# Workshop: Movie Magic – Part 2

Your team at work is tasked with creating a website that will attract movie enthusiasts. The site is named "Movie Magic". Its purpose is to create an attractive and interactive platform for movie fans that offers a unique and enriching experience.

## Main Task

If you have completed the previous task, good job! Now it's time to **upgrade** your app and **implement** a few new features. For instance, change the way you **store** data by using **MongoDB** and **Mongoose**, **create** and **attach** **cast** to each **movie**, make some **relations** between them, and **include** a few more **pages**.

## Installing Dependencies

As you already know, you should **install** a bunch of new things so you could be able to continue with this part of the workshop.

Here's the list:

1. [MongoDB Download Center](https://www.mongodb.com/download-center) – You can check the [Installation Instructions](https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/) as well
2. [MongoDB Node.JS Driver](https://www.npmjs.com/package/mongodb)
3. [Mongoose](https://www.npmjs.com/package/mongoose) – **Very useful** [**Mongoose Documentation**](https://mongoosejs.com/docs/guide.html)
4. [Robo 3T](https://robomongo.org/download)

## Database Connection with ExpressJS

Your **database.json** file inside the **config** **folder** will be **modified** because you **no longer** will **store** the data in a **JSON** file. So, make sure that inside it the **mongoose** **connection** via **MongoDB connection string** is **made** and **exported.**

The **index.js** file should **require** the exported mongoose connection (**database**) before the server starts.

## Model

If you follow the previous structure, you probably created ES6 class Model for each movie in this format:

* **id** – number
* **title** – string
* **genre** – string
* **director** – string
* **year** – number
* **imageURL** – string
* **rating** – number
* **description** – string

Now it's time to refactor this ES6 class to a **Mongoose** **Schema**, so each **Movie** has the following structure:

* title – **String**, **required**
* genre – **String**, **required**
* director – **String**, **required**
* year – **Number**, **required, max and min value**
* rating – **Number**, **required, max and min value**
* description – **String**, **required,** **max length validation**
* imageURL – **String**, **required**, **http/https validation**
* cast – a collection of **ObjectIds**, **ref** **Cast** **Model**

And create another model (**Cast**) in the following format:

* name – **String**, **required**
* age – **Number**, **required, max and min value**
* born – **String**, **required**
* name in movie – **String**, **required**
* cast image – **String**, **required**, **http/https validation**
* movie – **ObjectId**, **ref** **Movie** **Model**

Your model's folder should look like this:

Картина, която съдържа текст, Шрифт, бял, екранна снимка

Описанието е генерирано автоматично

## Database Persistence

**All** **pages** in the application should persist data to **MongoDB** & work with **MongoDB.**

## Additional Pages

You should implement **2** new routes:

* **/create/cast** – should render the form for creating a cast form
* **/attach/cast/:id** – should render the cast page about attaching new cast to a movie,

and **update the view** on the **/details/:id** route that renders the movie details.

Use the provided resources to create the additional templates (and **update the navigation** accordingly) using Handlebars. Identify the dynamic parts and use appropriate syntax for interpolating and rendering the application context.

### Add Cast Page View

### Картина, която съдържа текст, екранна снимка, дизайн Описанието е генерирано автоматичноAttach New Cast View

Картина, която съдържа текст, Човешко лице, екранна снимка, човек

Описанието е генерирано автоматично

### Updated Details Page View

Картина, която съдържа текст, екранна снимка, човек, софтуер

Описанието е генерирано автоматично

**Good Luck! 😊**