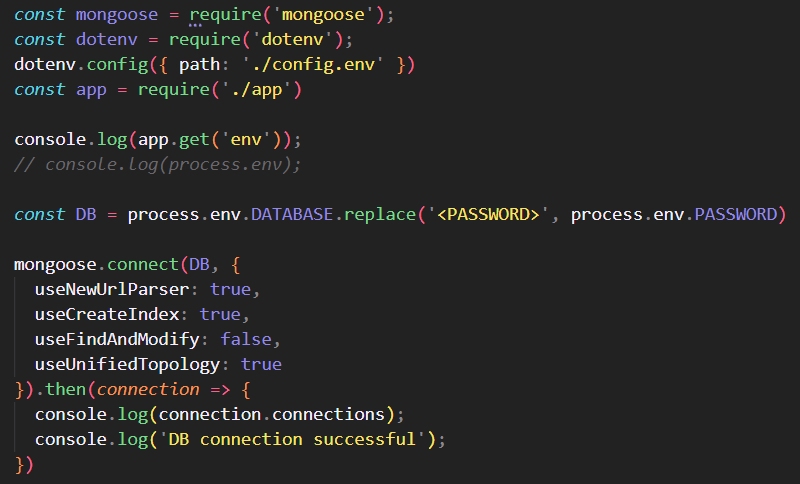
Connect to the DB

1.Connect to the Db with express app

Install mongoose 5

Import mongo

Specify some option to deal with deprecation warning



2. Introduction to mongoose

Mongoose is an object data modeling – write javascript code to interact with database library for mongodb and nodejs , a higher level of abstraction

Like relationship of express and node, express is higher level of abstraction though regular node

We can use regular mongodb to access to db, but for more convinient, we will use mongoose, it allow faster and simplier development of application

Feature of mongoose: schemas to model data and relationships, easy data validation, simple query API, middleware,..

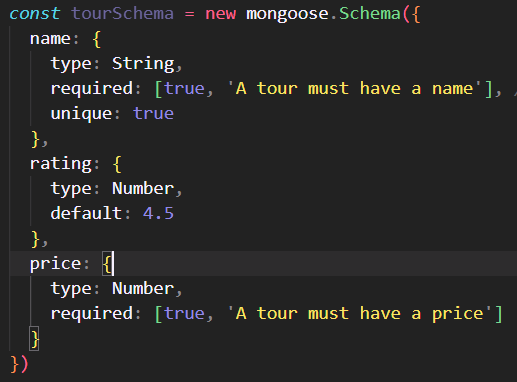
Use mongoose schema to model data, by describing the structure of the data, default value, validation

From schemas, we form model, like a wrapper for the schema, providing an interface to the the db for CRUD operations

3. Create simple tour model

create field and schema type option,

Schema type option with required and unique property

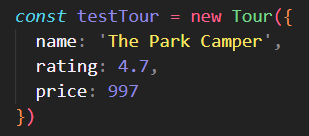


Create a model based on the schemas instance, this model call a object model(like a table in SQL)

