Final Project - LittleLend

Email Address	Identify Number	Full Name
noahaliva123@gmail.com	206966707	Noa Halili
shirelll134500@gmail.com	212222566	Shirel Boaron
avital27300@gmail.com	212423602	Avital Rahmanian
saraosditcher@gmail.com	209429703	Sari Osditcher

Host : D.r Galit Haim

Client Side Git : https://github.com/Shirel1345/LittleLend

Server Side Git : https://github.com/SaraOsdi/LittleLend

Table of Contents

Table of Figures

Background

This app is for social services, especially to help lend out baby items easily and efficiently. It was created because a person who runs a lending service found it too hard to use existing apps for this job. The main problems were:

- Hard to use on phones because there’s no mobile support.
- The app's design wasn't user-friendly.
- Users couldn't access it to see loan details or sensitive information like guarantees and when things need to be returned.
- It didn't connect items to their loans properly.
- Items could be accidentally deleted too easily.

We looked for a good app to solve these problems and found "Lending app - by Gosip Busak." But it had its own issues:

- You couldn’t manage loans or see a list of items and loans.
- You couldn't tell what was available or manage information about the items.
- It only showed a list of customers. For each person who ever borrowed something, it listed if they have something borrowed now and what it is.
- All details were just in a general description without specific places for each piece of information.
- It was hard and annoying to scroll through people to check the status of an item, especially if they didn’t currently have a loan, making them irrelevant.
- It seemed like the app was originally made for lending money because it wasn’t easy to check if an item was available without looking through each one by hand.

Professional literature review

- Stack Overflow
- Youtube
- Github
- Node.js.org
- docs.microsoft.com
- Angular Material
- primeng.org

Project's Target

This app makes managing a lending service easier with a functional and informative management interface. It has key features and focuses on high-quality management in the least amount of time.

Management Interface

- Create, update, and delete items.
- Create, update, and delete categories.
- Track loans and returns.
- Approve or reject loan requests.
- Rating product after loan

These features are only accessible to the manager because they contain sensitive details for the lending service, like deposit amounts and loan statuses.

User Interface

- Easily see available items only.
- Search and filter options based on customer needs.
- Easily request to borrow a product.
- Track the status of loan requests made by the user.
- Easy login/sign-up interface.

The screen will be user-friendly, showing only relevant details. The app is designed with baby products in mind as a conceptual extension of a successful lending service for carriers, but it can be adapted for general lending in any area.

What We Solve

We aim to fix problems with the previously mentioned app and create an app suited for efficiently managing product lending services with minimal hassle. The app will have:

A list of loans where you can see all active loans and their details.

A list of items where you can see all items, their loan status, and a user interface that reduces the manager's burden and eliminates the need for questions about item availability and details.

Requirements

- Connecting the client side to the server side: This allows users' actions on the app (like requesting loans) to be processed and responded to by the server.
- Connecting the server side to a MongoDB database: This stores all the data the app uses, such as details about items, loans, and users.
- Connecting the server side to AWS S3 for image uploads: This feature lets users upload pictures of items to the cloud, making it easier to see what they're borrowing.
- On the "My Loans" page, allowing customers to give feedback on products: After using an item, customers can leave feedback,

which will be displayed on the product pages for future users to see.
- Updating inventory after a loan is approved by the manager: This ensures the list of available items is always accurate.
- Interface for the manager to approve or reject loan requests: This gives control over which loans go through, based on item availability and user reliability.
- Option to create a new loan with dates and a choice of payment method: This makes the loan process flexible and convenient for users.
- Dedicated manager interface for creating, updating, and deleting products: This helps in keeping the product listings up to date with the current inventory.
- Dedicated manager interface for creating, updating, and deleting categories: This assists in organizing products into categories, making them easier to find.
- Sorting and filtering on the list of products: Users can easily find what they need by sorting items by various criteria or filtering them based on specific features.
- Sorting and filtering on the list of loans: This helps both users and managers track loans by status, date, or other relevant criteria.
- Together, these features aim to streamline the lending process, making it more manageable for the service providers and more engaging for the users.

Technologies we used - functional description

General

Angular

Angular is an open-source software framework for web applications, maintained by Google and a broad community of developers. It's designed to address various challenges in developing Single Page Applications (SPAs) and simplifies the development and testing of these applications by using the Angular design pattern, MVC (Model-View-Controller). Angular enhances current HTML tags by adding new attributes to the tag, which provides a more flexible and

diverse experience for the user than before. Angular first reads the HTML page to which these tags with different attributes are attached. It knows how to interpret these attributes and accordingly displays different information, which comes from the model. This information retrieval can be done either by accessing the server to fetch JSON files or by manipulating the information according to the user's request.

NodeJs

Node.js is a server-side, event-driven development environment written in JavaScript and built on Google Chrome's V8 engine. Its main use is to develop single-page network applications, video streaming sites, and generally any input-output broadcasting area.

Node.js works asynchronously, meaning that instead of using multiple threads for different operations, it operates with a single thread where all actions happen simultaneously. Whenever there's a specific request, the main thread allows the operation to occur elsewhere, not related to the main process. Therefore, it continues to operate until an interrupt signals that the operation has finished and can be dealt with. This approach saves resources and increases the throughput and flexibility of applications that handle many inputs and outputs, including real-time applications.

Among its primary purposes is to set up HTTP and WebSocket servers that are based on the HTTP protocol.

MongoDB

MongoDB is a NoSQL database based on documents, known for its flexibility, scalability, and convenience in storing and processing large volumes of data in various structures. The database stores data in documents composed of field and value pairs similar to JSON, making it highly compatible with complex data structures. MongoDB's schema-less nature allows developers to quickly build applications by adapting the data model on the fly without the need for pre-definition. Designed to meet high availability and scalability requirements, features

like data replication and data distribution methods make MongoDB an .ideal solution for applications of any size and complexity

AWS S3

Amazon S3 (Simple Storage Service) is an advanced object storage service from Amazon Web Services (AWS), allowing you to store and retrieve any amount of data from anywhere on the web. It is characterized by high durability, availability, and scalability, making it an ideal solution for backup and restore, data archiving, and large-scale applications. S3 provides a simple web service interface that enables you to upload, store, and retrieve any amount of data at any time, with advanced security and monitoring options to ensure data protection and availability. S3 is commonly used for content delivery for websites, .storing application data, and more

RESTful Software Architectural Style

The main idea behind this architecture is the definition of a resource whose state changes as a result of the interaction between the service provider and the consumer. In the REST architectural concept, which is a client-server model, there must be a client and a server. Clients initiate requests to servers, which process these requests and return .responses accordingly

The uniqueness of REST lies in the fact that the content transferred from the client to the server is a representation of a resource, which is the main concept in RESTful API. A resource is an object with these attributes: a type, information related to it, a relationship with other resources, and a set of operations that can be performed on it. It resembles an instance of an object in an object-oriented programming language, with the important difference that each resource has only a few predefined operations defined for it, which are the standard operations of HTTP (PUT, DELETE, POST, GET), while a regular object .has many more operations

Resources can be divided into collections. Each collection is homogeneous – containing only one type of resource. There's also a case where a resource might not be part of a collection, in which case .it's called a Singleton Resource

Mostly, each resource is represented by a JSON document, which .describes the various attributes of the object

JWT

JWT (JSON Web Token) is an open standard (RFC 7519) for securely transmitting information between two parties, typically between a server and a client. JWT allows you to transfer JSON objects in a compact and secure manner, where the information can be verified by checking a digital signature. These tokens are primarily used for user authentication and maintaining their session state in the system, enabling efficient management of login states without relying on concepts like sessions. Each token consists of three parts: Header, Payload, and Signature. The Header includes information about the type of token and the algorithm used, the Payload contains the actual information, and the Signature authenticates the token. Using JWT ensures that the information has not been tampered with or altered after being issued, providing an easy and efficient way to identify and authenticate users.

MVC (Model-View-Controller)

In the MVC (Model-View-Controller) architecture, splitting the application into three main components facilitates easier code management and maintenance. In backend development, this structure is translated into three key components: Model, Service, and Routes.

- Model: Defines the data structure and handles data access and operations in the database. This is the part where the business logic is described, responsible for representing the data that the application manages.
- Service: This component serves as the business logic layer. It processes data, both from the models and from external sources, and provides results to other components. The services are responsible for executing logic that is not directly dependent on the user interface or data storage.
- Routes: Defines the paths in the application, i.e., the addresses where users can request information or perform actions. Each

route is linked to a controller or a function on the server that handles the request and returns a response. This is the interface where communication between the client (e.g., a web browser) and the server occurs.

By dividing the application into models, services, and routes, we have managed to create much more organized and structured code, while maintaining consistency and ease of maintenance. Each component focuses on a specific function, allowing a modular approach to the development and management of complex applications.

MongoDB Atlas

MongoDB Atlas is a Database as a Service (DBaaS) that provides MongoDB, a popular NoSQL database. Atlas allows developers to store data flexibly and efficiently while offering a fully managed solution. This means operations like backups, maintenance, and network configuration are built-in and managed by MongoDB. Thus, developers can focus on developing their applications instead of managing infrastructure. We moved our database to Atlas to create a unified data model and give our system the flexibility to operate independently of its runtime environment.

Development Environment

Visual Studio Code

Visual Studio Code (VS Code) is a modern, open-source development environment that caters to the needs of web application development. It offers a wide range of features that enhance the development

experience, mainly through the use of various extensions that can improve the quality of development.

