "…", "description": "…", "createdBy": "…" } }
Response code	200 - OK
Possible error codes	404 – if genre is not found 401 – if token and ID do not match or token is expired
Example request	localhost:5000/api/genre/674ef5e72af74416a3a50c81
Example response	{ "message": "genre found", "genre": { "_id": "674ef5e72af74416a3a50c81", "title": "comics", "description": "This is for comic books",

	"createdBy": "673dc96f0dccfbc19bf44d9b" } }

API method	GET genres
Purpose	Receive a list of all genres
Route	/api/genre/
Request structure	-
Header	token: JWT-Token creatorID: ID of the current user
Response structure	{ "genreList": [{ "_id": "…", "title": "…", "description": "…", "createdBy": "…" }, { "_id": "…", "title": "…", "description": "…", "createdBy": "…" }, { "_id": "…", "title": "…",

	"description": "…", "createdBy": "…" }] }
Response code	200 - OK
Possible error codes	404 – if no genre is found 401 – if token and ID do not match or token is expired
Example request	localhost:5000/api/genre/
Example response	{ "genreList": [{ "_id": "67373cce4181eec600b64e18", "title": "fantasy", "description": "books about fantasy", "createdBy": "672f52efea3cbc1b6fc94096" }, { "_id": "674ef5e72af74416a3a50c81", "title": "comics", "description": "This is for comic books", "createdBy": "673dc96f0dccfbc19bf44d9b" }, {

	"_id": "674efb5e2af74416a3a50c82", "title": "Crime", "description": "Books about (true) crime", "createdBy": "673dc96f0dccfbc19bf44d9b" }] }

API method	PUT genre
Purpose	Update an existing genre
Route	/api/genre/:genre_id
Request structure	{ "title": "…", "definition": "…", "creatorID": "…" }
Header	token: JWT-Token
Response structure	{ "message": "genre found", "genre": { "_id": "…", "title": "…", "description": "…",

	"createdBy": "…" } }
Response code	200 - OK
Possible error codes	404 – if genre is not found 403 – if user is neither admin or creator of the genre 401 – if token and ID do not match or token is expired 400 – if the request does not have all the parameters

Example request	localhost:5000/api/genre/674ef5e72af74416a3a50c81
	
	{
	"title": "history",
	"definition": "books about history",
	"creatorID": "67503720170caf685843d691"
	}

Example response	{
	"message": "Updating successful",
	"data": {
	"genre_id": "67503f0cbd86610033d8034e",
	"title": "history",
	"definition": "books about history",
	"creatorID": "67503720170caf685843d691"
	}
	}

API method	POST genre
Purpose	Create a new genre
Route	/api/genre/

Request structure	
	```
	{
	    "title": "…",
	    "definition": "…",
	    "creatorID": "…"
	}
	```

Header	token: JWT-Token

Response structure	
	```
	{
	    "message": "genre found",
	    "genre": {
	        "_id": "…",
	        "title": "…",
	        "description": "…",
	        "createdBy": "…"
	    }
	}
	```

Response code	201 - Created
Possible error codes	405 – if the genre already exists 401 – if token and ID do not match or token is expired 400 – if the request does not have all the parameters

Example request	localhost:5000/api/genre/
	```
	{
	    "title": "comics",
	```

	"definition": "comic books", "creatorID": "673b2d48d68ebf7f18322757" }

Example response	{
	"message": "Updating successful",
	"data": {
	"genre_id": "67503f0cbd86610033d8034e",
	"title": "history",
	"definition": "books about history",
	"creatorID": "67503720170caf685843d691"
	}
	}

API method	DELETE genre
Purpose	Delete a genre
Route	/api/genre/:genre_id

Request structure	{
	"creatorID": "…"
	}

Header	token: JWT-Token

Response structure	{
	 "message": "Deleting successful",
	 "data": {
	 "genre_id": "…",
	 "creatorID": "…"
	 }
	 }

Response code	200 - OK
Possible error codes	404 – if genre is not found 403 – if user is neither admin nor creator of the genre 401 – if token and ID do not match or token is expired 400 – if the request does not have all the parameters

Example request	localhost:5000/api/genre/674ef5e72af74416a3a50c81
	
	 {
	 "creatorID": "673dc96f0dccfbc19bf44d9b"
	 }

Example response	{
	 "message": "Deleting successful",
	 "data": {
	 "genre_id": "675c48a151e6168bac661654",
	 "creatorID": "67503720170caf685843d691"
	 }
	 }
	 }

3.2 API METHODS FOR BOOKS

The following tables describe the five API methods for books:

API method	GET book
Purpose	Receive information about one book
Route	/api/genre/:genre_id/books/:book_id
Request structure	-
Header	token: JWT-Token creatorID: ID of the current user
Response structure	{ "_id": "…", "title": "…", "description": "…", "link": "…", "createdBy": "…", "genre": "…" }
Response code	200 - OK
Possible error codes	404 – if genre or book is not found 401 – if token and ID do not match or token is expired
Example request	localhost:5000/api/genre/67373cce4181eec600b64e18/books/6739b863a8861763e697913f
Example response	{ "_id": "6739b863a8861763e697913f", "title": "newer title for my book", "description": "Text", "link": "shop.com", "createdBy": "672f52efea3cbc1b6fc94096",

	"genre": "67373cce4181eec600b64e18" }

API method	GET books
Purpose	Receive a list of all books in a genre
Route	/api/genre/:genre_id/books
Request structure	-
Header	token: JWT-Token creatorID: ID of the current user