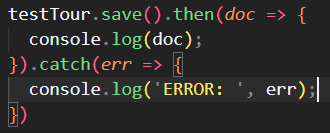


4. Creating document and testing model – adding item to the model

Create a object document, like a row in SQL



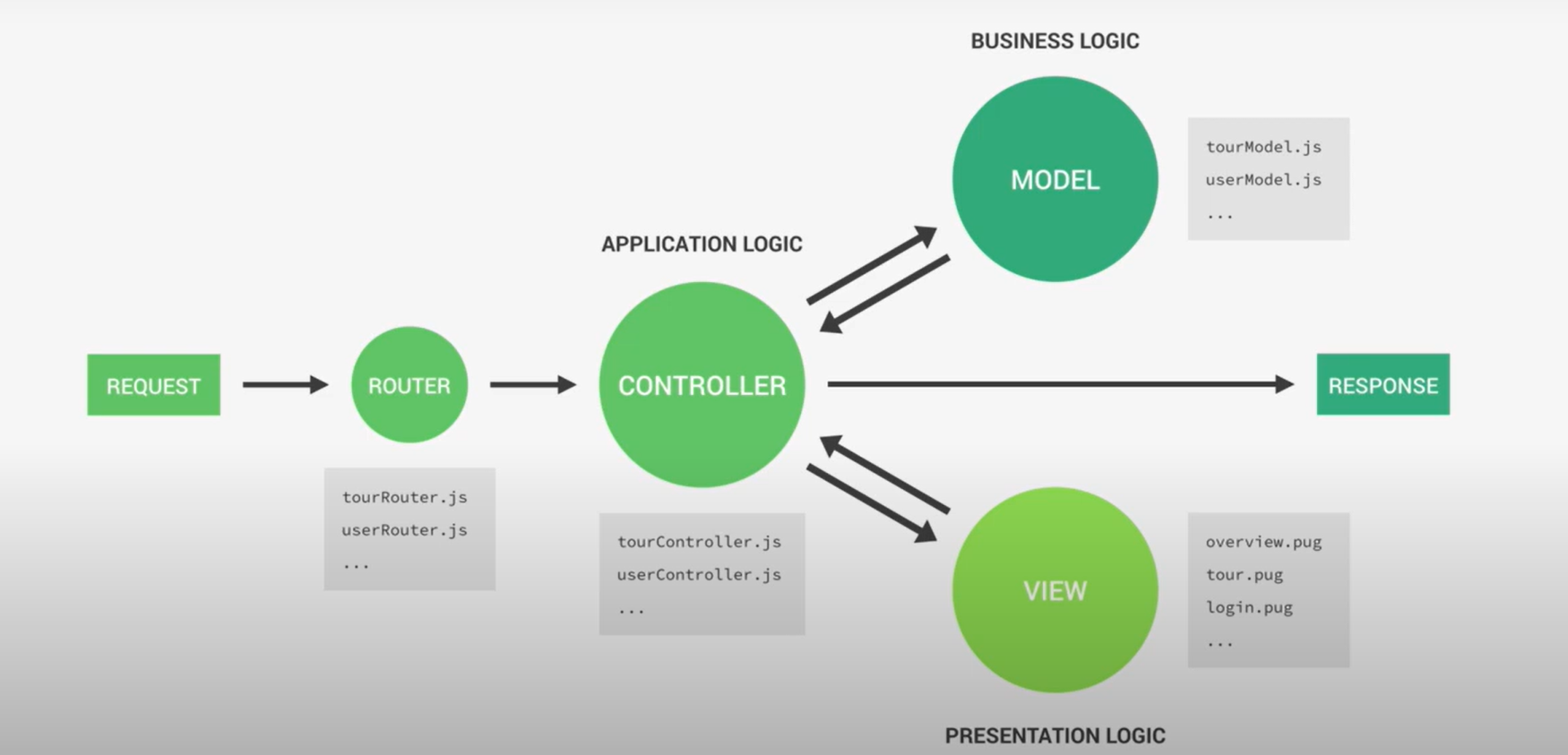
Save document to the model



That array is automatically throw an error if the property is not defined or not declared