Git

Git is an open-source version control system designed to help developers manage code, coordinate team efforts, and track changes in software files. Its main goals are to provide speed, data integrity, and support for distributed, non-linear workflows. As a distributed version control system, every Git directory on every computer is considered a full-fledged repository with complete history and full version-tracking capabilities, independent of network access or a central server.

MongoDB Compass

MongoDB Compass is the official graphical user interface (GUI) tool provided by MongoDB. It offers an intuitive and visual way to manage and analyze MongoDB databases. This tool allows users to view data structures, execute queries, edit data, and analyze database performance graphically without the need to type complex queries. We used this tool to ensure that the data we entered was indeed saved in the database, making it easier to manage and interact with our MongoDB databases effectively.

Installation

Backend

Env File

The .env file is used to store environment-specific variables for an application in a Node.js environment. These environment variables are stored in the file in a Key=Value format and are hidden from the public code to protect sensitive information such as API keys, passwords, etc. Here's a breakdown of our code variables:

PORT specifies the port on which the application will run.

JWT_SECRET serves as a secret key for signing and authenticating the tokens. This ensures that the tokens sent and received from the server are authentic and have not been tampered with.

JWT_TIME indicates the duration after which the token expires. It can be set in units of time like minutes, hours, or days.

DB contains the URL for connecting to our database, in our case, it will be MongoDB Atlas.

AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY are the access identity and secret key designed to authenticate access to S3.

AWS_REGION specifies the region where our S3 is configured.

BUCKET_NAME is the name of the AWS S3 bucket to which we upload the images.

Running the API

1. git clone https://github.com/SaraOsdi/LittleLend
2. cd backend
3. npm i
4. npm start

Client Side

1. git clone https://github.com/Shirel1345/LittleLend.git
2. cd client
3. npm i
4. ng serve

System Architecture

General Schema

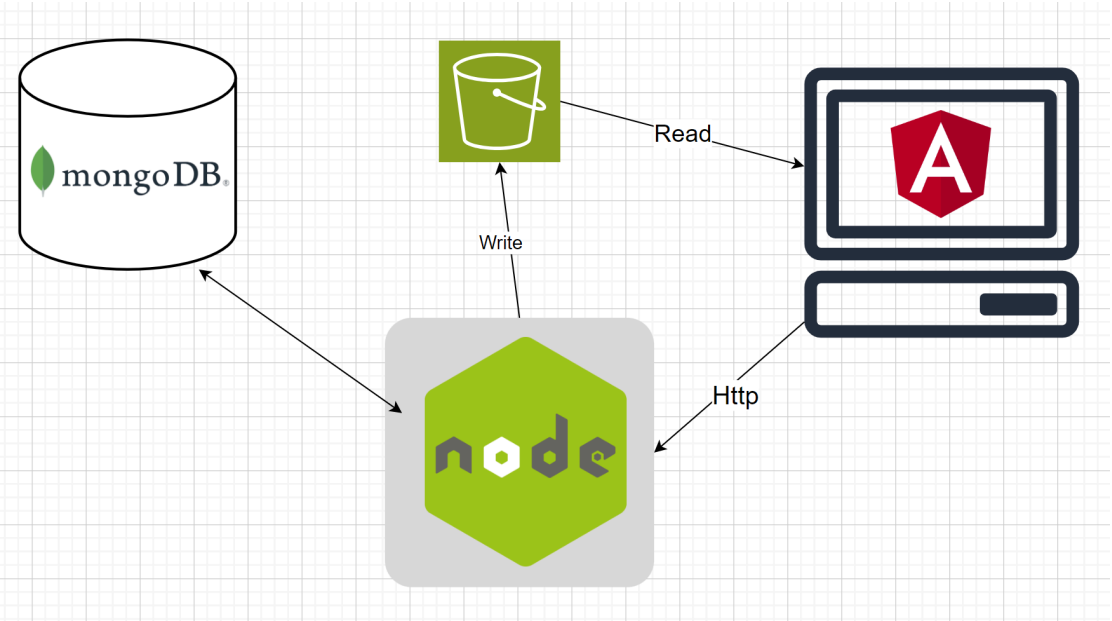


Figure 1 - general schema of system

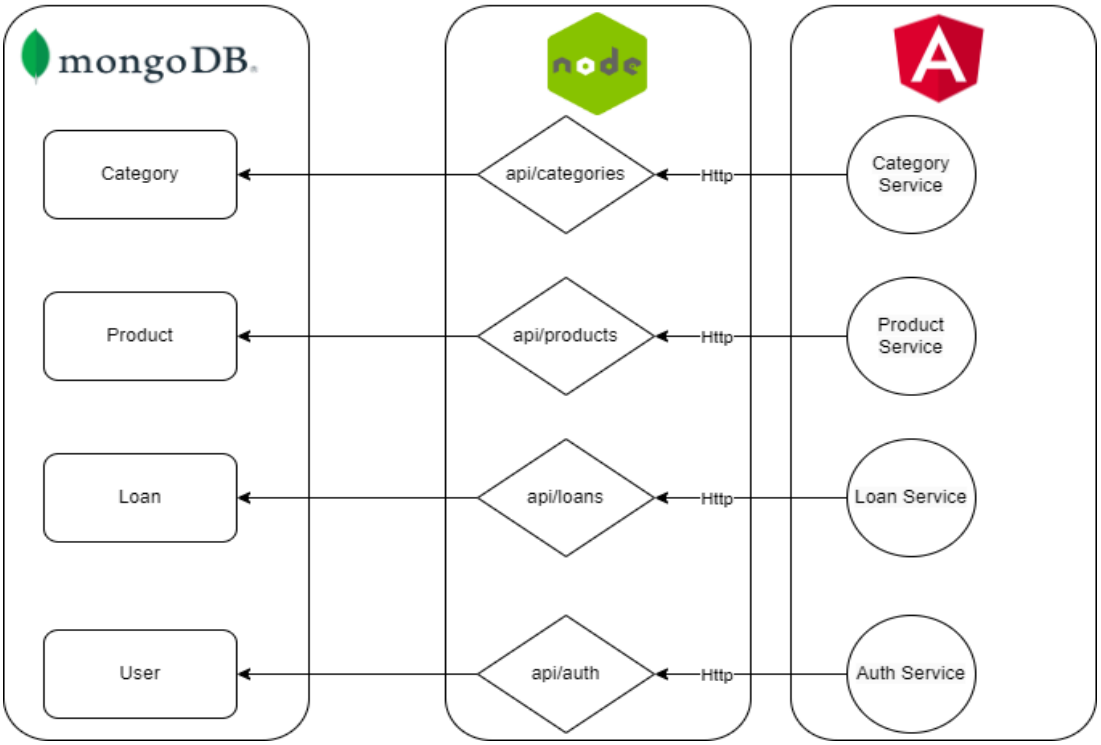


Figure 2 - schema of system interface

System Flow

The system, based on Angular for the client side, Node.js for the server side, and MongoDB for data management, utilizes CRUD services (Create, Read, Update, Delete) as the foundation for managing data of products, categories, loans, and user authentication. Each of these functionalities is represented by a service in Angular, designed to communicate with an API created in Node.js. Here's the general flow of the system:

Angular Services: On the client side, there's a separate Angular service for each functionality (product, category, loan, authentication). These services are responsible for sending organized HTTP requests to the server to create, read, update, and delete data.

Node.js API: The Node.js server handles the HTTP requests sent from the Angular services. For each category of actions (products, categories, loans, authentication), there are defined routes and controllers that execute business logic and perform operations on the database. If the request includes an image, the server will upload it to S3, receive its public URL, and place it in an object that will be saved in the database.

Authentication (Auth): Authentication is carried out using JWT. When a user logs in, the server generates a JWT token containing the user's identity and sends it back to the client. The client stores this token and uses it in future API requests to authenticate the user.

MongoDB: The MongoDB database stores all data related to products, categories, loans, and users. The API in Node.js performs operations on the database based on the requests received from the client and returns appropriate responses.

The core flow involves Angular services sending requests to the Node.js API, which then handles the logic, performs operations on MongoDB, and returns data to the client. The authentication process secures the communication and ensures that only authorized users can perform certain actions.

Libraries

PrimeNG

PrimeNG is a rich and comprehensive component library for Angular, offering a wide range of design and responsive UI (User Interface) components. Developed by PrimeTek Informatics, it is suitable for developers looking to create feature-rich applications quickly and efficiently while maintaining a professional and modern look. PrimeNG includes components such as dynamic tables, dropdowns, buttons, and many others, including support for built-in design themes and extensive customization.

Angular Material

Angular Material is a user interface component library based on Google's Material Design guidelines. The library provides a rich and sophisticated set of ready-to-use components, allowing developers to easily build beautiful, consistent, and responsive interfaces. Angular Material includes components like buttons, menus, dialog boxes, and form controls, all in accordance with the Material Design guide, ensuring a modern and uniform user experience.

Express

Express is a popular Node.js library that provides a robust, elegant, and flexible server-side framework. It enables developers to efficiently and simply build applications and web services, using a minimalist API. Express provides basic functionality for managing HTTP requests, middleware, routing, and more, and can be easily extended with plugins and external libraries.

Mongoose

Mongoose is an ODM (Object Data Modeling) library for MongoDB and Node.js.

