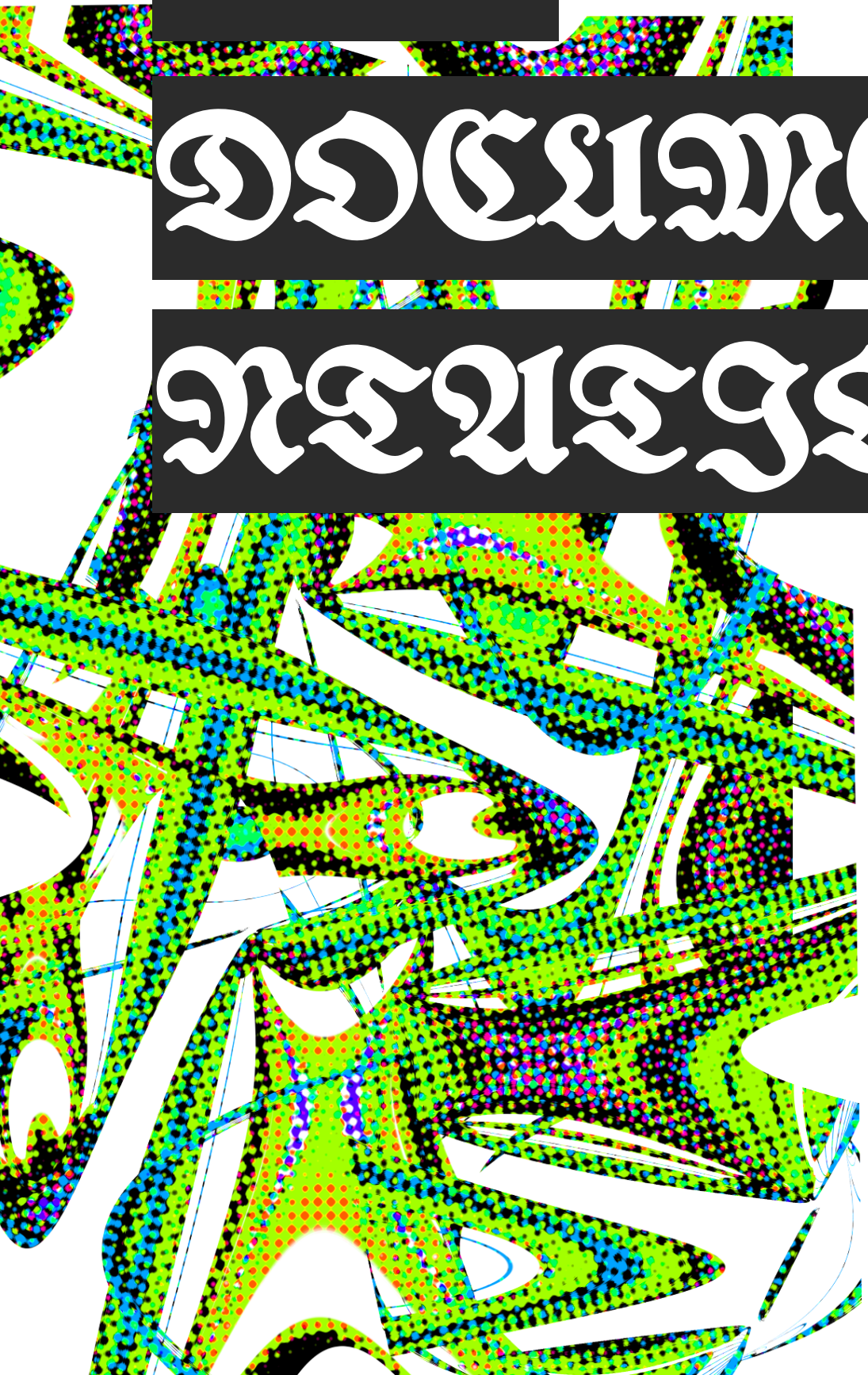
# DEV DOCUME NTATION



Developer Documentation // Webshop 2022 // lucaslichner.de

**Frontend**: React - No Additional Framework, written in JS not TS.
**Backend**: MongoDB Atlas, Node.js - Express.js
**ATLAS_URI**=mmongodb+srv://webTwoPointO:webTwoPointO@http://cluster0.w5zop.mongodb.net/webweb?retryWrites=true&w=majority

Atlas link is set in config.env in the server folder.
Conn.js Establishes the connection to MongoDB based on Atlas_URI

**Repo: https://github.com/oopera/WebAndMultimedia_2022**

# My-app // React Frontend

**App.js:**

**Use:**
Renders the Application, which in turn gets rendered by index.js, which sets the value of root - found in index.html
**States**:
***Basket***
- Array of Products which have been previously added to the Basket via updateBasket() in ProductFocus

***Comments***
- Array of all Comments

***Products***
- Array of all Products

***openedItem***
- String corresponding to currently opened FocusWindow

***isLoggedIn***
- false on load - contains all of the Account Data received from Login after login

***Loading*** 1 & ***loading*** 2
- boolean updated once all the data has been received on first render.

***Loadingscreen***
- boolean which flips once loading1 and loading 2 have been set false, ***controlled*** through timeout after loading the data.

