<https://learn.shayhowe.com/advanced-html-css/complex-selectors/> html selector

<https://www.freecodecamp.org/news/an-animated-guide-to-flexbox-d280cf6afc35/?fbclid=IwAR3PZnXEImImTh5uvIRe1Nzq8D6d6i_OP1UI4it7JZh1hwLI-yKmWyaIwMU> greed and flexbox

<https://css-tricks.com/snippets/css/complete-guide-grid/> greed

<https://flexboxfroggy.com/> flex game

<https://cssgridgarden.com/> greed game

<https://codepen.io/> for coding js css html

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements%20w2d2) w2d2 decleartion and statement

<https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Variables#declaring_a_variable> variable js

<https://nodejs.org/api/modules.html> for node.js module it is official website

<https://dmitripavlutin.com/value-vs-reference-javascript/> primitive/non primitive data type

<https://edabit.com/challenges/javascript> for coding place

<https://mongoosejs.com/docs/index.html> mongoosejs to study

<https://www.sitepoint.com/managing-dates-times-using-moment-js/> manage date time

**command**

**Git Commands**

**git clone https://github.com/sabihak89/plutonium.git :** downloads the specified repository on local system

**git status :** shows if you have modified any files on your local after cloning the repository

**git branch :** shows current branch

**git branch -a :** Lists all the branches of the repository

**git checkout -b practice :** creates a new branch called ‘practice’ from the current branch(main by default)

**git checkout account :** Switches to an existing branch called ‘account’  from the current branch

**git add description.txt :** Adds a modified (or new) file to the staging area. This means that this file will be included in the commit

**git commit -m “Added a text file for demo” :** Creates a commit with a relevant message

**git push :** pushes the commit to remote on the same repository where you are at

**git push <your-repository-link>:** pushes the commit to remote at the specified repository

**git push --set-upstream origin <branch\_name>** (Only applicable when pushing a new branch for the first time)

**Git config**

**Note:** Don’t forget to create an access token at Github. This token should be used as password in the command below.  
  
Create access token here: Account > Setting > Developer Setting > Personal Access Token > Generate Access Token > Save the token somewhere