Main Processes In The Application

Upload Image Process

The process of image upload involves capturing an image, uploading it to Amazon S3, and saving its URL in a MongoDB database using Node.js and Multer. This process can be broken down into several main steps:

Step 1 - Receiving the Image

The process begins when a user uploads an image of a product or category through the application's user interface. The image upload request is made through a form where the user selects the image from their device and then sends the image to the server along with any additional parameters of the object.

Step 2 - Capturing the Image with Multer

On the server, the image is captured using Multer, a Node.js library that handles files sent through multipart/form-data forms. Multer defines where and how files are stored during processing—in our case, it prepares the file for upload to S3, but it can also temporarily store the file in the server's file system.

Step 3 - Uploading the Image to Amazon S3

Once captured, the image is uploaded to Amazon S3, a flexible and secure file storage service from Amazon Web Services (AWS) that allows for cloud storage. The server uses the AWS S3 API to perform the upload, setting permissions and specifying the bucket where the file will be stored in S3 as part of the process.

Step 4 - Saving the URL in MongoDB

After uploading to S3, the server receives a URL for the stored image. This URL represents the location where the image can be accessed on the internet. The next step is to save this URL in the database.

Login Process

The process of logging into an application with username and password, securing the session with a JWT (JSON Web Token), and handling token validation can be summarized in several key steps:

Step 1 - Connection

The process begins when the user enters their ID and password on the login screen and clicks the connect button. At this moment, a POST request is sent to the server side with the username and password in the request body.

Step 2 - Verification

The server receives the request, encrypts the password, and searches the database to see if there is a user with that ID and the encrypted password. If found, the server takes the user details, signs them with JWT, and returns the TOKEN to the client. If not found, it returns a 401 error.

Step 3 - Response Processing

The client receives the response. If the login was successful, it stores the returned token in Local Storage and navigates the user to the home page. If the login failed, it displays an error message on the user interface saying "ID or password is invalid."

Step 4 - Application Launch

On the client side, a Guard checks if a token exists in Local Storage. If a token exists, it sends it to the server side for verification. If no token exists or if an existing token has expired, the client is redirected to the login page.

Step 5 - Token Verification

The server receives the request with the token. It verifies the request using Jwt verify to check if it was created within the last hour. If not, it returns a 401 error. If yes, it returns the user details associated with the token.

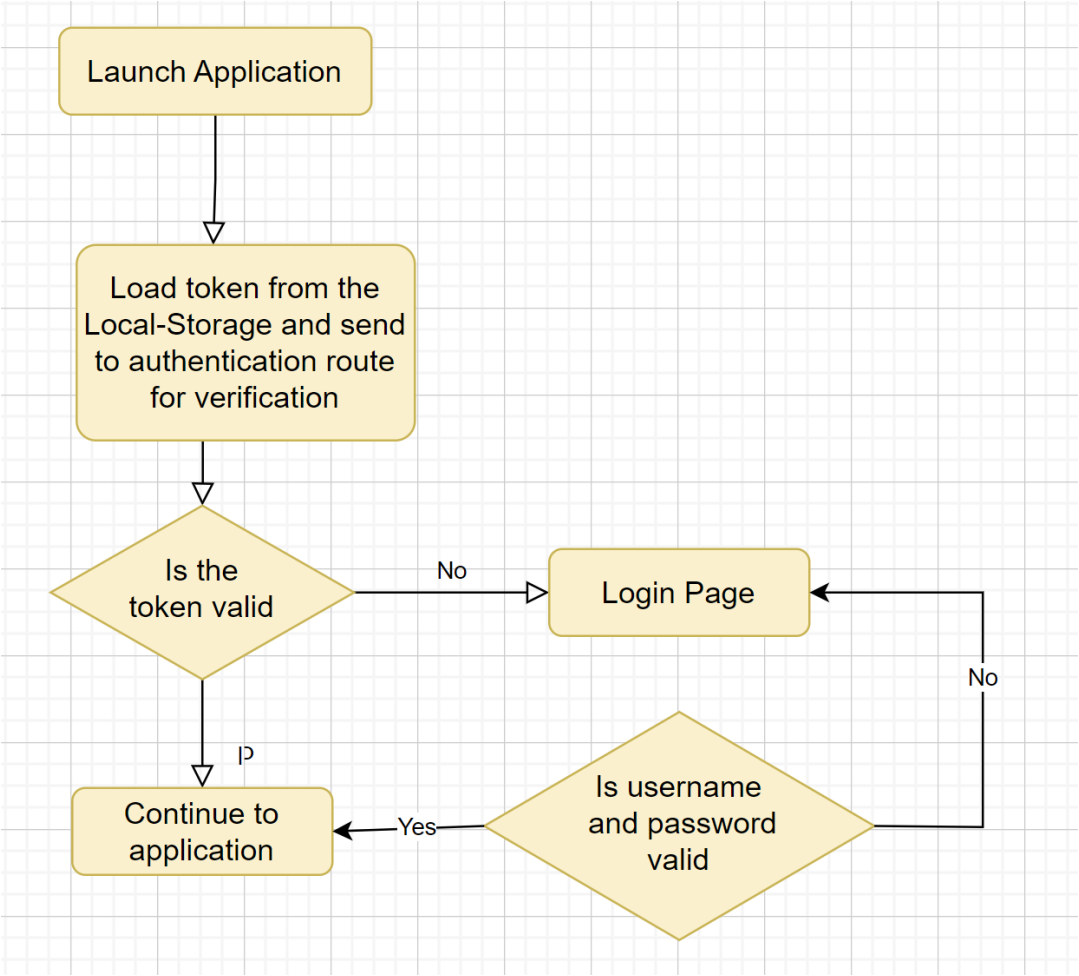


Figure 3 - logic process

Product Rate Process

The process you've outlined for managing a lending system with a product rating feature involves several steps, highlighting how users interact with the system, and how the server processes these interactions. Let's summarize and clarify the workflow:

Step 1- Request for Lending

A customer initiates a lending request for a specific product. It's important that the requested dates are in the future and that the end date is not before the start date; otherwise, the creation button will be disabled.

Step 2 - Approval by an Admin User

An admin user views the lending request on the lending management page. Upon approval, a PATCH request is sent to the server to:

1. Change the lending status from pending to approved.
2. Access the product schema and update the inventory quantity accordingly.

Step 3 - Product Rating by the Customer on "My Lendings"

Page The customer opens the "My Lendings" page, sees the request as approved, and finds an option to "Rate the product" on the leftmost column. Clicking this button opens a popup window for product rating.

Step 4 - Rating Action on the Server Side

After the customer rates and clicks submit, a PATCH request is sent to the server to:

1. Update the order's rating to the number chosen by the customer.

2. Add the customer's rating to the product's ratings array.

Step 5 - Viewing

On the products screen, the overall rating for a product is determined by the sum of all values in the ratings array divided by the number of values in the array.

This workflow demonstrates a comprehensive approach to handling lending requests and product ratings within a system, utilizing user and admin interactions alongside server-side processing to manage and update lending statuses, inventory quantities, and product ratings efficiently.

DB Architecture

Here's an overview of the database schema:

Category - Includes a table of categories, each described by a name and an image. The 'name' is a required string, and the 'picture' field is necessary for displaying the category image, essential for categorization.

Product - Contains products with fields for name, description, image, category (linking to the Category table), quantity, security rate, whether the product is returnable, ratings (a list of numbers), and company. Each product is linked to a specific category and has a unique identifier, enriching product information and categorization within the system.

Loan - Describes the loans table, linking a user and a product with start and end dates, loan approval, rating, and payment method. The 'isApproved' field indicates if the loan is approved, and the 'rate' field represents the customer's rating of the product after use. This model captures the essence of the lending process, documenting the essential details of each transaction.

User - The users table stores information such as phone number, password, ID, and role (admin or standard user). The 'role' field defines the user's role in the system, with a default value of 'User'. This distinction is crucial for managing access and functionalities within the system based on user roles.

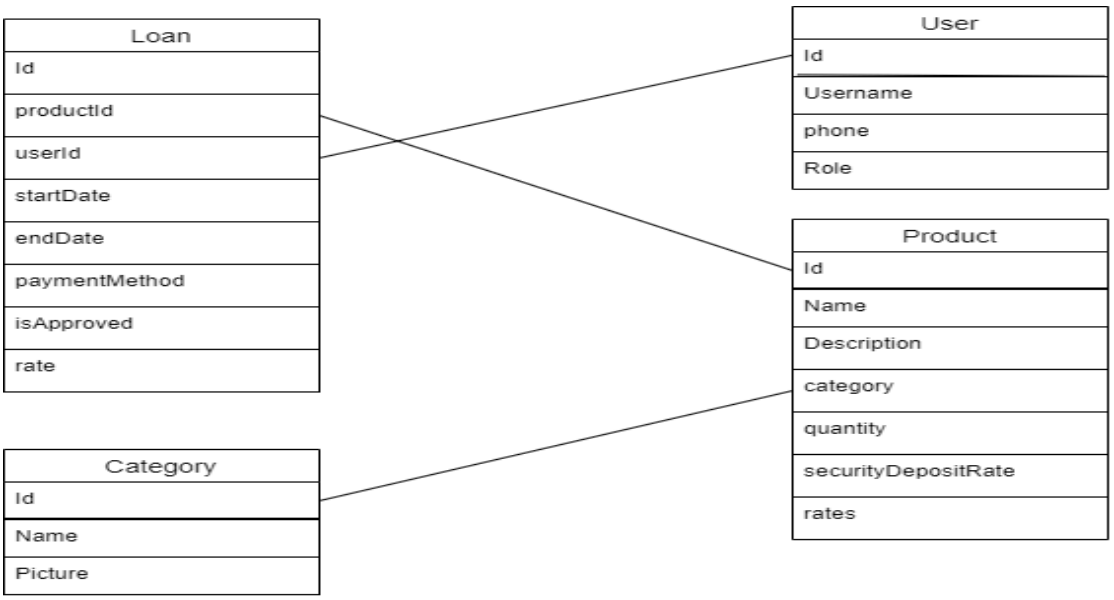


Figure 4 - DB Schem

Security

The application incorporates two main components in information security: Angular code for the client side and Node.js code for the server side.