Response structure	{
	  "bookList": [
	    {
	      "_id": "…",
	      "title": "…",
	      "description": "…",
	      "link": "…",
	      "createdBy": "…",
	      "genre": "…"
	    },
	    {
	      "_id": "…",
	      "title": "…",
	      "description": "…",
	      "link": "…",
	      "createdBy": "…",
	      "genre": "…"
	    }

	] }
Response code	200 - OK
Possible error codes	404 – if no genre is found or the genre has no books 401 – if token and ID do not match or token is expired
Example request	localhost:5000/api/genre/67373cce4181eec600b64e18/books/
Example response	{ "bookList": [{ "_id": "6739b863a8861763e697913f", "title": "newer title for my book", "description": "Text", "link": "shop.com", "createdBy": "672f52efea3cbc1b6fc94096", "genre": "67373cce4181eec600b64e18" }, { "_id": "674ec90190810b653296508c", "title": "Fantasy Book", "description": "I love the book", "link": "bookLink", "createdBy": "673dc96f0dccfbc19bf44d9b", "genre": "67373cce4181eec600b64e18" }] }

API method	PUT book
Purpose	Update an existingbkook
Route	/api/genre/:genre_id/books/:book_id

Request structure	{
	    "title": "…",
	    "description": "…",
	    "genre": "…",
	    "link": "…",
	    "creatorID": "…"
	}

Header	token: JWT-Token

Response structure	{
	    "message": "Updating successful",
	    "data": {
	        "genre_id": "…",
	        "newGenre": "example genre",
	        "title": "…",
	        "description": "…",
	        "link": "… ",
	        "creatorID": "…",
	        "token": "…"
	    }
	}

Response code	200 - OK

Possible error codes	404 – if genre or book is not found 403 – if user is neither admin nor creator of the genre 401 – if token and ID do not match or token is expired 400 – if the request does not have all the parameters

Example request	localhost:5000/api/genre/67373cce4181eec600b64e18/books/674ec90190810b653296508c
	{
	"title": "newer title for my book",
	"description": "Text",
	"genre": "example genre",
	"link": "shop.com",
	"creatorID": "67503720170caf685843d691"
	}

Example response	{
	"message": "Updating successful",
	"data": {
	"genre_id": "67503f0cbd86610033d8034e",
	"title": "history",
	"definition": "books about history",
	"creatorID": "67503720170caf685843d691"
	}
	}

API method	POST book
Purpose	Create a new book
Route	/api/genre/books

Request structure	{
	"title": "…",
	"definition": "…”
	"link": "…",
	"creatorID":"…"
	}

Header	token: JWT-Token

Response structure	{
	"message": "Adding successful",
	"data": {
	"title": "…",
	"definition": "…
	"creatorID": "…",
	"genre_id": "…",
	}
	}

Response code	201 - Created
Possible error codes	404 – if the genre does not exist 401 – if token and ID do not match or token is expired 400 – if the request does not have all the parameters

Example request	localhost:5000/api/genre/67373cce4181eec600b64e18/books
	{
	"title": "newer history book",
	"definition": "I like this other book about historical events",
	"link": "shop2.com",
	"creatorID":"67503720170caf685843d691"
	}

Example response	{ "message": "Deleting successful", "data": { "genre_id": "67373cce4181eec600b64e18", "book_id": "674ec90190810b653296508c", "creatorID": "67503720170caf685843d691" } }

API method	DELETE book
Purpose	Delete a book
Route	/api/genre/:genre_id/books/:book_id
Request structure	{ "creatorID": "…" }
Header	token: JWT-Token
Response structure	{ "message": "Deleting successful", "data": { "genre_id": "…", "book_id": "…", "creatorID": "…" } }

Response code	200 - OK
Possible error codes	404 – if genre is not found 403 – if user is neither admin nor creator of the genre 401 – if token and ID do not match or token is expired 400 – if the request does not have all the parameters

Example request	localhost:5000/api/genre/67373cce4181eec600b64e18/books/6739b91ea8861763e6979140
	{
	    "creatorID": "673dc96f0dccfbc19bf44d9b"
	}

Example response	{
	    "message": "Deleting successful",
	    "data": {
	        "genre_id": "675c48a151e6168bac661654",
	        "creatorID": "67503720170caf685843d691"
	    }
	}
	}

3.3 API METHODS FOR REVIEWS

The following tables describe the five API methods for reviews:

API method	GET review
Purpose	Receive information about one review
Route	/api/genre/:genre_id/books/:book_id/reviews
Request structure	-
Header	token: JWT-Token creatorID: ID of the current user

Response structure	{
	"_id": "…",
	"title": "…",
	"description": "…",
	"createdBy": "…",
	"genre": "…",
	"book": "…",
	"rating": …
	}

Response code	200 - OK
Possible error codes	404 – if genre, book or review is not found 401 – if token and ID do not match or token is expired
Example request	localhost:5000/api/genre/67503f0cbd86610033d8034e/books/67503f34bd86610033d8034f/reviews/67503f57bd86610033d80350
Example response	{ "_id": "67503f57bd86610033d80350", "title": "Review", "description": "I like this book",

	"createdBy": "67503720170caf685843d691", "genre": "67503f0cbd86610033d8034e", "book": "67503f34bd86610033d8034f", "rating": 0 }

API method	GET reviews
Purpose	Receive a list of all reviews for one book
Route	/api/genre/:genre_id/books/:book_id/reviews
Request structure	-
Header	token: JWT-Token creatorID: ID of the current user

Response structure	{
	"reviewList": [
	{
	"_id": "…",
	"title": "…",
	"description": "…",
	"createdBy": "…",
	"genre": "…",
	"book": "…",
	"rating": …
	}
	]
	}

