mister-toy !(yourToy)

Part 1 – Frontend First!

Name of the app: *mister-toy*

Note: this is going to be an end-to-end project so we will eventually have two folders inside the *mister-toy* folder: *frontend* and *backend*.

Setup your folders and your Git repository, in this project, the Git log should be meaningful and present the progress of the development work.

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',
    price: 123,
    labels: ['Doll', 'Battery Powered', 'Baby'],
    createdAt: 1631031801011,
    inStock: true,
}
```

Frontend

Build your frontend from scratch.

- Use the **CLI** inside your *mister-toy* folder and create a project named ***frontend***
- implement full CRUD, manage your 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> (Smart, Routable)
    a. This page renders full details about the toy
4. <ToyEdit> (Smart, Routable)
5. <ToyIndex> (Smart, Routable)
    a. <ToyList>
    b. <ToyPreview>
    c. <ToyFilter>
        i. By name
        ii. In stock
        iii. Toy label multiselect dropdown
        iv. Sort by: name / price / created

(git) commit your job: "Frontend now works"

Done? Build that backend

Create your own backend.

1. Provide an API for CRUD based on a json file.
2. Use the inClass project as reference
3. Use postman to test your API

Best strategy

1. In your project folder *mister-toy* create an empty folder: backend
2. npm init --yes
3. Set up a basic express application
4. Copy & Paste & Refactor yourself a backend toyService

5. 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
6. 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
7. DELETE toy
    - Create a request for DELETE */api/toy/:id* in Postman
    - Implement endpoint DELETE /api/toy/:id that deletes a toy
8. CREATE toy
    - Create a request for POST */api/toy* in Postman
    - Implement endpoint POST /api/toy that adds a new toy
9. UPDATE toy
    - Create a request for PUT */api/toy* in Postman
    - Implement endpoint PUT /api/toy that updates the toy
10. Refactor your 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

Part 2 – Awesome mister-toy

Use Community components and libraries.

Let's use some community components and libraries, use the ones demonstrated in class and add your own

Dashboard Page

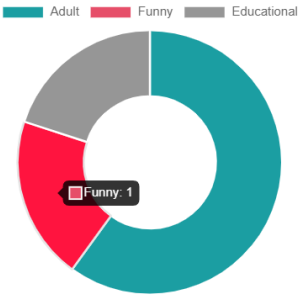
Add a dashboard page with charts:

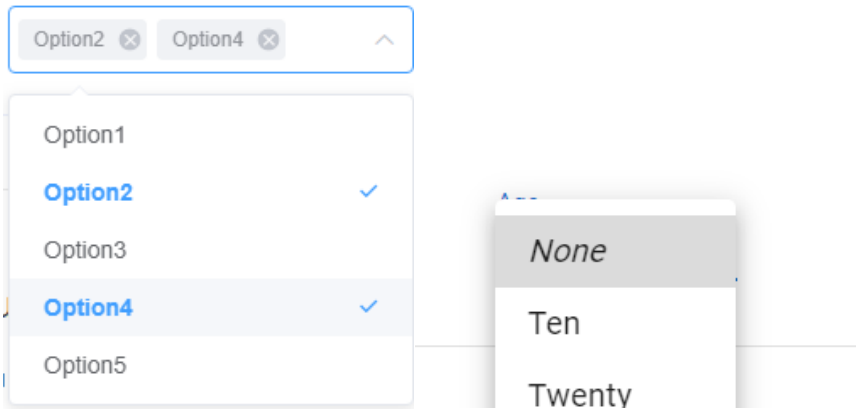
- Prices per label (Art, Baby, etc.)
- Inventory by label – Chart showing the percentage of toys that are in stock by labels
- Generate some random numbers and dates for a line chart

UI Components

Use various UI components

Example: in your toyFilter use a *select* component such as:



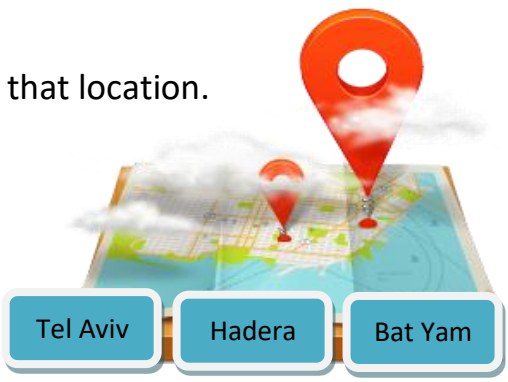


Form Validation

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 SCSS

- Use a full SCSS architecture
- Convert your CSS to SCSS
  - Use nesting
  - Use variables
  - Use mixins
  - Use functions
- Make it look amazing on desktop, tablet and mobile

Part 4 – Toys and Users with Mongo

Story

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

General

Use async-await, try-catch across your app

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

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

Check your backend from *postman*

Support authentication

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

Add msgs support

- Inside your 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 Ga'
            }
        }
    ]
}
```

- In ToyDetails
  - show 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

Part 5 – Reviews

Render, Cloudinary, MongoDB on Atlas with Aggregation

Lets add another option for the user to interact – he 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, will hold documents such as:

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

Lets use aggregation.

Aggregation of review, toy, and user

Use the reference code to aggregate reviews with users and toys and get the following output:

```
{
    "_id": "5bfa538166597429743c1ff0",
    "content": "Not your toy!",
    "toy": {
      "_id": "5b4f0b081043ae5f9cf3494c",
      "name": "Talking Doll",
      "price": 19779
    },
    "user": {
      "_id": "5b507e97f20dd52bb6e67a44",
      "nickname": "loris"
    }
}
```

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

Part 6 – Uploading and Going Live

- Upload a toy image using **Cloudinary**
- Edit the backend config file, use your **Atlas** url for production
- Build and publish Your App to **Render.com**

coding_
academy

Part 6 – Getting real-time with Sockets

Implement the following chat functionality

Tasks

- In ToyDetails page, render a chat-room 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.
- Bonus: save the chat history in the toy document
- Bonus: All connected users should get a notification when the admin changes something in the shop

Part 7 – PWA

Add PWA support for your project

Tasks

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

GOOD JOB!!!