Client Side

AuthGuard - This is a guard that checks if the user is authorized to access certain paths in the application. If not authorized, the user is redirected to the login page.

AdminGuard - Operates similarly to AuthGuard but also checks if the user is an admin and if they are authorized to access the admin interface sections of the application. If not, the user is redirected to the login page.

AuthService - This authentication service provides functionality for logging in, registering, checking permission levels, and whether the user is an admin. It's also responsible for storing data in Local Storage, such as the Token and UserId, and retrieving them.

During the initialization and setup of the routes in the application, permissions are defined for each path:

- Admin-only: Editing and creating products, approving loan requests, etc.
- Registered users only: Viewing products, opening a loan request, etc.
- Open to all: Registration and login.

Server Side

Registration - On a POST request to /register, the server receives details of a new user, encrypts the password using bcrypt, and saves the user in the database.

Login - On a POST request to /login, the server looks for a user in the database using the username, compares the entered password with the encrypted password in the database, and if the login is successful, generates a Token with user details and their role.

Permission Retrieval - On a GET request to /role, the server decodes the Token and returns the user's role (User or Admin).

Detail Retrieval - On a GET request to /details, the server decodes the Token and returns detailed user information.

In this process, the Token generated on the server is a JWT, a secure authentication medium containing the user's ID and role, allowing the server to verify the user's identity on each request. The Token is stored on the client side. This structured approach ensures secure user authentication and role-based access control within the application.

Middleware

Middleware is a function that allows intervening and performing actions between the request and the response in the data flow of the server code. In this specific case, two types of middleware, isAdmin and isUser, were created on the server side to manage access control.

isAdmin

- The isAdmin middleware verifies that the user is an admin before allowing the request to proceed to the next function (next()).
- It attempts to read the Token from the Authorization header of the request.
- If there is no Token, it returns an error response with a message indicating the absence of a Token.
- If there is a Token, it decodes it using the secret defined in the environment variables.
- If the Token is verified as valid and the role is "Admin", the request proceeds to the next handler (next()).
- If the role is not Admin, it returns a response with an access error.

isUser

- The isUser middleware functions similarly to isAdmin but allows both regular users and admins to proceed.
- Like isAdmin, it looks for a Token in the Authorization header.
- If the Token exists and is valid, and the role is either "Admin" or "User", it allows the request to proceed.
- If the role does not match, it returns an error response indicating insufficient permissions.

In both cases, if there is a problem with verifying the Token, such as if it is invalid or expired, an error response with a corresponding message is returned. These middleware functions are crucial for ensuring the application's security and preventing unauthorized access. In the event of an authorization error, an HttpStatusCode – 401 is returned.

Mobile Support

During the development of the application, a high priority was placed on ensuring optimal accessibility from mobile devices. This was achieved by utilizing the Inspect tool in the Chrome browser, which allows developers to examine, analyze, and modify the display of websites or applications to accommodate a wide range of screens and resolutions. Additionally, the @media screen feature of CSS was used, along with specific Angular tools that provide easy and quick access to design adjustments based on screen size.

Through the use of these tools, the interface and design of the application were adapted for mobile devices, enabling the identification and correction of display issues specific to these devices and ensuring a comfortable, intuitive, and fluid user experience across all types of devices. The process required focusing on mobile device optimization, including improving load times, adjusting text sizes and interactive elements, and ensuring the interface's responsiveness.

The result is an application that is accessible and user-friendly regardless of screen size, allowing users on various devices to enjoy the services and content it offers.

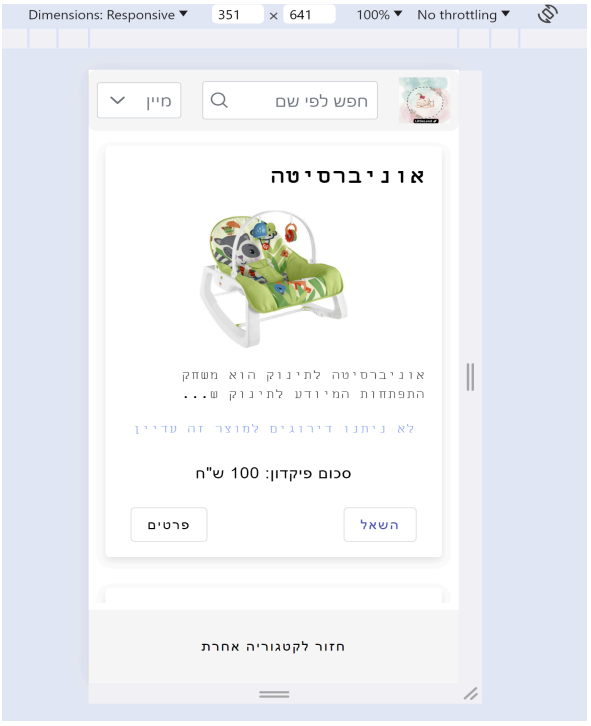


Figure 5 - Example to mobile development strategy

Work Planning

Task	Responsible of	Date Destination
Create DB	Sari	24/12/23
Create Server	Sari	31/12/23
Define Mongoose Models	Sari	07/01/24
Crate Services to make operations with DB	Sari	14/01/24
Connect the server to DB	Sari	30/01/24
Get Product –API	Sari	11/02/24
Search available Products	Sari	18/02/24
Create Products	Noa	07/01/24
Create lend	Noa	15/01/24
Get lends	Noa	22/01/24
Edit delete Product	Noa	31/01/24
Approve lend	Noa	11/02/24
Home Component	Avital	28/12/23
Login Component	Avital	30/01/24
Register Component	Avital	1/01/24
Create/Edit/Delete Category	Avital	7/01/24
Categories list admin	Avital	14/02/24
Categories component	Avital	19/02/24
Product Details	Avital	22/02/24
Open new lending component	Shirel	25/12/23
My loans component	Shirel	01/01/24
Rate Product	Shirel	08/01/24
Product-list-admin component	Shirel	15/01/24
Product-list component	Shirel	22/01/24
Create and edit product	Shirel	29/01/24
Delete Product	Shirel	06/02/24
Confirm lends component - Admin	Shirel	14/02/24

Client Side

Angular Capabilities

Interceptor - Allows for the interception of outgoing HTTP requests or the processing of incoming responses before they reach their caller. This is particularly useful for handling authentication, logging, or adding custom headers.

Guard - Used to protect routes within the application from unauthorized access by checking access permissions before allowing navigation to the route. Guards can determine whether a user may navigate to or leave a certain route.

Pipe - Used for data transformation in a format convenient for direct viewing from templates. For example, formatting dates, numbers, text, etc., without needing to alter the original data.

Service - Used to separate logic from components to improve maintainability and code sharing between components. Services provide access to data or business logic and can be injected into various components.

Component - The most basic building block in Angular, used to construct the user interface. Each component consists of HTML (template), CSS (styles), and a TypeScript file that defines the component's logic.

Resolver - Used to preload data before navigating to a specific component, ensuring that data is immediately available upon the user's arrival at the component. This prevents the display of a partial or empty user interface while waiting for data.

Pages

Register Page

This screen allows users to register in the system by entering their ID number, phone number, and password. If any of the parameters are incorrect, a corresponding message will be displayed, and the "Create User" button will be disabled.