Response code	200 - OK
Possible error codes	404 – if no genre or book is found or the book has no reviews 401 – if token and ID do not match or token is expired
Example request	localhost:5000/api/genre/67503f0cbd86610033d8034e/books/67503f34bd86610033d8034f/reviews/67503f57bd86610033d80350

Example response	{
	"reviewList": [
	{
	"_id": "67503f57bd86610033d80350",
	"title": "Review",
	"description": "I like this book",
	"createdBy": "67503720170caf685843d691",
	"genre": "67503f0cbd86610033d8034e",
	"book": "67503f34bd86610033d8034f",
	"rating": 0
	}
	]
	}

API method	PUT review
Purpose	Update an existing genre
Route	/api/genre/:genre_id/books/:book_id
Request structure	{ "title":"…",

	"text":"…", "creatorID":"…" }
Header	token: JWT-Token

Response structure	{
	"message": "Updating successful",
	"data": {
	"genre_id": …
	"book_id": …
	"title": …
	"text": …
	"creatorID": …
	}
	}

Response code	200 - OK
Possible error codes	404 – if genre or book or review is not found 403 – if user is neither admin nor creator of the review 401 – if token and ID do not match or token is expired 400 – if the request does not have all the parameters
Example request	localhost:5000/api/genre/67373cce4181eec600b64e18/books/674ec90190810b653296508c { "title": "newer title for my book", "description": "Text", "genre": "example genre", "link": "shop.com", "creatorID": "67503720170caf685843d691"

	}

Example response	{
	"message": "Updating successful",
	"data": {
	"genre_id": "67503f0cbd86610033d8034e",
	"book_id": "67503f34bd86610033d8034f",
	"title": "this review",
	"text": "new updated text",
	"creatorID": "67503720170caf685843d691"
	}
	}

API method	POST review
Purpose	Create a new review
Route	/api/genre/:genre_id/books/:book_id/reviews

Request structure	{
	"title":"…",
	"text":"…",
	"creatorID":"…",
	"rating":…
	}

Header	token: JWT-Token

Response structure	{
	"message": "Adding successful",
	"data": {
	"title": "…",

	"text": "…", "creatorID": "…", "genre_id": "…", "book_id": "…" } }
Response code	201 - Created
Possible error codes	404 – if the genre does not exist 401 – if token and ID do not match or token is expired 400 – if the request does not have all the parameters

Example request	localhost:5000/api/genre/67373cce4181eec600b64e18/books/674ec90190810b653296508c/reviews
	{
	"title":"this is another title",
	"text":"this is the text",
	"creatorID":"67503720170caf685843d691",
	"rating":1
	}

Example response	{
	"message": "Adding successful",
	"data": {
	"title": "this is another title",
	"text": "this is the text",
	"creatorID": "67503720170caf685843d691",
	"genre_id": "67373cce4181eec600b64e18",
	"book_id": "674ec90190810b653296508c"
	}
	}

API method	DELETE review
Purpose	Delete a review
Route	/api/genre/:genre_id/books/:book_id/reviews/:review_id
Request structure	{ "creatorID": "…" }
Header	token: JWT-Token
Response structure	{ "message": "Deleting successful", "data": { "genre_id": "…", "book_id": "…", "review_id": "…", "creatorID": "…" } }
Response code	200 - OK
Possible error codes	404 – if genre is not found 403 – if user is neither admin nor creator of the genre 401 – if token and ID do not match or token is expired 400 – if the request does not have all the parameters
Example request	localhost:5000/api/genre/67373cce4181eec600b64e18/books/674ec90190810b653296508c/reviews/676819a22cb3b54dbe450b8c

	{
	"creatorID": "673dc96f0dccfbc19bf44d9b"
	}

Example response	{
	"message": "Deleting successful",
	"data": {
	"genre_id": "67373cce4181eec600b64e18",
	"book_id": "674ec90190810b653296508c",
	"review_id": "676819a22cb3b54dbe450b8c",
	"creatorID": "67503720170caf685843d691" }
	}

3.4 API METHODS FOR USER MANAGEMENT

The following tables describe the five API methods for user management:

API method	GET users
Purpose	Receive information about the users
Route	/api/user/
Request structure	-
Header	token: JWT-Token creatorID: ID of the current user

Response structure	{
	"user": {
	"_id": "…",
	"email": "…",
	"password": "…",
	"admin": **…**,
	"token": "…",

	"timeout": … } }
Response code	200 - OK
Possible error codes	401 – if token and ID do not match or token is expired
Example request	localhost:5000/api/user/

Example response	{
	"user": {
	"_id": "67503720170caf685843d691",
	"email": "final@user.de",
	"password": "987",
	"admin": **false**,
	"token":
	"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJFbWFpb
	CI6ImZpbmFsQHVzZXIuZGUiLCJQYXNzd29yZCI6Ij
	k4NyIsImlhdCI6MTczNDg3NDc0MCwiZXhwIjoxNzM
	0ODc2NTQwfQ.9D5INASvbLXevOU5QpdkoyN3Qm2
	KEBX6-SVm1cB0E1A",
	"timeout": 1734879450000
	}
	}

API method	PUT user
Purpose	Login an existing user
Route	/api/user/

Request structure	{
	"email":"…",
	"password":"…"
	}

Header	-

Response structure	{
	"message": "Login successful",
	"data": {
	"Email": "…",
	"Password": "…",
	"token": "…",
	"id": "…"
	}
	}

Response code	200 - OK
Possible error codes	400 – if the request does not have all the parameters or login does not work