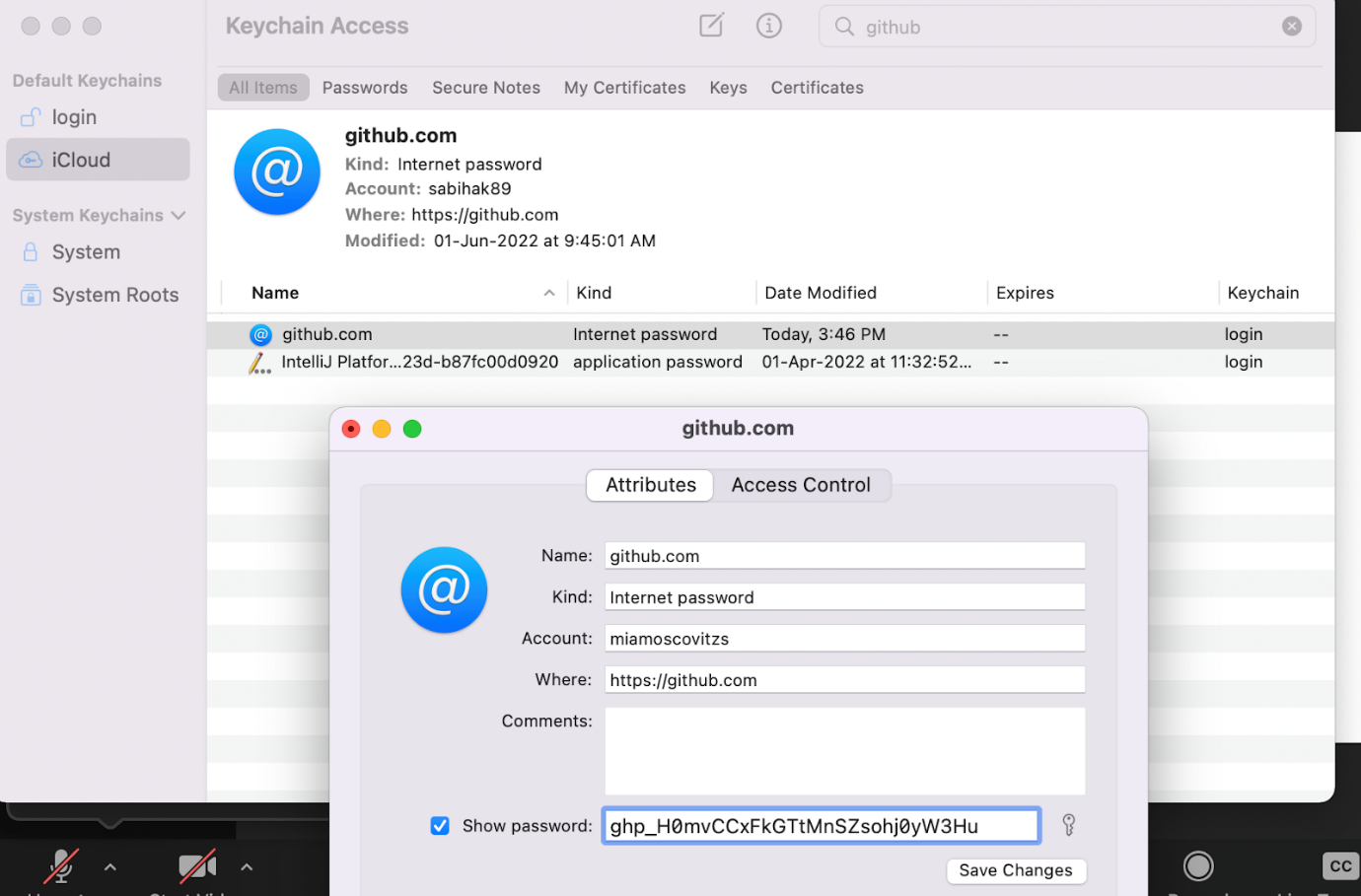
git config –list : shows the content of git config file

git config –global user.name sabiha : sets sabiha as user name in git config

git config –global user.password access\_token : sets access\_token as user password in git config

**Mac Users:**

You can additionally save the github credentials at Keychain Access (if you have this application)

****

# [Query Parameters in Express - Mastering JS](https://masteringjs.io/tutorials/express/query-parameters) Query Parameters in Express

**W5d2 assignment**

**Assignment :**

Create a books collection in your DB ( using bookModel with following fields)- bookName( mandatory field), price containing Indian and european price, year ( should be 2021 if no year is provided) , tags array, authorName, totalPages , stockAvailable ( true false)

create the following API’s (write logic in bookController and routes in routes):

* **createBook :** to create a new entry..use this api to create 11+ entries in your collection
* **bookList :** gives all the books- their bookName and authorName only
* **getBooksInYear:** takes year as input in post request and gives list of all books published that year
* **getParticularBooks:-** (this is a good one, make sincere effort to solve this) take any input and use it as a condition to fetch books that satisfy that condition
  + e.g if body had { name: “hi”} then you would fetch the books with this name
  + if body had { year: 2020} then you would fetch the books in this year
  + hence the condition will differ based on what you input in the request body
* **getXINRBooks-** request to return all books who have an Indian price tag of “100INR” or “200INR” or “500INR”
* **getRandomBooks -** returns books that are available in stock or have more than 500 pages

Find video explanation of the question here : <https://drive.google.com/file/d/1D9UOEl5rbGGDPjVLDGsf1L4hg9BI1ZH7/view?usp=sharing>

**W5d3 Assignment**

Index:

* **mixed** type in mongoose schema
* **Date and moment**
* **findOne vs find**
* **CRUD** in mongo
  + esp **updateMany** and **findOneAndUpdate**
  + **new** flag in update operations
  + **upsert** flag in update operations
* **isDeleted** key in mongoose schema
* [**Assignment**](https://docs.google.com/document/d/1ypIh7p62bI_L_xpWHeD4WkEGwcGZJBVC7eSIcj9cMnA/edit#heading=h.o5lbjx9k7822) **( we have not discussed ref and populate yet)**

# Content

**Mixed** data type



e.g. from schema

summary: mongoose.Schema.Types.Mixed,

summary key can have all these types of book summaries now:

    // summary : "this is a suspense novel"

    //  summary : ["ch1: Intro to backend", "ch2: intro to mongodb", "ch3: intro to nodejs:"]

    // summary : {

    //              chapter1: "How to get started with tech",

    //              chapter2: "lets start with basics"

    //             }

**Date type** and Date.now, new Date(), momentJs,

* **Date** maniupaltions( brief intro) + shared this resource on **moment** module :<https://www.sitepoint.com/managing-dates-times-using-moment-js/>

**find vs findOne**

* find gives all the documents that match the condition | findOne gives only the first document that matches
* find gives an array | findOne gives an object
* If no match is found -  find gives [] empty array which is a truthy value  || findOne gives null which is a falsey value …… user=findOne()  and then if(user)  :- will fail with find as it will be truthy even if no user is found

**updateMany and findOneAndUpdate**

//   let books = await BookModel.updateMany (  {isPublished: false } ,  {author : "PK"}   );  // first json is the query condition  || second condition is the required update or change

//   let books = await BookModel.findOneAndUpdate(  {isPublished: true } ,  {author : "Sabiha"}   );  // it updates only the first matching doc

//   let books = await BookModel.findOneAndUpdate(  {isPublished: true } ,  {author : "Sabiha 3"} , { new: true}  );  // third param : new: true - will give you the updated document

**upsert: true** - it finds and updates the document but if the doc is not found(i.e it does not exist) then it creates a new document

// let books = await BookModel.findOneAndUpdate(  {bookName : "Hi Pritesh2" } ,  {bookName : "Hi My New Book" , ISBN : "basd87g8h7a88b"} , { upsert: true}  );

**isDeleted flag**

// how to delete a document: never ever use remove.. always maintain a flag(a key in schema) "isDeleted: false" and whenever a doc is being deleted change this to "isDeleted: true”   (mark dirty)

# ASSIGNMENT  1:- ( dont use ref and populate)

You have to replicate the below data in your database. With this in mind, create a node application and APIs to do the following:

1. Write down the schemas for book and authors (keeping the data given below in mind). Also create the documents (corresponding to the data given below) in your database.

2. CRUD operations. Write API's to do the following:

* Write create APIs for both books and authors ---> If author\_id is not available then do not accept the entry(in neither the author collection nor the books collection)
* List out the books written by "Chetan Bhagat" ( this will need 2 DB queries one after another- first query will find the author\_id for "Chetan Bhagat”. Then next query will get the list of books with that author\_id )
* find the author of “Two states” and update the book price to 100;  Send back the author\_name and updated price in response.  ( This will also need 2  queries- 1st will be a findOneAndUpdate. The second will be a find query aith author\_id from previous query)
* Find the books which costs between 50-100(50,100 inclusive) and respond back with the author names of respective books..

bookModel.find( { price : { $gte: 50}  ,  price: {$lte: 100} } ) // WRONG

bookModel.find( { price : { $gte: 50, $lte: 100} } ).select({ author\_id :1})..run a map(or forEach) loop and get all the authorName corresponding to the authorId’s ( by querying authorModel)

**DATA:**

// \_id:ObjectId("8781263871293"), \_id will be automatically generated

**Authors:**

    {

        author\_id:1,

        author\_name:"Chetan Bhagat",

        age:25,

        address:"New delhi"

    } ,

    {

        author\_id:2,

        author\_name:"J.k Rowling",

        age:60,

        address:"Britain"

    } ,

    {

        author\_id:3,

        author\_name:"Ramanujan",

        age:100,

        address:"Tamilnadu"