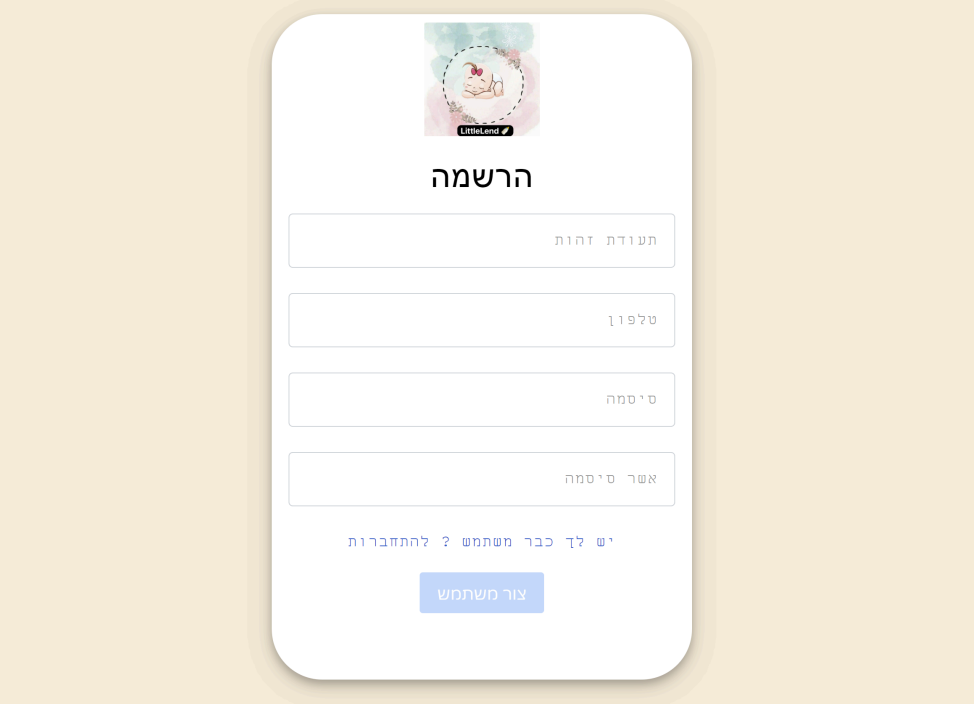
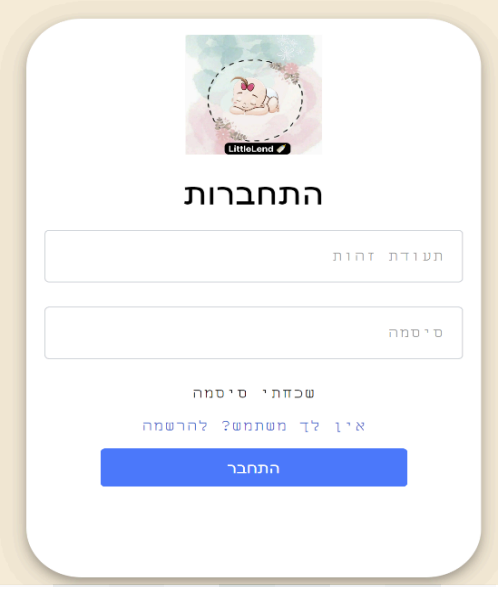


Figure 6 - Register screen

Login Page

This screen enables users to log into the system by entering their ID number and password. If the user does not exist, a corresponding message will be displayed. If the user does exist, the system will proceed to the home screen.



Reset Password Page

This screen facilitates password reset through the entry of an ID number.

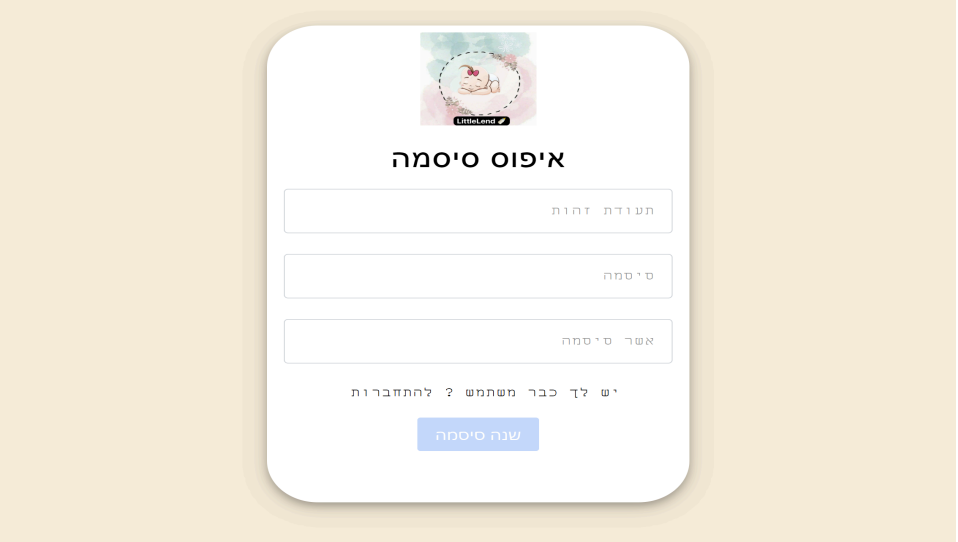


Figure 8 - Reset Password Screen

Home Page

The main screen of the application offers options to view available products, view personal loans, and if the user is an admin, there's an option to access the admin interface. There is a logout button located at the top left part of the screen.

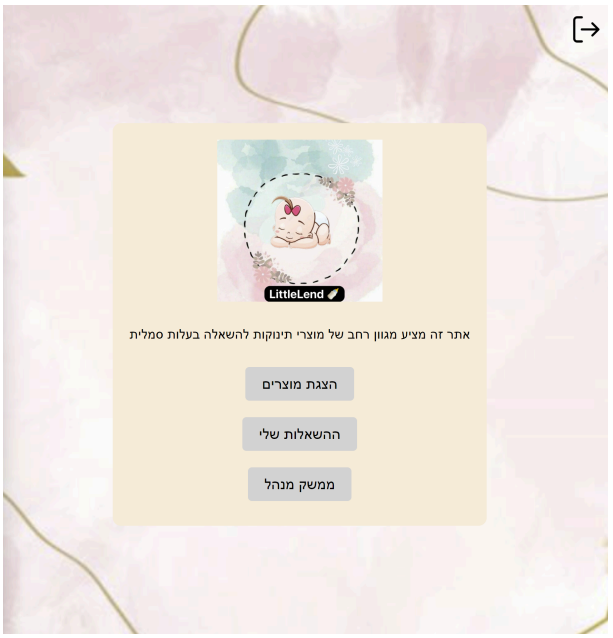


Figure 9 - Home Screen

Product Categories Page

This screen displays all the product categories available in the system for lending. It is responsive and presents each category as a card. Clicking on a button leads to the products screen of that particular category. At the bottom of the screen, there is a button that, when clicked, leads back to the home screen. Similarly, clicking on the icon at the top right corner will also navigate back to the home screen.

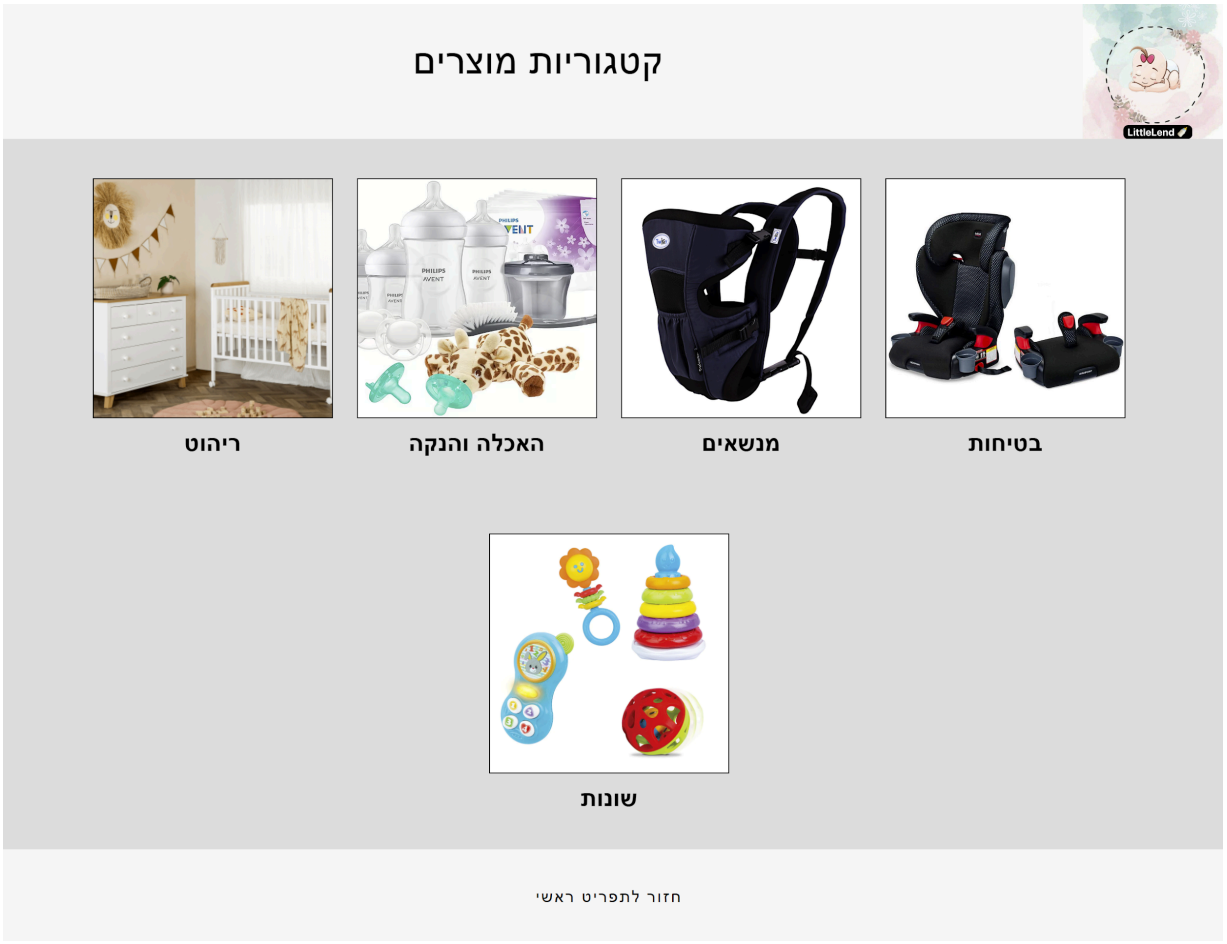


Figure 10 - Product Categories Screen

Products Page

This screen showcases all the products within a specific category available for lending in the system. It is designed to be responsive and presents each product as a card. On each product card, the image of the product, a description, ratings from previous customers, and "Lend" and "Details" buttons are displayed. If a product is not available in stock, it will be displayed as such, with appropriate labeling.