Example request	localhost:5000/api/user/
	{
	"email":"final@user.de",
	"password":"987"
	}

Example response	{
	"message": "Login successful",
	"data": {
	"Email": "final@user.de",
	"Password": "987",

	"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJFbWFpbCI6ImZpbmFsQHVzZXIuZGUiLCJQYXNzd29yZCI6Ijk4NyIsImlhdCI6MTczNDg4MzUyOSwiZXhwIjoxNzM0ODg1MzI5fQ.7v-feyPc3DBqY7QcmEJQSBF00htO7jdaiHiH5bLpcVs", "id": "67503720170caf685843d691" } }

API method	POST user
Purpose	Create a new user
Route	/api/user/

Request structure	{
	"email":"…",
	"password":"…",
	"confirm":"…",
	"admin":**…**
	}

Header	-

Response structure	{
	"message": "genre found",
	"genre": {
	"_id": "…",
	"title": "…",
	"description": "…",
	"createdBy": "…"

	} }
Response code	201 - Created
Possible error codes	403 – if the email is already used 401 – if passwords do not match 400 – if the request does not have all the parameters

Example request	localhost:5000/api/user/
	
	{
	"email":"user@mail.com",
	"password":"1234",
	"confirm":"1234",
	"admin":**false**
	}

Example response	{
	"message": "Creating account successful",
	"data": {
	"email": "user2@mail.com",
	"password": "1234",
	"confirm": "1234",
	"admin": **false**,
	"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6InVzZXIyQG1haWwuY29tIiwicGFzc3dvcmQiOiIxMjM0IiwiYWRtaW4iOmZhbHNlLCJpYXQiOjE3MzQ4ODM4MDMsImV4cCI6MTczNDg4NTYw

	M30.mSw6nVkelNtqJWXFyY508EFbN4O UqzmzshvZ0exj1zU", "creatorID": "676839db2cb3b54dbe450b8d" } }

API method	DELETE user to logout
Purpose	Logging out the user
Route	/api/user/
Request structure	-
Header	token: JWT-Token

Response structure	{
	"message": "Logged out",
	"data": {
	"email": "…",
	"password": "…",
	"_id": "…"
	}
	}

Response code	200 - OK
Possible error codes	400 – if the request does not have a token
Example request	localhost:5000/api/user/

Example response	{
	"message": "Logged out",
	"data": {

	"email": "new@user.net", "password": "abc", "_id": "674d8efa117eb24327e2ada0" } }

API method	DELETE user entirely
Purpose	Deleting the user
Route	/api/admin/user/

Request structure	{
	"email":"…",
	"adminID":"…"
	}

Header	token: JWT-Token

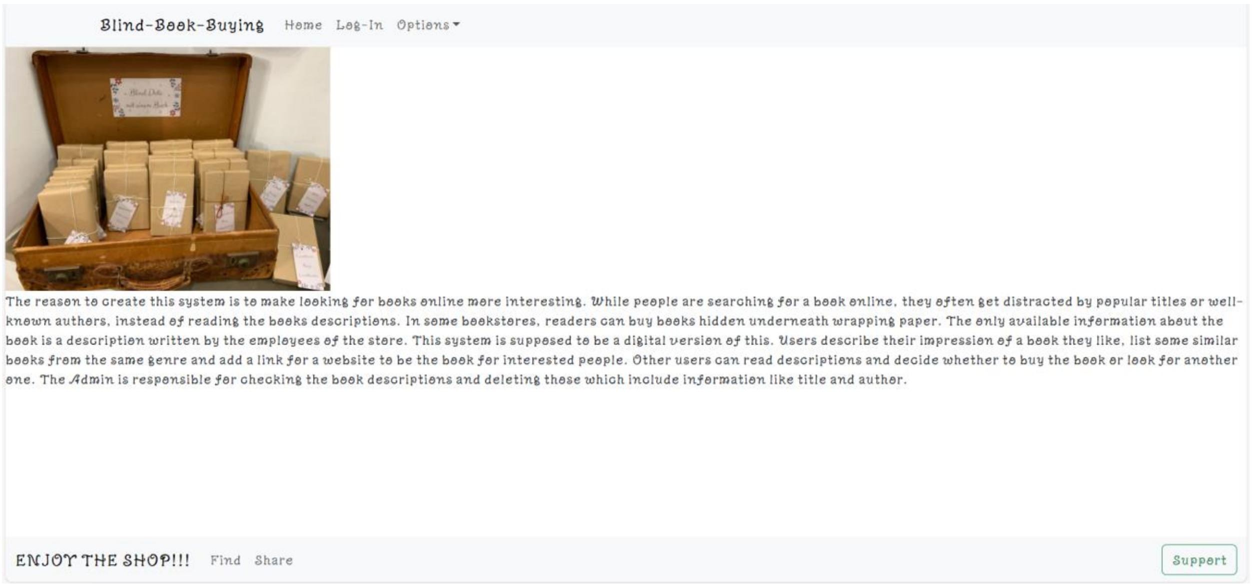
Response structure	{
	"message": "user deleted",
	"data": {
	"email": "user@mail.com",
	"adminID": "673db96423f2890ae4537069"
	}
	}

Response code	200 - OK
Possible error codes	404 – if the request has neither userID nor email 400 – if the request does not have a token
Example request	localhost:5000/api/admin/user/ { "email":"user@mail.com",

	"adminID":"673db96423f2890ae4537069" }
Example response	{ "message": "user deleted", "data": { "email": "user@mail.com", "adminID": "673db96423f2890ae4537069” } }

