mister-toy !(yourToy)

Setup

1. Setup the project folder: *mistertoy*-proj
2. This is going to be an end-to-end project so we will eventually have two folders inside the project's folder: *mistertoy-frontend* and *mistertoy-backend*
3. Inside the project folder, create the *mistertoy-frontend* folder and initialize the frontend Git-repo
4. We will use a separate Git-repo for the frontend and for the backend

Note: In this project, the Git log should be meaningful and present the progress of the development work. It is recommended to use git branches for every feature we develop.

Part 1 – Frontend First!

Here is an initial model:

```
const labels = ['On wheels', 'Box game', 'Art', 'Baby', 'Doll', 'Puzzle',
'Outdoor', 'Battery Powered']

const toy = {
    _id: 't101',
    name: 'Talking Doll',
    imgUrl: 'hardcoded-url-for-now',
    price: 123,
    labels: ['Doll', 'Battery Powered', 'Baby'],
    createdAt: 1631031801011,
    inStock: true,
}
```

Basic Frontend

Build a frontend from scratch.

- Use the **CLI** inside the project folder and create a project named **mistertoy-*frontend***
- Commit and push the code
- implement full CRUD, manage the state with a store.

You should have the following:

1. store
2. toyService
    a. We kick off the frontend first using a service that works with storageService which provides an async access (CRUDL) on a collection kept to the browser's localStorage)
    b. We will later convert this service communicate remotely with our backend via AJAX
3. <ToyDetails> (Page)
    a. This page renders full details about the toy
4. <ToyEdit> (Page)
5. <ToyIndex> (Page)
    a. <ToyList>
    b. <ToyPreview>
    c. <ToyFilter>
        i. By name (use debounce)
        ii. In stock (remember that there are 3 states here: true / false / undefined (all))
        iii. Toy label multiselect dropdown
        iv. Sort by: name / price / created

(git) commit your work: "Frontend basic functionality"

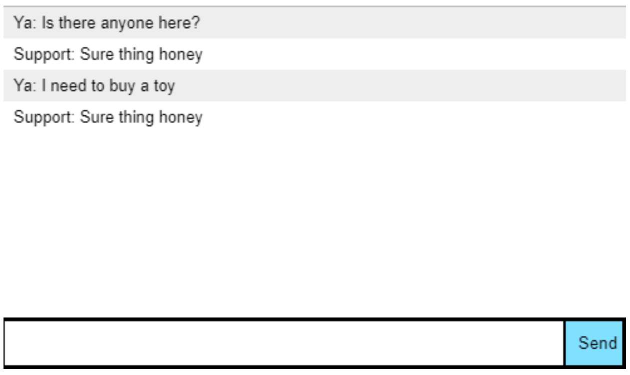
Children

Nice Popup

Build a nice-popup component

- Esc key (on the body) should close the popup
- The popup should have 3 elements: *<header>, <footer>* and <main>
    - The children should be placed in the <main>
    - The heading and footing props are placed accordingly
- Inside <ToyDetails>, add a chat icon that opens the popup:
    - Inside the popup, render a <Chat> component.

- For now, it just allows the user to enter some text that appear in a list of msgs (local state). Also, anything the user says is auto responded after a short delay:

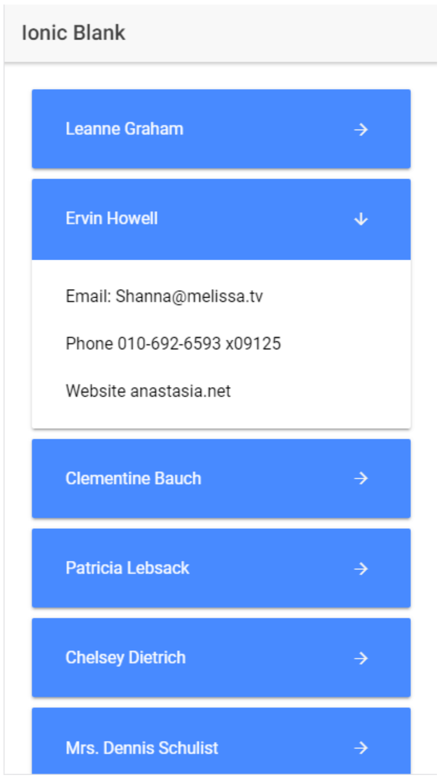

Ya: Is there anyone here?
Support: Sure thing honey
Ya: I need to buy a toy
Support: Sure thing honey