Users can filter the products by product name and also sort the products (from expensive to cheap, alphabetically, and by availability in

stock). At the bottom of the screen, there's a button that leads to the categories screen when clicked. Similarly, clicking on the icon at the top right corner will navigate back to the home screen.

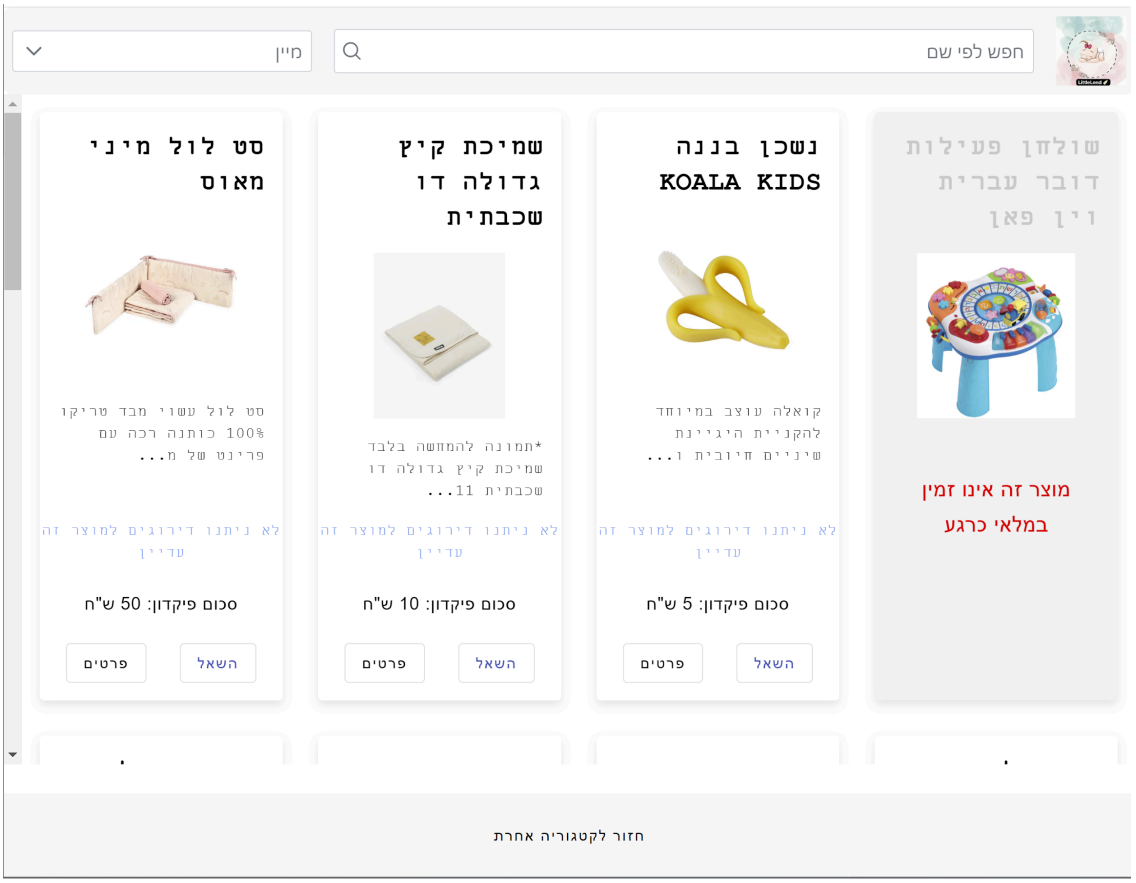
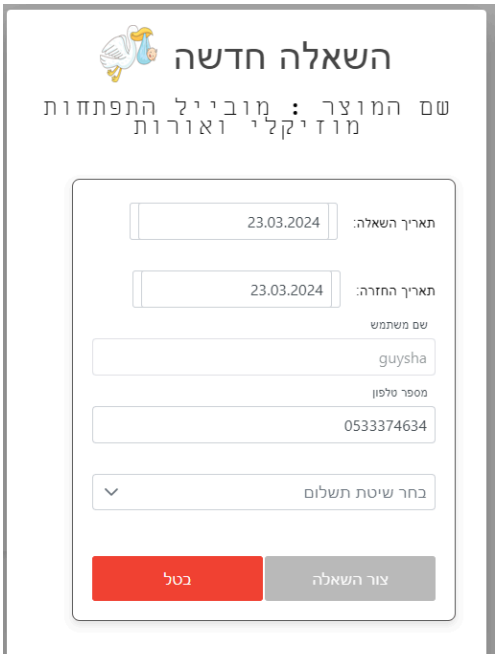


Figure 11 - Products Screen

Lending Product Page

The product lending screen opens as a pop-up and allows users to initiate a lending request for a product. Users must select a start date and an end date, enter a phone number, and choose a payment method. Validators on the form ensure that the phone number is valid and that the return date is set after the lending date. As



long as the form is not filled out correctly, the "Create Loan" button remains disabled.

My Loans Page

A table is displayed, showcasing the lending requests made by the user, detailing the product name, start date, end date, and the status of each request. Users can filter the requests by product name and sort the lending requests by date or by the status of the request. If a request has been approved, there's an option to rate the product. If a rating has already been submitted for a particular order, the text "Rating submitted for this product" will be displayed. At the bottom of the page, there's a button that, when clicked, will return the user to the main page.

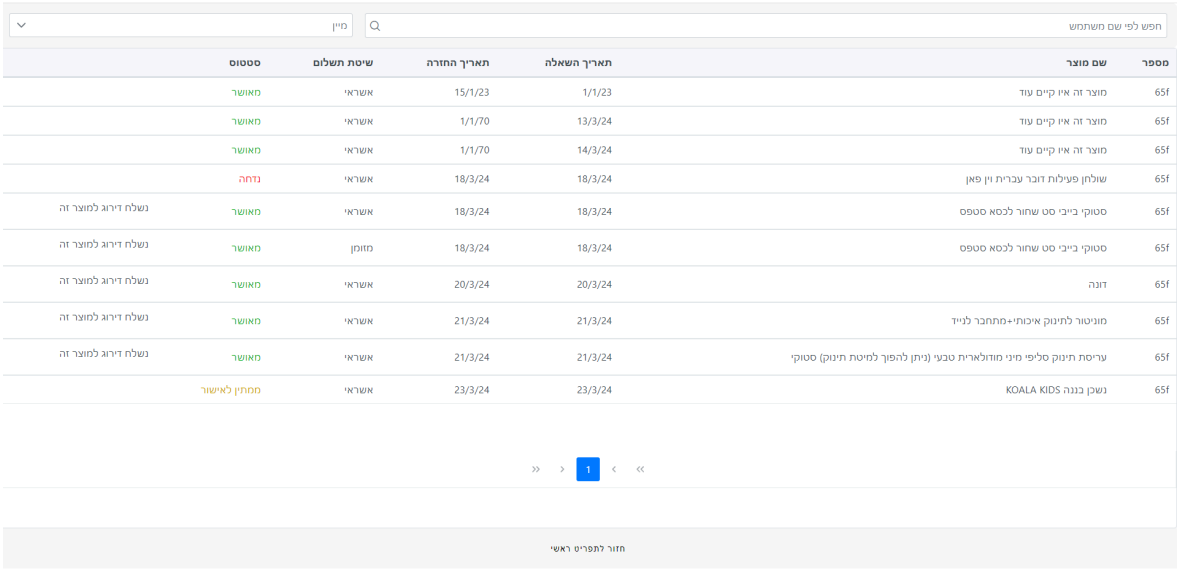
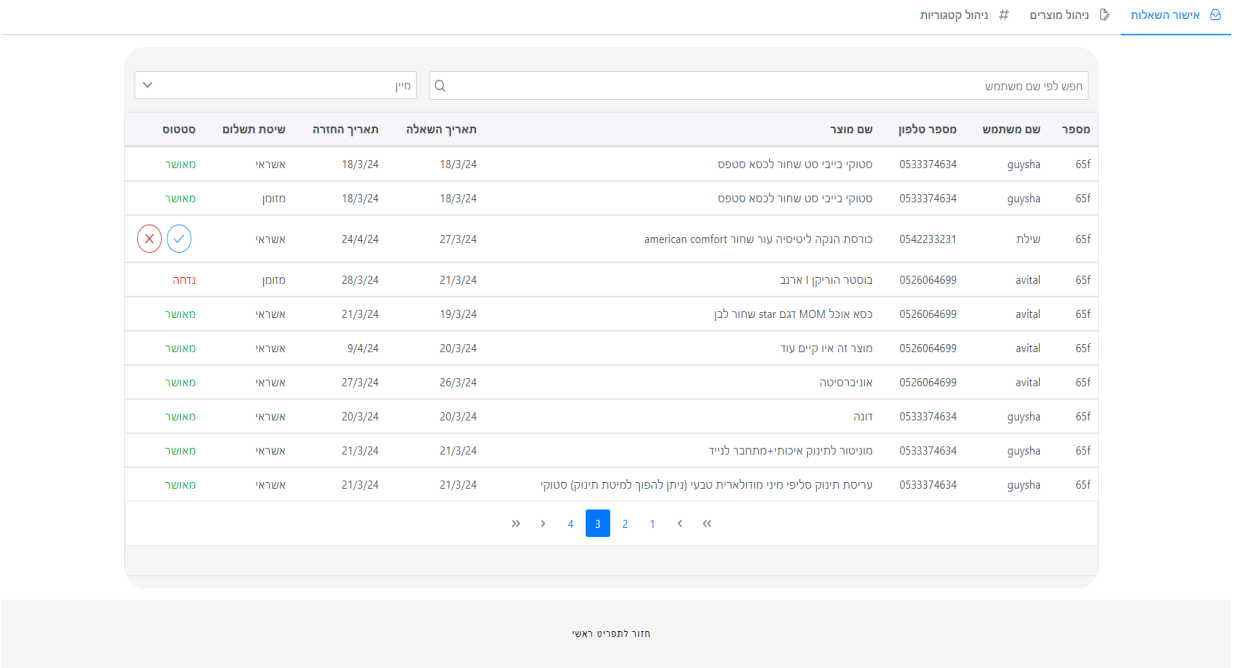


Figure 13 - My Loans Screen

Loans Confirmation Page – Admin Interface

A table is displayed, showing the loans by product name, customer name, phone number, start date, end date, and the status of the request. If the request status is "Pending Approval," there are buttons for approval and rejection. The administrator can approve or reject the lending request.