4. SYSTEM USER INTERFACE

This is the home page appearing after opening the application:

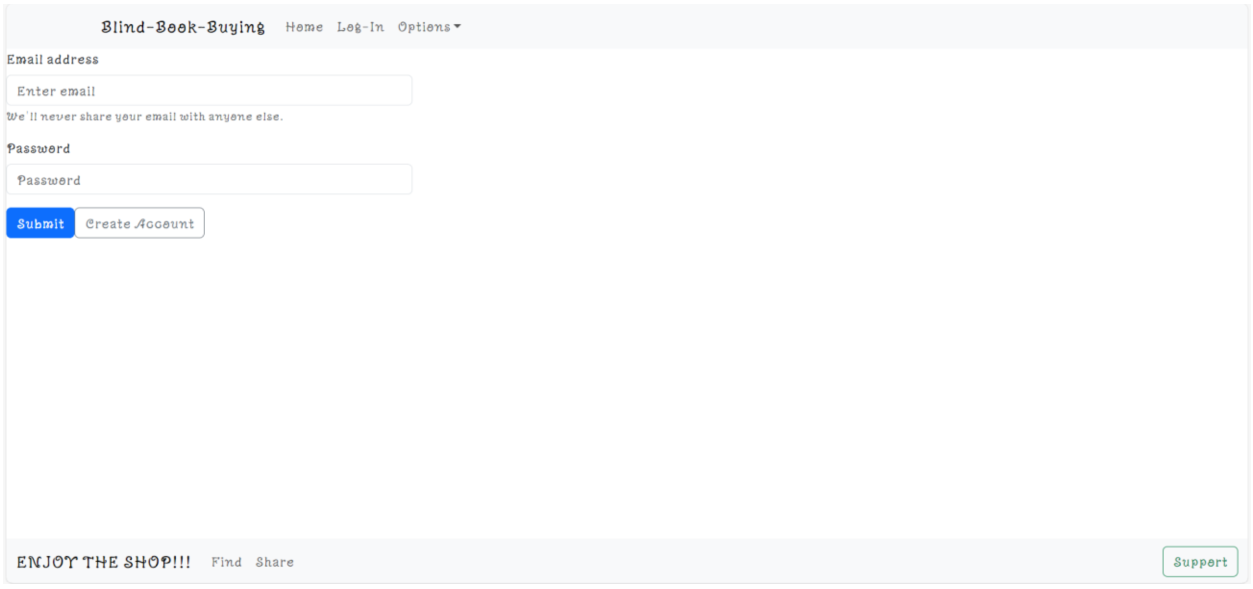


The reason to create this system is to make looking for books online more interesting. While people are searching for a book online, they often get distracted by popular titles or well-known authors, instead of reading the books descriptions. In some bookstores, readers can buy books hidden underneath wrapping paper. The only available information about the book is a description written by the employees of the store. This system is supposed to be a digital version of this. Users describe their impression of a book they like, list some similar books from the same genre and add a link for a website to be the book for interested people. Other users can read descriptions and decide whether to buy the book or look for another one. The Admin is responsible for checking the book descriptions and deleting those which include information like title and author.

This representative page is accessible for all three roles unregistered user, registered user and admin. Clicking “Home” in the header leads back to this page. The button “Log-in” redirects to the log-in page. The dropdown “Options” includes “Search Book”, “Recommend Book” and “Account”. Which lead to the pages for searching a new book, recommending a book and the account page. In the footer, “Find” redirects to the page for searching books and

“Share” redirects to the page for recommending a book. The “Support” button is a shortcut for writing an email to the support (my KTU email). Unregistered users will be redirected to the log-in page if they try to access the pages for searching or recommending books.

The next screenshot shows the log-in page:



Blind-Book-Buying Home Log-In Options

Email address

Enter email

We'll never share your email with anyone else.