Custom hooks

Use the custom hooks displayed in class to add online-status and exit-while-unsaved-changes behavior to the app

BONUS – Improve the Accordion

Improve the Accordion component



Email: Shanna@melissa.tv
Phone 010-692-6593 x09125
Website anastasia.net

- Add your own design and a sliding animation

Bonus: Add a backend

{coding
academy

1. Inside the *mistertoy-proj* folder, create the *mistertoy-backend* folder
2. Setup Git-repo for the backend, add, commit and push
3. Provide an API for CRUD based on a JSON file
4. Use the inClass project as a reference and Use postman to test the API

Best strategy

1. npm init --yes
2. Set up a basic express application
3. Copy & Paste & Refactor yourself a backend toyService
4. LIST toys:
    - Create a request for GET */api/toy* in Postman and watch it failing with 404 NOT FOUND
    - Implement endpoint GET */api/toy* that returns all toys
    - Test with Postman
    - Add basic *filterBy* support
5. READ toy
    - Create a request for GET */api/toy/:id* in Postman and watch it fail
    - Implement endpoint GET */api/toy/:id* that returns a specific toy
        - This endpoint should add a dummy "msgs" property to the returned toy object. For now use some hardcoded msgs in the backend toyService
    - Test with Postman
6. DELETE toy
    - Create a request for DELETE */api/toy/:id* in Postman
    - Implement endpoint DELETE /api/toy/:id that deletes a toy
7. CREATE toy
    - Create a request for POST */api/toy* in Postman
    - Implement endpoint POST /api/toy that adds a new toy
8. UPDATE toy
    - Create a request for PUT */api/toy* in Postman
    - Implement endpoint PUT /api/toy that updates the toy
9. Refactor the frontend's *toyService* to work with the backend via AJAX

* Note: The frontend runs on a different port than our backend – so remember to allow CORS and to use the provided http.service

{coding academy

Part 2 – Awesome mister-toy

Use Community components and libraries.

Let's use some community components and libraries, use the ones demonstrated in class, you can also add some other libraries

Dashboard Page

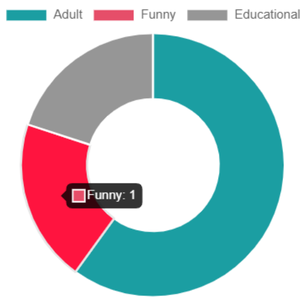
Add a dashboard page with charts:

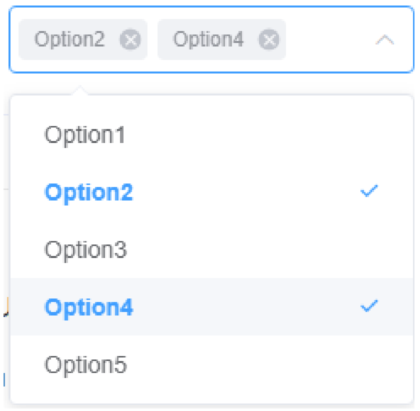
- Prices per label (Art, Baby, etc.)
- Inventory by label – Chart showing the percentage of toys that are in stock by labels
- Generate some random numbers and dates for a line chart

UI Components

Use various UI components

Example: in the <ToyFilter> use a *select* component such as:



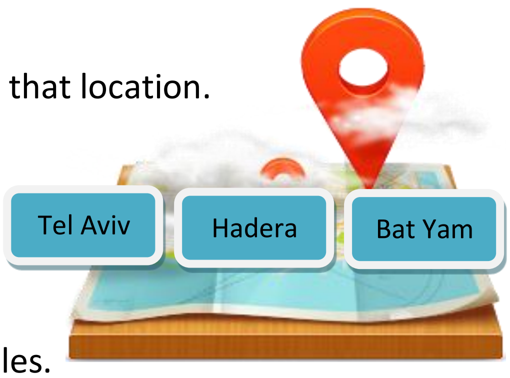


Form Validation

Validate the inputs using a validation library

Maps

- In the About page, show a map with markers for the shop branches.
- Each branch will appear as marker on the map.
- When user clicks a branch button the map is centered on that location.