**Functions:**
- 4 useEffects,
  - one to set local storage, one to fetch Products, one to fetch Comments and one to disable the Loading Screen
- useMousePosition
  - gets the current mouse position to render the circlePointer.

**Components:**
- HamNav
- TopNav
- BackgroundGrafix
- SideNav
- ProductList
- SubWindow

**Topnav.js:**

**Use:**

Displays user navigation on desktop view

Renders the TopNavigation in different states. Also gets used in HamNav triggered by mediaQueries through css.

**States**:

***form***

- Saves userInput in login form

***Reform***

- Saves userInput in registration form

***wantsToRegistre***

- Manages which form to show, based on if “registre” has been clicked

**Props:**

- Basket, is & setLoggedIn, opened & setOpenedItem

Use: Defines the User Inputs and Buttons for use in the Interaction with login and out, Basket, Account and Adminpanel. Gets reused in Hamnav,

**Components:**

- 4 render cases:
- 1 - not logged in and wantsToRegistre is false
  - Base case, with displays login form. Displays “login” “registre” and “basket”
- 2 - not logged in and wantsToRegistre is true
  - Registre case with registre form. Displays “registre” “close” (which returns to login form) and “basket
- 3 - logged in but not admin.
  - Base loggedIn case, shows no form displays “logout”, “account” and “basket
- 4 - logged in as admin.
  - Shows no form Displays “logout” “account” “admin” and “basket

**HamNav:**

**Use**. Defines the Container for Topnav tied to Media Queries - i.e. removes Topnav from the actual Top, and reuses it in the ArrowNavigation.

**States:**

***hamNav***

- Checks if hamnav is currently selected or not

***Help***

- Checks if the help Button has been selected.

**Props:**

- Basket, is & setLoggedIn, opened & setOpenedItem

**Components:**

- 4 render cases:
- 1 - help is true
- Displays sidenav guide. Button to trigger help only gets rendered on the smalles media query max width 500
- 2 - hamnav is false
  - Displays the arrow pointing away from the focuswindow to indicate opening the hamnav

- 3 - displays the arrow toward the focuswindow to indicate closing the hamnav
- Also displays the actual focuswindow, containing the topnav + help button on max width 500

**BasketComponent**

**Use**: Displays the items currently in basket, and facilitates the update of said basket

**States:**

***Rerender***

- Exists to trigger rerender in singular component

**Props:**

- setOpenedItem
- isLoggedIn
- setBasket
- Baket

**Functions:**

- removeFromBasket
    - Removes item from basket at position x (index given by basketitems map)

**Components:**

- Xbutton
    - View xbutton documentation
- BasketItems
    - Maps the Items in the basket

**SubWindow**

**Use**: This component exists to Allow easier navigation through the multiple possible Side/SubWindows containing: The Admin Control Panel, the Account Panel, and the Basket.

**Props:**

- isLoggedIn
- setIsLoggedIn
- setBasket
- Basket
- Products
- setProducts
- openedItem
- comments
- setComments

**ProductList**

**Use:** Displays the Productlist, manages which Item is focused and rendered based on click.

**States:**

***searchInput***

- Contains userinput in searchfield, products shown get filtered based on input

***openedProduct***

- Saves the ID of currently focused product, saves null if nothing is focused

**Functions:**

- updateProduct
    - manages setOpenedProduct value, props is the productID as string. If the same productID that is already saved in state gets passed, set null

**Components:**

- SearchField
  ProductList
- Product
- Productfocus

**Product**

**Use**:
Renders ProductList item based on product given

**ProductFocus**
**Use:**
-This component gets passed product information and displays all relevant information on screen for a selected product
-Disables product functions for specific statements. Items are unable to be added if the availability is <= 0. Comment field is not displayed if the user is not logged in and/or if the user has not bought an item.