Users can filter the loans by product name and sort the loans by date or by the status of the request. At the bottom of the page, there's a button that, when clicked, will return the user to the main page.



מסך אישור השאלות - 13

Products Management Page – Admin Interface

This screen displays to the administrator all the products available in the system for lending. The screen is responsive and presents each product as a card. On each product card, the image, a description, and the deposit amount are displayed, along with "Edit" and "Delete" buttons, which allow for the deletion and editing of the product.

Products can be filtered by product name. Clicking on "Add Item" opens a pop-up window for creating/editing a product. At the bottom of the screen, there is a button that, when clicked, leads back to the home screen

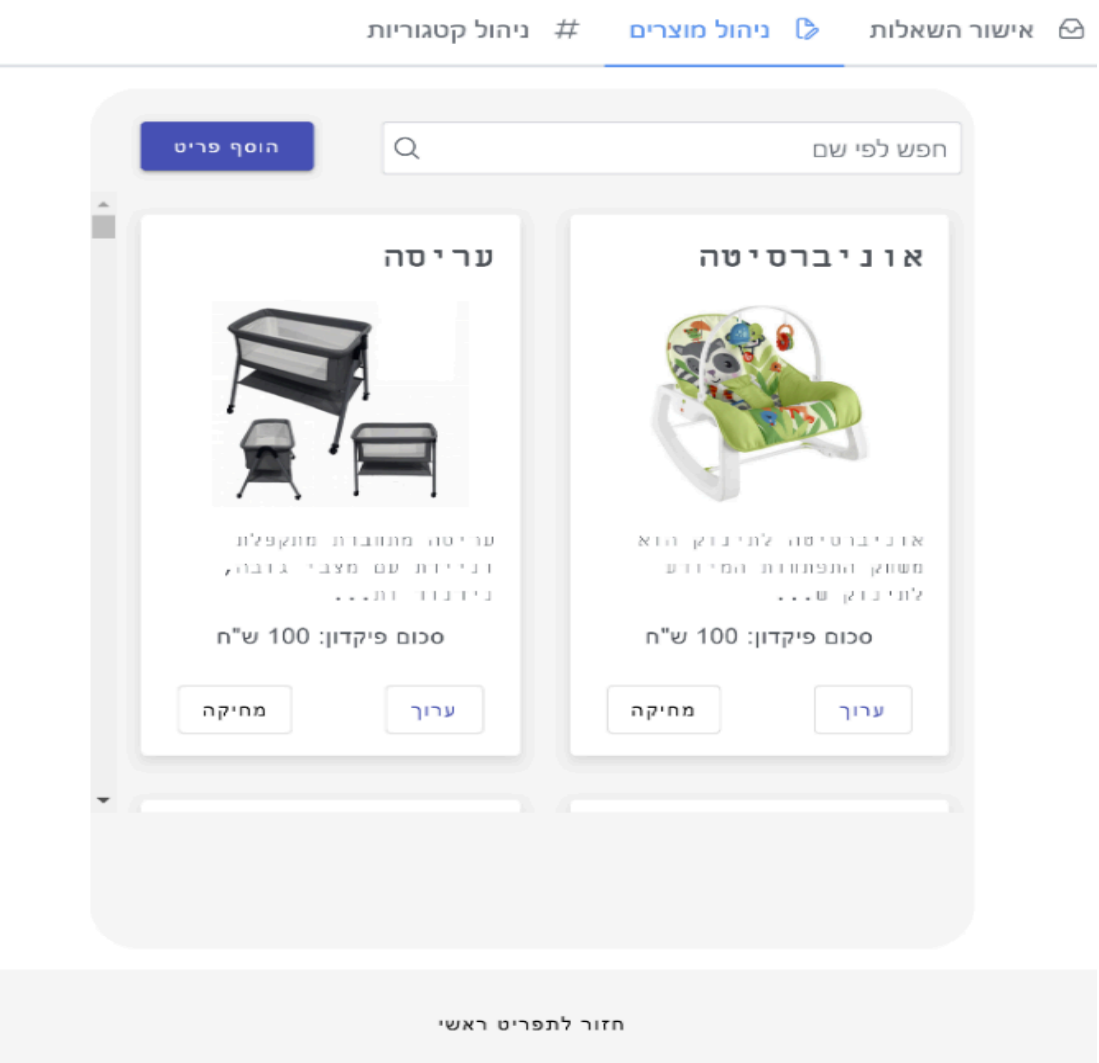


Figure 14 - Products Management Screen

Categories Management Page – Admin Interface

This screen presents the administrator with all the categories in the system. It is designed to be responsive and displays each category as a card. On each category card, the image, name, and "Edit" and "Delete" buttons are shown, allowing for the deletion and editing of the category. Categories can be filtered by name.

Clicking on "Add Category" opens a pop-up window for creating/editing a category. At the bottom of the screen, there is a button that, when clicked, leads back to the home screen.

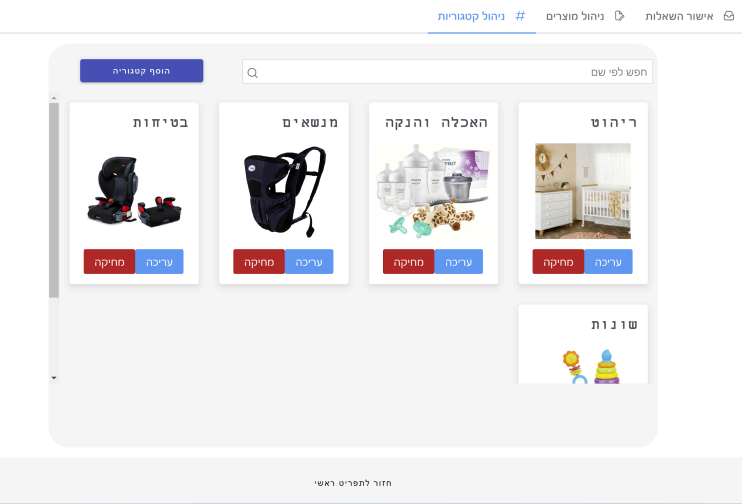
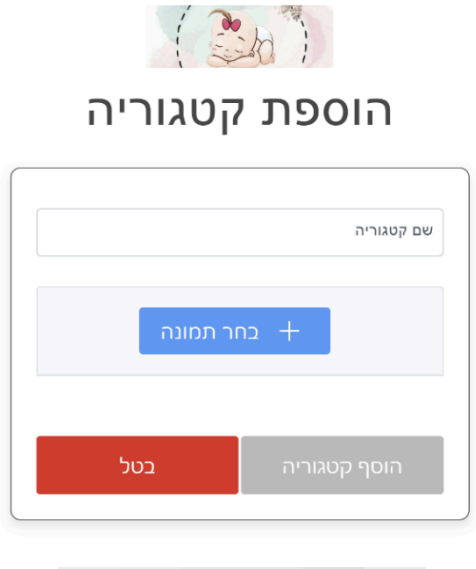


Figure 15 - Categories Management Screen

Create/Edit Category Popup – Admin Interface

The screen opens as a pop-up and is accessible only to users with administrator permissions. The form requires entering a category name and adding or changing an existing image. It also asks for the deposit amount and the quantity of the product in stock. There is an "Add Category" button (available only when the form is valid) and a "Cancel" button.



Create/Edit Product Popup– Admin Interface

The screen opens as a pop-up and is available only to users with administrator permissions. It features a creation/editing form that requires selecting a category from a drop-down list, entering a product name, providing a product description, adding or changing an existing image, specifying the deposit amount, and the quantity of the product in stock. There's also a requirement to indicate whether the product can only be returned as new (for example, wipes).

The form includes an "Add Item" button (available only when the form is valid) and a "Cancel" button.