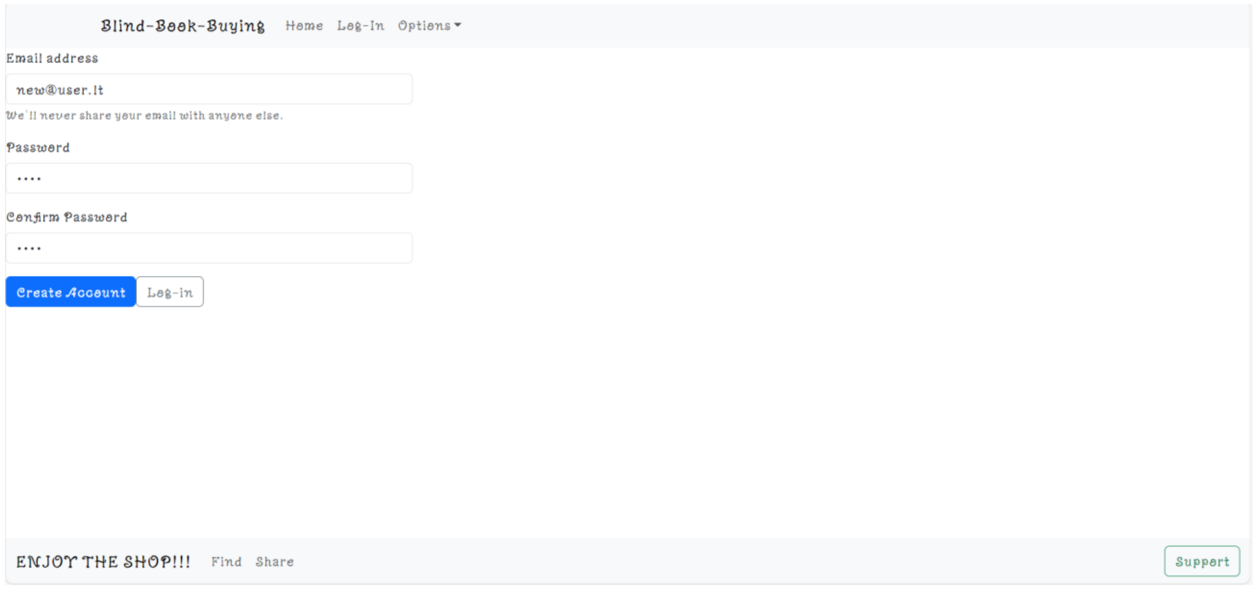
Password

Password

Submit Create Account

ENJOY THE SHOP!!! Find Share Support

This site offers a log-in with password and email. If both are correct, the user will be redirected to the homepage again after pressing “Submit”. If not, there will be a message to tell the log-in failed. Clicking “Create Account” switches to the page for registering a new user. It can be seen here:



Blind-Book-Buying Home Log-In Options

Email address

new@user.lt

We'll never share your email with anyone else.

Password

····

Confirm Password

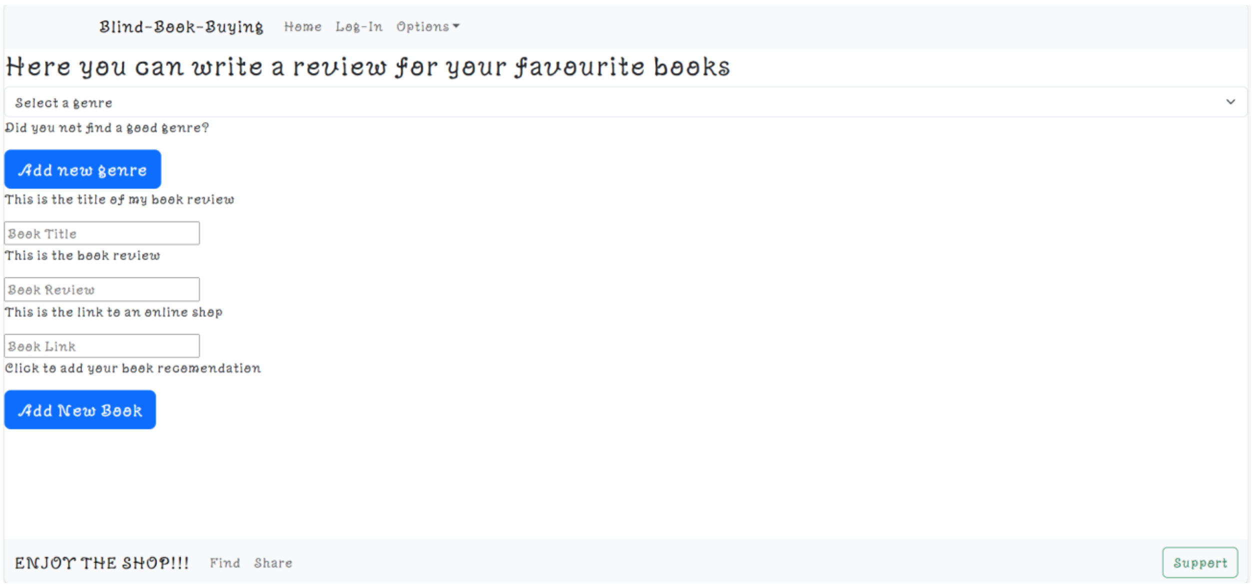
····

Create Account Log-in

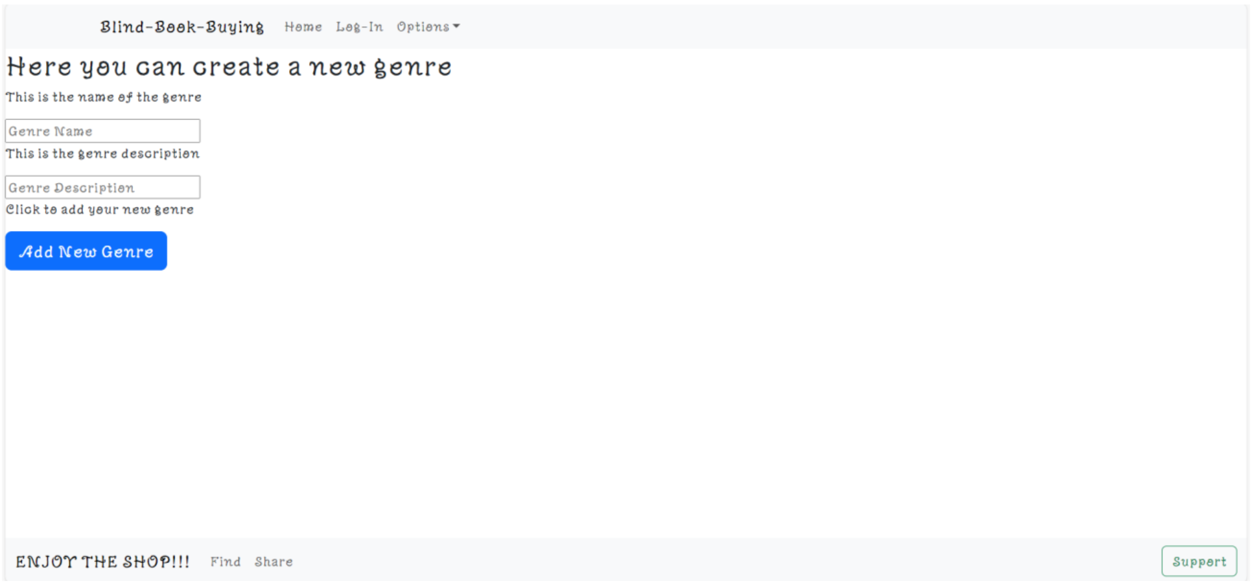
ENJOY THE SHOP!!! Find Share Support

After entering an email which is not used by another user, typing the same password twice and clicking “Create Account”, a new account is created and the user is redirected to the log-in. The button “Log-in” switches back to the page for logging in without creating an account. Any success or mistake will be shown by a message.