**Functions:**
updateBasket()
- Updates the product basket with passed product information

**CommentList**

**Use:** This component maps the comments of a specific product, which gets called in ProductFocus

***The following functions are part of the HelperFunctions and are used to sort the source code and make it more readable***

**SessionFunctions**

**Use:**
-The functions located in SessionFunctions.js are used for the websites session management

**Functions:**

- clear()
  - Clears local storage

- clearCache()
  - Clears the local storage with response of on input of “clear cache” button in AccountWindow

- setStorage()

- If the user is logged in, set the storage to the currently logged in user, set hasData to true
- If the user is not logged in, but hasData is true, load user from local storage

**ProductFunctions**

**Use:**
-The functions located in ProductFunctions.js are used for the websites product management

**Functions:**

- sendComment()
  - Creates new comment object from props
  - Posts new comment on database
  - Saves comment from database (including unique ID) and adds it to the user
  - Updates the user in the database with new Data - Both adds comment to comment collection and updates user in users collection

- purchase()
  - Checks availability for each basketItem and updates availability for each Item based on amount sitting in basket
  - Updates each products availability in database
  - Creates purchase object, posts purchase object to purchase database
  - Saves purchase document to variable, adds it to user and updates user in database

- deleteProduct()
  - Filters all comments to comments left on chosen product
  - Deletes all comments found
  - Deletes products and updates comments and products state

- addProduct()
  - Checks for invalid inputs (no name, duplicate name, no description, negative price)
  - If availability is negative, set to true
  - Creates productForm if all input is valid and finally add product to database
  - Updates product state to [] - reload of products

- updateProduct()
  - Updates product to information passed via from prop, sets productstate to [] to trigger reload

**AccountFunctions**

**Use:**

-The functions located in AccountFunctions.js are used to sort all account related methods in one file

**Functions:**

- updateUser()
  - Updates user to information passed by props (i.e. currently logged in user)
  - This makes it so react state lines up with database information

- login()
  - Calls login route with form passed by props
  - If email & password combination exists, set logged in to received user object

- register()
  - Checks for valid input
    - Password length
    - Password == Password2
    - E-Mail being a functional E-Mail
  - Calls wantsToRegistreFunc
    - Changes TopNav state so the registre form is no longer shown
  - If registration is successful sets login form to E-mail and password so user can login seamlessly
  - If route returns false, either username or E-Mail is already in use

- addUser()
  - Almost duplicate of register, but resets AdminInputs to “” and does not call wantsToRegistreFunc
  - Also updates user state which is only available in AdminComponents

- wantsToRegistreFunc()
  - Sets state in TopNav

- logout()
  - Sets logged in state to false (like on load)
  - Closes account or admin window if opened, since those can no longer be accessed

- deleteUser()
  - Filters user state down to user with passed iD (selectedUser in AdminControl)
  - Maps over all comments by a specific user and calls deleteComment for each comment
  - Deletes user and resets comments and users state

- deleteComment()
  - Deletes comment
  - Checks if comment.id exists, if not call with _id if it does call with id
  - Safety measure in case old data with differing structure is still present in database

- deleteAccComment()
    - Deletes a comment from user information, does not get called when product gets deleted, since it is not necessary
    - The comment is no longer displayed, but the user might still want to see it
    - Call deleteComments, splices the users comments at index of comments. Resets isLoggedIn state to updated UserInfo
- deleteOwnAccount()
    - Deletes account of the currently loggedIn user, and all comments linked to that account.

**/images**

- This Folder Contains all the Image Resources which are not linked to Products. To see where images for products are saved compare @Product

**SideNav**

-The SideNav div contains a simple user guide on how to use this webshop. It is divided under several helpful aspects such as how to register and login as a user and as well a short disclaimer not to use real data.

**BackGroundGrafix**

**Use:**

Displays Logo and MainText

**AdminControl**

**Use:**

-This component renders the AdminControl panel and contains methods which allows the user, as long as the user is logged in as an admin, to edit server side components such as adding and deleting users and products.

**States:**

***Users***

- Saves the user in an array

***Purchases***

- Saves the purchases in an array

***isChecked***

- Checks if user is an admin (Default: false)

***selectedUser***

- Shows the selected user, default empty

***selectedProduct***

- Shows the selected product, default empty

***searchInput***

- In this state the admin is able to search for a user account registered to this website

***openWindow***

- This state rerenders the window depending on which component has been opened

***userForm***

- Similar to the reform state from @TopNav, contains a form to input user information for registering

***productForm***

- Contains a form for uploading new products to the websites database, contains product attributes name, description, price, availability and optional img data.

**Functions:**

- checker()
  - Changes state for Admin Checkbox in AddUser()
- selectedUserChanged()
  - Updates selectedUser for userID
- selectedProductChanged()
  - Updates selectedProduct to productID and sets ProductForm to information of product with that ID.
  - If selectedProduct is newProduct, setForm to “empty”
- updateForm(), updateReform()
  - Updates forms
- getUsers() - AccountFunctions.js
  - Fetches all user data from mongoDB database
- getPurchases() - AccountFunctions.js
  - Fetches all purchase data from database
- addUser() - AccountFunctions.js
  - Adds user to database by filling out the form in AdminControls, reuse of register() function
- deleteUser() - AccountFunctions.js
  - Deletes selected user from database
- deleteComment() - AccountFunctions.js
  - Deletes comment if comment.id exists, method for safety reasons in case old data with different structure is still present in database
- addProduct() - ProductFunctions.js
  - Checks for invalid inputs (no name, duplicate name, no description, negative price)
  - If availability is negative, set to true
  - Creates productForm if all input is valid and finally add product to database
  - Updates product state to [] - reload of products
- deleteProduct() - ProductFunctions.js
  - Filters all comments to comments left on specific product
  - Deletes all found comments
  - Deletes products and updates comments and products state