5. Backend architecture: MVC, type of logic and more



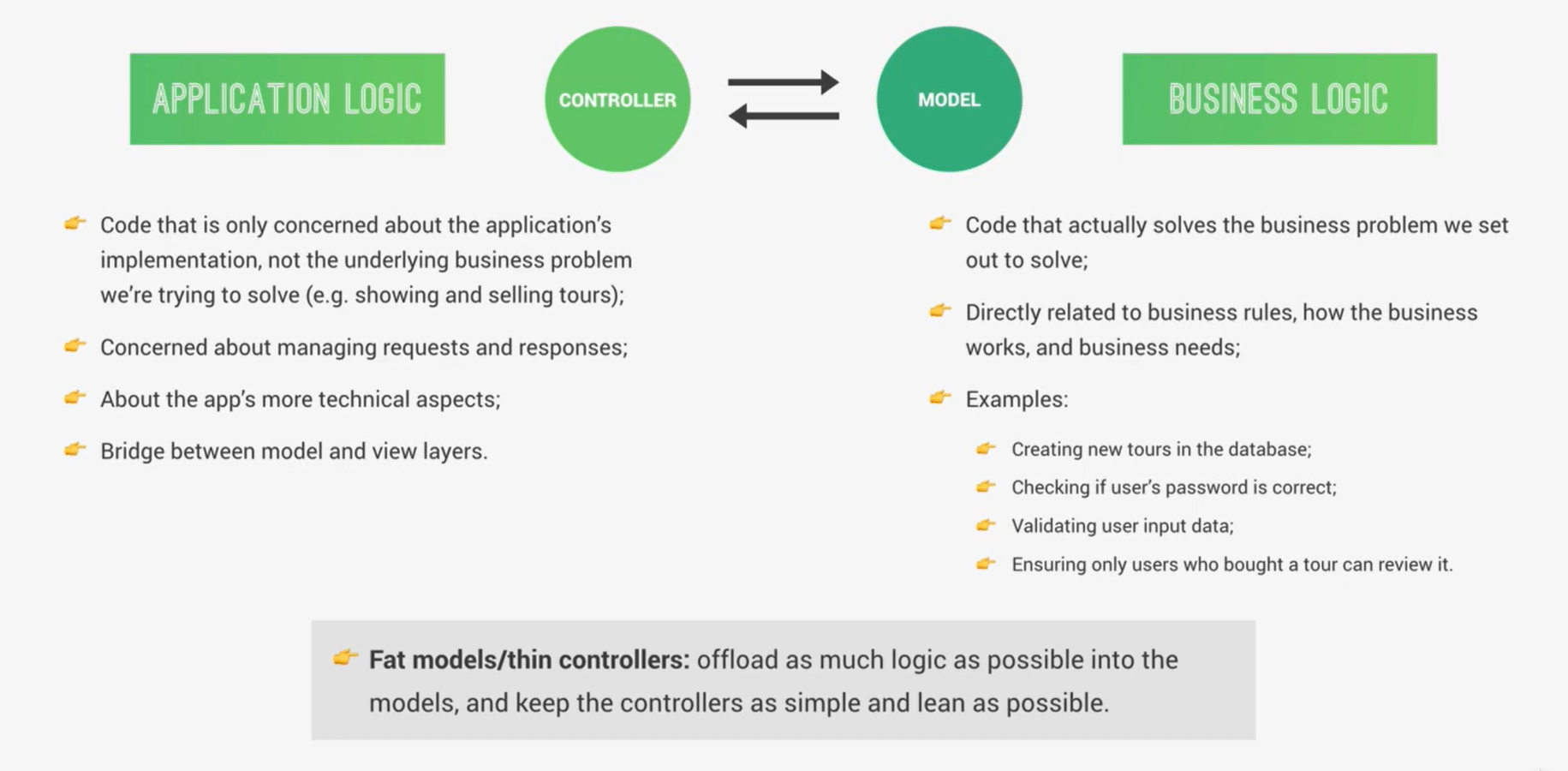
When a request from client, request will point to exact router and then go though controller logic

From controller, will take togic from model, for example, retrieve document from db or create a document

After getting data from the model, controller ready to send back a response to the client

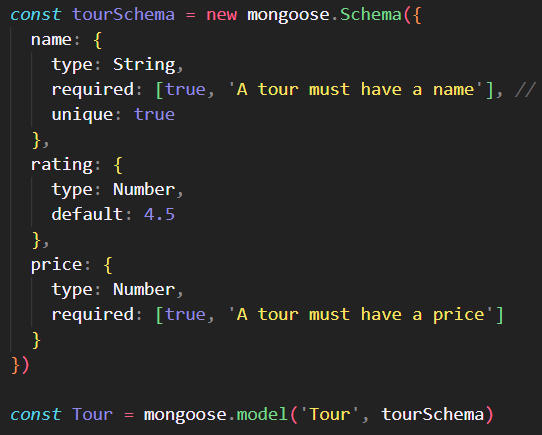
If want render a website, When have a data, controller will select properly template and inject data into it, return for controller

Controller send a website with interface for the user



6. Refactoring for MVC

Split model: related to DB

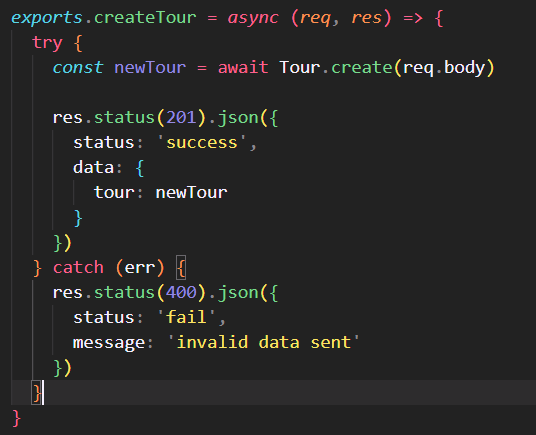


Refactor again controller

7. Another way of Inserting item to the db

Get rid of checkBody function because with param route, mongoose will take care it