Bonus: Internationalization

Add i18n support, allow the user to switch between the locales.

Part 3 – Beautiful mister-toy with CSS

- Use a full CSS architecture
- Level-up your CSS:
  - Use nesting
  - Use variables
  - Use currentColor and color-mix
  - Use advanced layouting (grid, subgrid)
- Make it look amazing on desktop, tablet and mobile

Part 4 – Toys and Users with Mongo

Story

- We need the shop owner (admin) to be able to manage the shop
- We need normal users to be able to add msgs about toys

General

Use async-await, try-catch across the app

In this exercise, start from the mister-backend project (reviewed in class), and add a toy route (under the API folder) for be used by the frontend.

- Add a toy mongodb collection
- Add a toy.service
- Add a toy.controlller
- Add a toy.route

Check the backend from *postman*

Support authentication

- Add a user collection (_id, fullname, username, password, isAdmin), have one admin user (isAdmin: true)
- Add a login page
- Only the admin user should have the Edit/Delete/Add options
- Protect the relevant routes using a middleware

Add msgs support

- Inside the toy, add a msgs array:

```
const toy = {
    _id: "t101",
    name: "Talking Doll",
    price: 123,
    labels: ["Doll", "Battery Powered", "Baby"],
    createdAt: 1631031801011,
    inStock: true,
    msgs: [
        {
            id: 'm101',
            txt: 'Great toy, how much',
            by: {
                _id: 'u101',
                fullname: 'Puki Ja'
            }
        }
    ]
}
```

- In <ToyDetails>
  - Display the current toy's msgs
  - Allow logged-in user to enter a msg
    - Add a route: POST /api/toy/:id/msg
    - When adding a msg, use $push to add it to the collection

{coding
academy

Part 5 – Reviews

MongoDB Aggregations

Let's add another feature: the user can enter a review (this is a separate feature from msgs)

In this case, we will use another collection for keeping the reviews

The review collection keeps documents such as:

```
{
    "_id": "5bfa538166597429743c1ff0",
    "userId": "5b507e97f20dd52bb6e67a44",
    "toyId": "5b4f0b081043ae5f9cf3494c",
    "txt": "Best toy ever!"
}
```

Let's use aggregations.

Aggregation of review, toy, and user

Aggregate reviews with users and toys and get the following output:

```
{
    "_id": "5bfa538166597429743c1ff0",
    "txt": "Best toy ever!",
    "toy": {
      "_id": "5b4f0b081043ae5f9cf3494c",
      "name": "Talking Doll",
      "price": 19779
    },
    "user": {
      "_id": "5b507e97f20dd52bb6e67a44",
      "fullname": "Puki Ja"
    }
}
```

1. In <ToyDetails> display the current toy's reviews and allow a logged-in user to enter a review
2. In <UserDetails> display the user details and all his reviews
3. In <ReviewExplore> show all the reviews in the system and allow filtering

Part 6 – Build and Deploy

- Create an Atlas account
- setup a database to be used by the app
- Set up the Atlas database user/password
- Set up network access
- Get the connection string
- Connect the compass app to the cloud database
- Test that the backend can access the Mongo Atlas
- In the backend config file, use the **Atlas** url for production
- Build and deploy the app to **Render.com**

Add a feature: toy image upload

- In <ToyEdit> add a feature to allow the user to upload a toy image using **Cloudinary**
- Build and deploy the app

{coding
academy

Part 7 – Getting real-time with Sockets

Let's chat about toys

Tasks

- In <ToyDetails> page, render a <ChatRoom> cmp.
- Each chat should be specific for the current toy (use the *toy._id* as the room topic).
- Chat cmp should render the chat-conversation, along with the user-name:

```
tal:hello
jonas: having fun with sockets?
yovel: dont forget google
```

- Add '*userName* is typing…' feature.
- Save the chat history in the toy document
- All connected users should get a notification when the admin changes something in the shop

Part 8 – PWA

Add PWA support for the project

- Update the manifest
- Bonus: offline support
  - Files are automatically cached by service worker
  - Cache data in localStorage

You made it 🏆