    }

**Books:**

    {

        name:"Two states",

        author\_id:1,

        price:50,

        ratings:4.5,

    } ,

    {

        name:"Five Point Someone",

        author\_id:1,

        price:50,

        ratings:4.5,

    } ,

    {

        name:"The 3 Mistakes of My Life",

        author\_id:1,

        price:50,

        ratings:4.5,

    } ,

    {

        name:"One Arranged Murder",

        author\_id:1,

        price:50,

        ratings:4.5,

    } ,

    {

        name:"Harry Porter",

        author\_id:2,

        price:50,

        ratings:4.5,

    } ,

    {

        name:"Harry Porter",

        author\_id:2,

        price:50,

        ratings:4.5,

    }

# Assignment 2:-

Read up **ref and populate** after the class and write a 2 line summary on the same. We will pick this up tomorrow so no need to discuss this in your mentor sessions today.

<https://medium.com/@nicknauert/mongooses-model-populate-b844ae6d1ee7>

<https://stackoverflow.com/questions/38051977/what-does-populate-in-mongoose-mean>

**w5d5 Assignment**

**TOPIC:** Mongoose Populate and Reference

* For this assignment the session branch is **session/populate-reference**
* For the solution you have to create a new branch in your own repo- **assignment/populate-reference**
* Use your own Atlas database links to avoid issues. Use these collection names if you have already used the collections in previous assignments - **newBook**, **newAuthor**, **newPublisher**.

* A newAuthor document should look like this (no author\_id anymore - you can delete this from the schema)

  {

\_id: ObjectId("61951bfa4d9fe0d34da86829"),

authorName:"Chetan Bhagat",

age:50,

address:"New Delhi",

rating: 2

}

* A newPublisher document looks like this.

{

\_id: ObjectId("61951bfa4d9fe0d34da86344"),

name: “Penguin”,

headQuarter: “New Delhi”,

}

* A newBook document should look like this. The author property is a reference to newAuthor collection.

{

\_id: ObjectId("61951bfa4d9fe0d34da86344"),

name:"Two states",

author:"61951bfa4d9fe0d34da86829",

price:50,

ratings:4.5,

publisher: "61951bfa4d9fe0d34da84523"

}

1. Write a POST api that creates an author from the details in request body

2. Write a POST api that creates a publisher from the details in the request body

3. Write a POST api that creates a book from the details in the request body. The api takes both the author and publisher from the request body.

In this api, you have to write a logic that validates the following :

1. The authorId is present in the request body. If absent send an error message that this detail is required
2. If present, make sure the authorId is a valid ObjectId in the author collection. If not then send an error message that the author is not present.
3. The publisherId is present in the request body. If absent send an error message that this detail is required
4. If present, make sure the publisherId is a valid ObjectId in the publisher collection. If not then send an error message that the publisher is not present.

4. Write a GET api that fetches all the books along with their author details (you have to populate for this) as well the publisher details (you have to populate for this)

Edit: New problem (5)

5. Create at least 4 publishers (Penguin, Bloomsbury, Saraswati House, HarperCollins). Create at least 6 authors with ratings 2, 3, 3.5, 4, 4.5 and 5. Create around 10 books with these publishers and authors.

Create a new **PUT** api **/books** and perform the following two operations

 a) Add a new boolean attribute in the book schema called isHardCover with a default false value. For the books published by 'Penguin' and 'HarperCollins', update this key to true.

 b) For the books written by authors having a rating greater than 3.5, update the books price by 10 (For eg if old price for such a book is 50, new will be 60)

Binary search

Linear search

Bubble sort

Selection short

Insertion short padh lo thoda dsa sahi hoga

+ Please note that you have to also write the logic for authorisation now so that a logged in user can modify or fetch ONLY their own data.

+ You have to implement authorisation for fetch user details, update user and delete user apis

+ Run this code and ensure the authorisation works fine for all the apis before following the next requirement

+ You now have to move this similar code in all the three apis in a suitable middleware