Refactor create tour: Using create method instead of create new instance of Tour



Using try catch to take the error, using status to control code

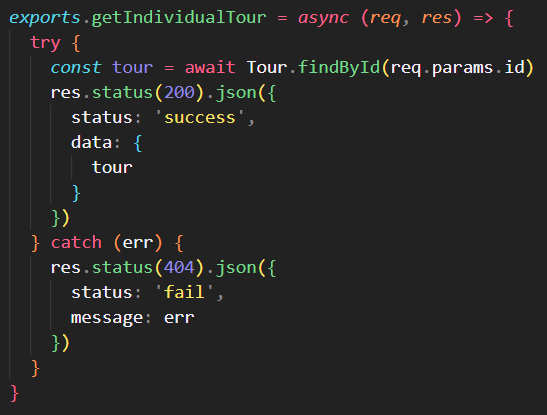
8 Reading document with mongoose

Using, find method to find all, async await method

Defined all of nesscessary prooperty



Get single tour

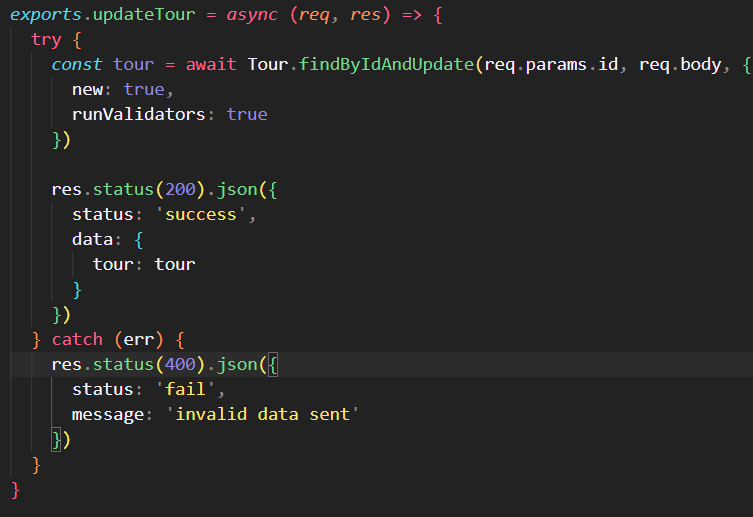


9. Updating document

Using find ByIdAndUpdate, taking 2 params. First: pass id that you want to update

Second: body sent from the request

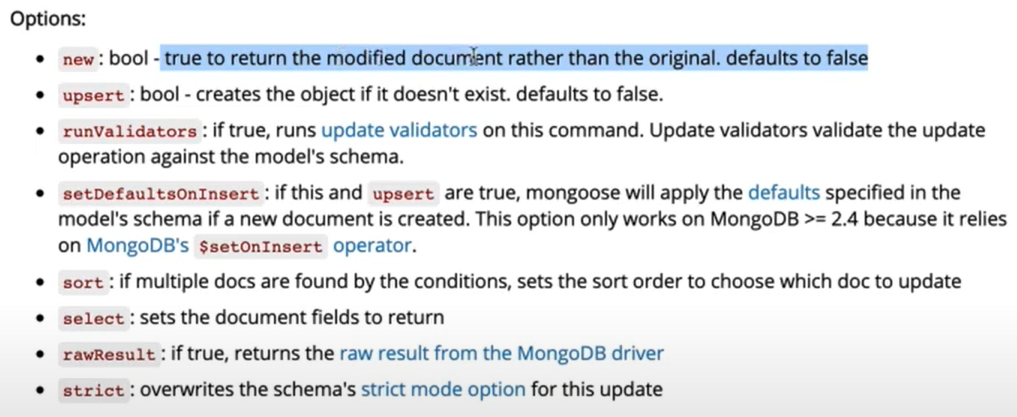
Third: object, specify some nesscessary condition to update