**Canvas**

**Use:**

-Renders Canvas in AdminControl, shows a graph of the websites sale statistics
-Also sets height and width dynamically so that the canvas does not overflow with higher activity

**Functions:**

- updateCounter()
  - Usage for consistent distance between bars and items and knowledge of the next items position

- useEffect()
    - First batch renders header informations with sales and comments
- render()
    - Renders information of specific product
- renderNumbers()
    - Renders the guideNumbers at the bottom of the graph

**AccountWindow**
**Use:**
-This component is used for the display of AccountWindow, it contains Comments and Purchases

**Functions:**
- deleteOwnAccount()
    - Props: isLoggedin, setLoggedIn, setComments, setOpenedItem
    - Logs out the user, deletes all of the users comments, and closes any opened Subwindow

- clearCache()
    - Clears cache of logged in user, so it will not be saved in local storage and user will be logged out

**AccPurchaseList**
**Use:**
-This component is used to map the purchases done by an account in a table format in AccountWindow

**AccCommentList**
**Use:**

-Similar to AccPurchaseList component is this one used to map comments for an account in AccountWindow

**XButton**
**Use:**

-This component is used as a small helper method in different components, such as AccountComponents, AdminComponents, ProductComponents and TopNavComponents

---

# Server // Node&Express Backend

Routes:
**@/updateUser/:id**
Use: Updates a Product by ID. Props is the data of the updated Product.
**@/updateProduct/:id**
Use: Updates a Product by ID. Props is the data of the updated Product.
**@/users/login**
Use: Returns Userdata for login data. If credentials are wrong, return false. Props are email and password for the account.
**@/users/add**
Use: Logs comment to comment collection. Props is the user to log.
**@/comments/add**
Use: Logs comment to comment collection. Props is the comment to log.
**@/products/add**
Use: Logs product to product collection. Props is the product  to log.
**@/purchases/add**
Use: Logs purchase to purchase collection. Props is the purchase to log.
**@/comments/:id**
Use: fetches comment by id. Call with id in route, return comment.
**@/webweb/users**
Use: fetches all users. No props, return array of comments.
**@/webweb/comments**
Use: fetches all comments. No props, return array of comments.
**@webweb/products**
Use: fetches all product. No props, return array of comments.

# MongoDB database

**Users**
**Use**: Saves Documents of Users
**Values**:
{*String*} Email: email of the user
{*String*} Password: bcrypt hash of the users password.
{*boolean*} Admin: controls admin privileges
{*Object*} {*Array*} Purchases:
{*String*}{*Array*} Products: saves the Products bought
{*float*} cost: saves price of the purchase
{*Date*} date: saves date of the purchases
{*ObjectID*} purchaseID: saves purchase ID
{*Object*}{*Array*} Comments:
{*String*} Comment: saves Comment
{*String*} Item: Saves item which has been commented
{*ObjectId*} id: Saves comment ID

The User Documents have a lot of redundancy built in by design. Account information should be precisely and easily fetched and found at a centralised location. Purchase and Comment IDs are still present to be referenced when deleting a comment or the Account.

**Products**
**Use**: Saves Documents of Products
**Values**:
{*string*} name: name of the product
{*string*} description: description of the product
{*float*} price: price of the product
{*int*/*boolean*} Availability: Availability of the product
{*string*} img: link of the productimage

Availability can be saved as int or boolean so that an infinite Availability can be displayed (downloads)
As images are not 100% necessary Image can be null or undefined, both is accounted for in the code.

**Purchases**
**Use**: Saves documents of purchases
**Values**:
{*date*} date: date of purchase
{*float*} price: total price of Purchase
{*string*} email: email of ther user who made the purchase
{*string*} {*array*}: array containing the names of the items

Items are saved as strings so that the deletion of an item does not interfere with a purchase.
**!**Purchases **need** to be save from deletion of other Documents**!**

User is saved as Email so that in case the user gets deleted after purchase, user can still be contacted. Saving both email and ID would be redundant as the email is unique.

**Comments:**
**Use**: Saves documents of Comments
**Values**:
{*String*} Name: Username of commenter
{*String*} Comment: Comment written by user
{*ObjectID*} productID: ID of Product the comment was placed under.
{*ObjectID*} userID: ID of commenter.

Name of the commenter is saved in the comment, so that the username must not be fetched for rendering the Product.

---