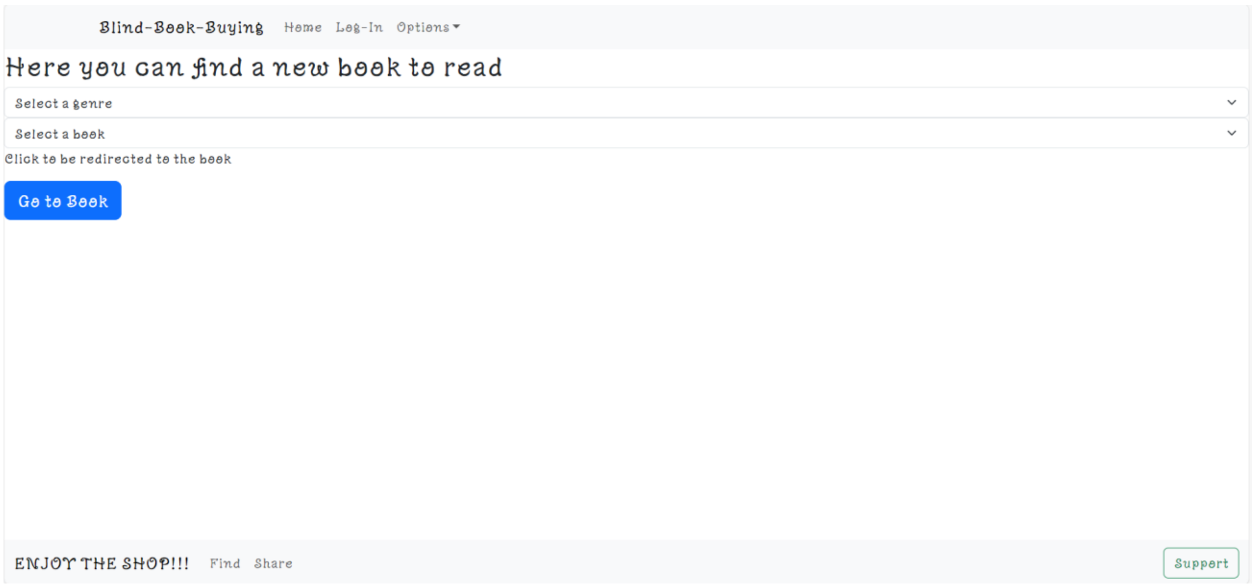
The following image shows the page for recommending a new book:


Blind-Book-Buying
Home
Log-In
Options
Here you can write a review for your favourite books
Select a genre
Did you not find a good genre?
Add new genre
This is the title of my book review
Book Title
This is the book review
Book Review
This is the link to an online shop
Book Link
Click to add your book recomendation
Add New Book
ENJOY THE SHOP!!!
Find
Share
Support

A genre can be chosen from the dropdown, or a new one can be created after clicking on "add a new genre". A descriptive title, the creators review and a link to an online shop should be added in the inputs. The button "Add New Book" adds a new book with the given features and redirects to the page for searching books. The next screenshot shows the page for a adding new genres which can be reached using the "Add new genre" button.


Blind-Book-Buying
Home
Log-In
Options
Here you can create a new genre
This is the name of the genre
Genre Name
This is the genre description
Genre Description
Click to add your new genre
Add New Genre
ENJOY THE SHOP!!!
Find
Share
Support

The name must not be used for another genre. Add new genre adds the genre to the available genres for new books. After adding the user is redirected back to the page for adding a book. The next screenshot shows the page for searching for an existing book:


Blind-Book-Buying Home Log-In Options

Here you can find a new book to read

Select a genre

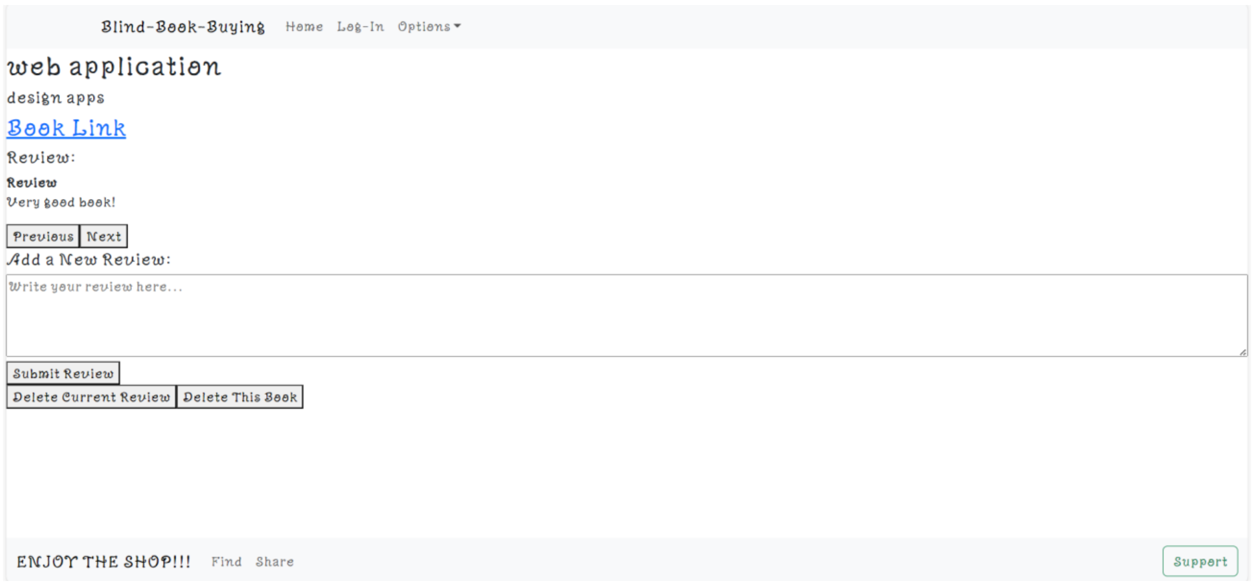
Select a book

Click to be redirected to the book

Go to Book

ENJOY THE SHOP!!! Find Share Support

The first dropdown is used to pick a genre. Afterwards, the second dropdown can be used to pick a book from this genre. The button “Go to Book” leads to the page about the book. The page about the book can be seen in the next screenshot.


Blind-Book-Buying Home Log-In Options

web application

design apps

Book Link

Review:

Review

Very good book!

Previous Next

Add a New Review:

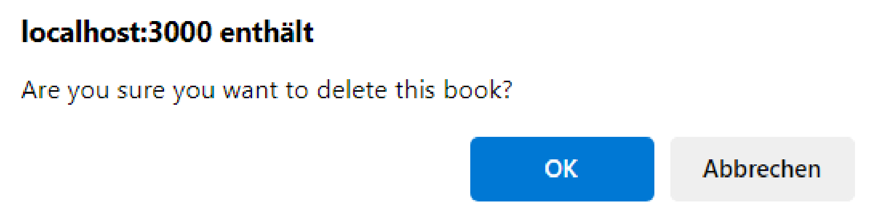
Write your review here...

Submit Review

Delete Current Review Delete This Book

ENJOY THE SHOP!!! Find Share Support

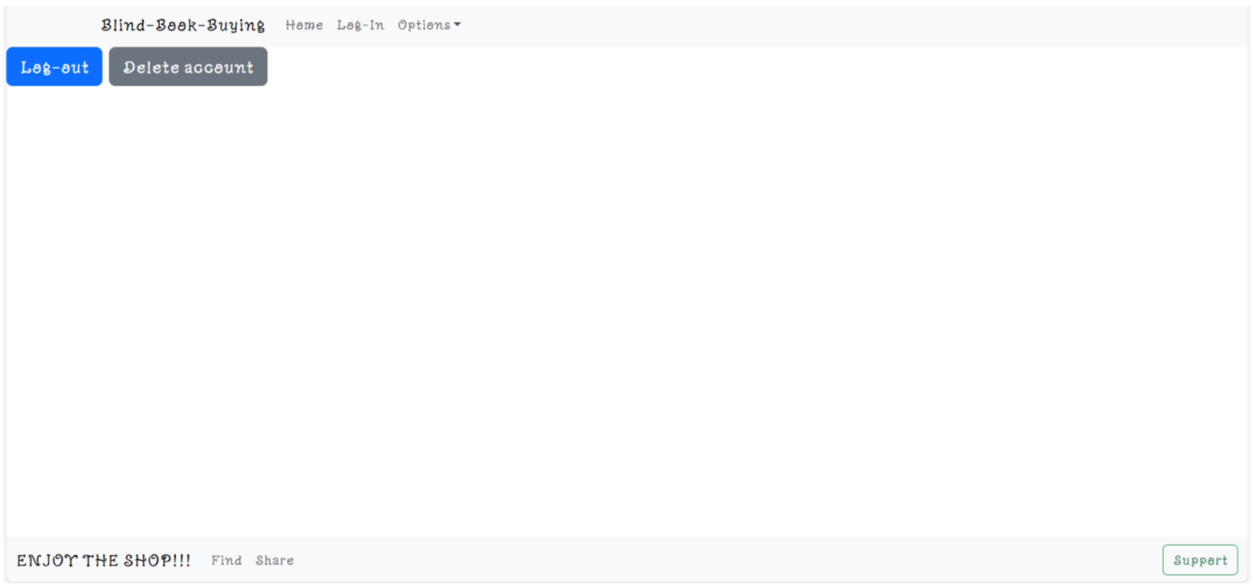
Here the information about a book is displayed including the link leading to an online shop. With the buttons “Previous” and “Next”, the user can switch the displayed review by other users. Clicking “Submit Review” adds the text in the input as a new review. The deleting buttons can be used by an admin or the creator to delete the currently displayed review or the whole book. Before deleting a review, this modal window pops up:


localhost:3000 enthält

Are you sure you want to delete this book?

OK Abbrechen

The final screenshots show the account options:



The buttons are used to Log-out or delete the account. Afterwards they redirect to the Log-in.

5. SYSTEM USER INTERFACE

1. Implemented system to enhance the online book shopping experience
2. The two parts of the system are a client and a server side
3. React.js is used on the client side and Node.js and Express.js on the server side.
4. The interface was implemented regarding the REST principles
5. 20 interface methods were implemented
6. Implemented authentication and authorization using JWT
7. A user interface was created, which accessses API methods
8. MongoDB is the software for the storage of data on the server side
9. The Code can be found in this repository phisch1211/Web-Application-Design