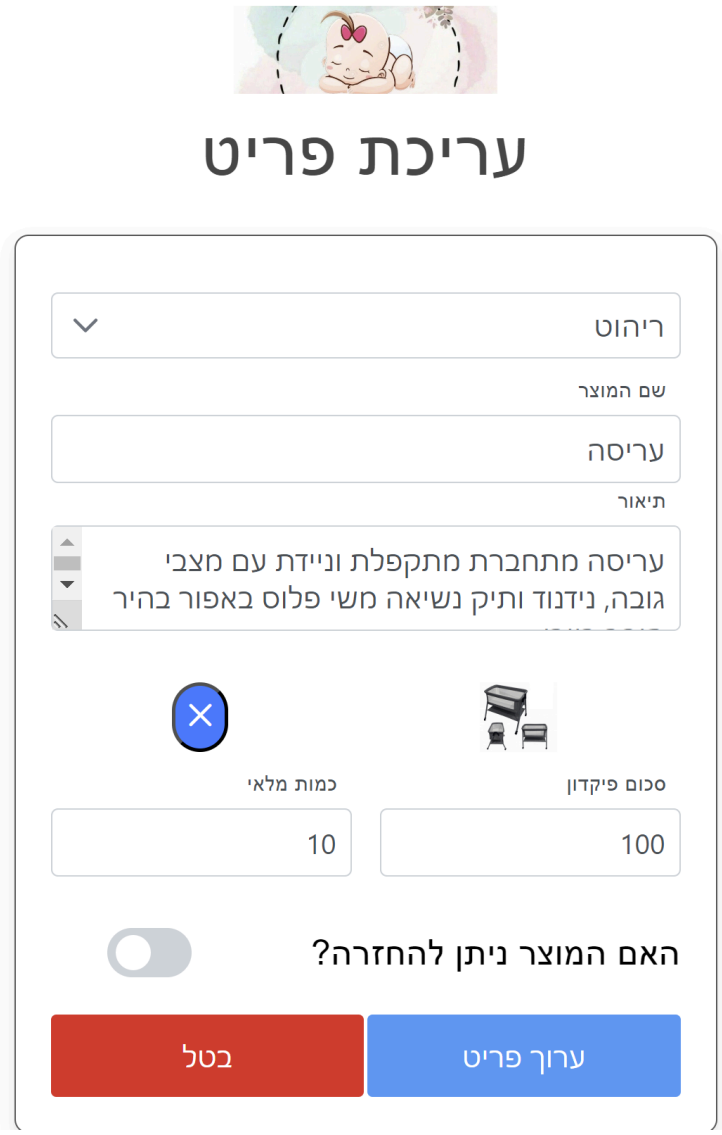


Figure 17 - Create/Edit Product Popup

Server Side

Description of the interfaces

On the server side of the application, there are four APIs responsible for the overall management of the system and for facilitating communication between all parts of the system.

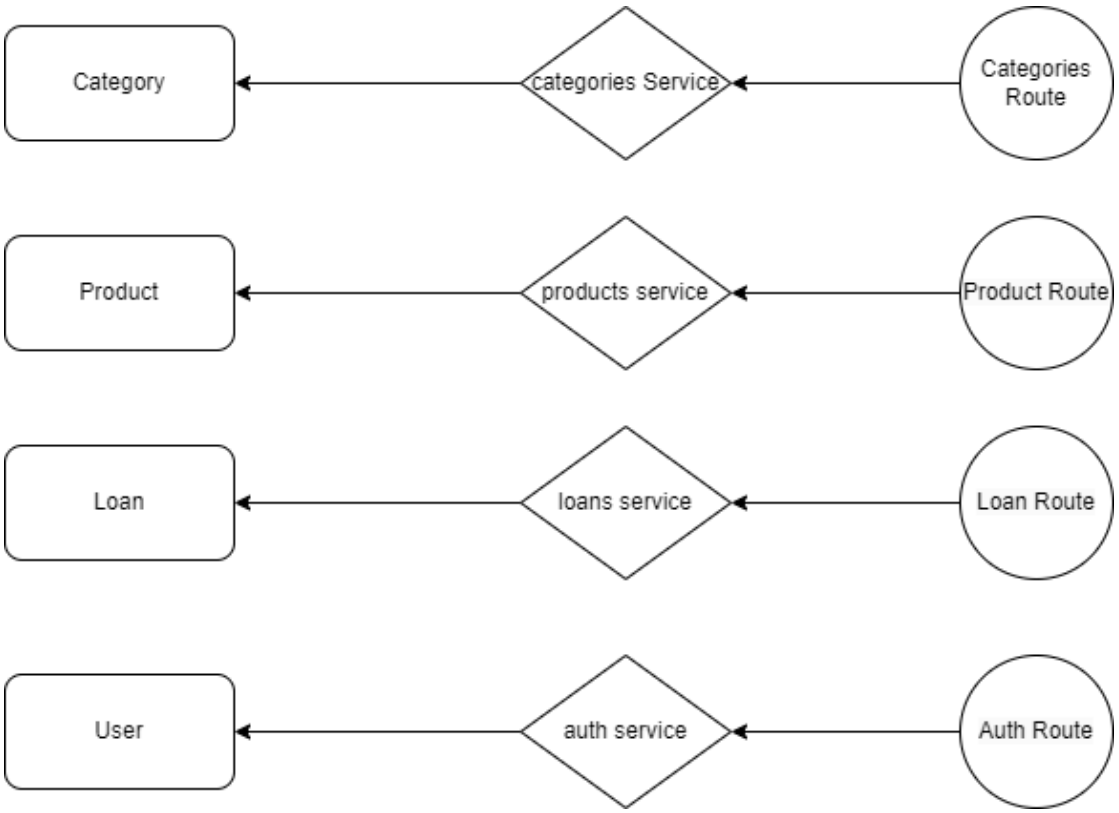


Figure 18 - Backend Interface Schema

Loans Interface

Route: api/loans

The lending interface describes the APIs for handling loans within the application, with each function corresponding to a different type of request and managed by lending and product services, using middleware for authentication.

- router.post('/'): Enables a registered user (after authentication with isUser) to create a new loan. The request body contains the details of the loan.

- router.get('/'): Allows only an admin (after authentication with isAdmin) to read the list of all existing loans. If there are no loans, it returns an empty list.

- router.get('/:userId'): Enables registered users to retrieve a list of loans by the specific user ID, received as a parameter in the path. The isUser middleware ensures authentication and authorization.

- router.patch("/:id/approve"): Allows an admin to update the approval status of a specific loan, with the loan ID provided as part of the path. If the approval is positive, the function also updates the available quantity of the products.

- router.patch("/:id/rate"): Enables a registered user to rate the loan after it has been approved. This rating also serves to update the overall rating of the lent product.

Overall, this interface facilitates the management of the entire loan request process - from creating a new request to updating its status and ratings after use. All routes use external services to execute the appropriate business logic and return information or a response to the user.

Categories Interface

Route: api/categories

The category interface outlines the APIs for managing categories within the application.
Each function corresponds to a different type of request and is managed by the category service, utilizing middleware for authentication.

- router.post('/'): Enables administrators to create a new category. This route uses Multer for handling image uploads and an S3 service for storing the image in the cloud. After uploading the image and receiving the URL, the information is saved in the database.

- router.get('/'): Returns all the categories from the database. Any user with user access can request and receive the list of categories.

- router.get('/:id'): Returns information about a specific category based on the identifier passed as a parameter in the path. Here too, any user with access can obtain the information.

- router.delete('/:id'): Provides administrators the ability to delete a category from the database by its identifier. Additionally, all products associated with this category are also deleted to maintain data consistency.

- router.put('/:id'): Allows administrators to update an existing category by its identifier. This process can also include uploading a new image and updating the image link in the database.

Each of these paths utilizes the category service and product service, which handle the product logic behind the scenes, providing the necessary information or performing the required actions on the database.

Authentication Interface

Route: api/auth

The authentication interface outlines the APIs for permissions and user management within the application. Authentication paths deal with critical processes of user registration and login, as well as operations related to authentication and retrieving user details.

- router.post('/register'): Allows new users to register in the system. This POST request takes the user's details (phone, password, ID) and saves them in the database after encrypting the password using bcrypt. If the operation is successful, the server returns a positive response.

- router.post('/login'): Enables existing users to log into the system. It authenticates the username and password by searching for the user in the database and comparing the encrypted password. If the details are correct, a JWT Token is generated and returned, allowing for the identification and authentication of the user in future requests.

- router.get('/role'): This path allows checking the user's role (for example, admin or standard user) by decoding the Token provided in the AuthorizationHeader. This is an essential tool for restricting access to certain resources in the application according to the level of permission (admin/user).

- router.get('/details'): This path is used to obtain detailed information about the user, such as ID and phone, through Token decoding. It allows the user to view their information in the application or to perform actions that require identity authentication or retrieval using userId.

Each of these paths uses the User Schema defined in the database and JWT to secure processes within the application.

Products Interface

Route: api/products

The product interface describes the APIs for managing products within the application. These routes are part of a product management system that includes capabilities such as viewing products, adding a new product, updating a product, and deleting a product. Each of these operations requires different permissions and uses specific services to execute the required logic.

- router.post('/'): Allows an administrator to add a new product to the system. The product can include an image, which will be uploaded to S3, and the URL of the image will be saved in the product's database.

- router.get('/'): Allows all authorized users to view all the products available in the system. If there are no products, it returns an empty array.

- router.get('/:category'): Enables viewing products by a specific category. This requires user authorization and allows the retrieval of products by a specific category passed as a parameter.

- router.delete('/:id'): Allows administrators to delete a product from the system by its ID. Product deletion is carried out by an administrator only and is final.

- router.put('/:id'): Allows administrators to update an existing product, including changing the image. The update can only be performed by an administrator and requires the product's ID passed as a parameter. If a new image is included, it will be uploaded to S3, and its URL will be updated.

The use of Multer and the S3 service facilitates the uploading and storage of images in the cloud, making the process more efficient and secure. Middleware such as isAdmin and isUser ensures that

operations are performed only by users with the correct permissions, enhancing the system's security.