There are some queries here:



Here is some options for the third param

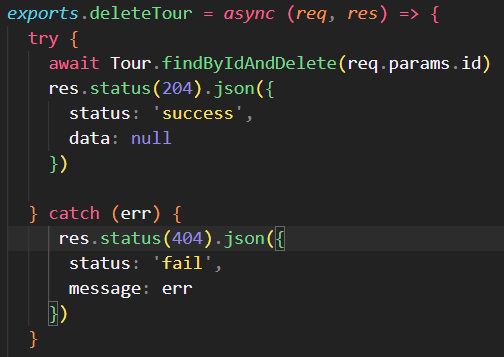


True if it is modified document,

And runvalidators, if true, run update validators on this command.

And then set data return to the tour that we just find out

10. Deleting documents



Delete by id, remenber don’t have return keyword,

Status 204: no content

11. Modelling the tour

Add more property to the model

Other than 3 available property

Add durations,

Add maxGroupSize,

Add difficulty,

Add ratingsAverage,

Add ratingsQuantity,

Add priceDiscount,

Add Summary,

Add description,

Add imageCover,

Add images,

Add createdAt,

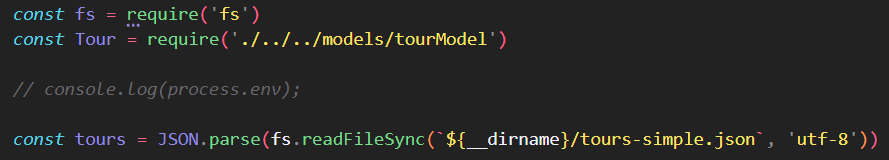
Add startDates: different date for the same tour

12. Importing development data

Create a file to load Data from dummy data – json type to the DB

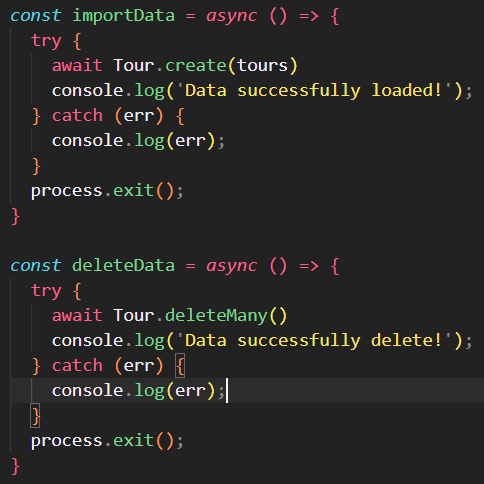
Using fs library to read file, parse string to json

Import tour model contain schema, for that, we easy to write and import data to the DB



Some attention for import the tour-simple.json, if we just specify ./tour-simple, it just find the file in the home folder, instead, we need to use dirname

Create deleteData and importData to the DB, it look the same with each other



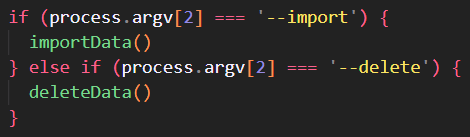
Call the build-in method in the Tour schema, it already defined

Next, we use process.argv to view the current thread of the program





To import, we specify import option. Depend on option we specify, we will call to the compatible method – delete or import



Based on the array process return



But still have error, when get all data, result is 0 because the process is still running

Use process.exit() for both delete and import data

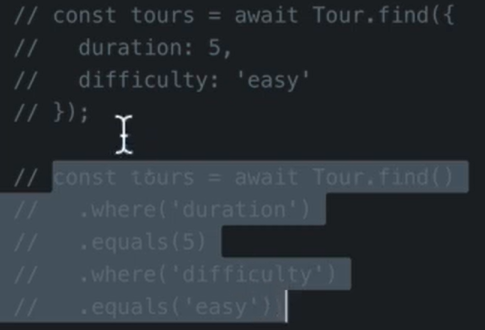
Because we always need to hit the exit function, so we bring it out the try and catch block

13. Making the API better

Filtering data in params

Filter by object:

Filter by optional chaining property



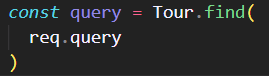
Create a shallow copy of the params by destructuring



Create exclude array field



Find item by query



Execute the query



14. Making the API better advanced filtering

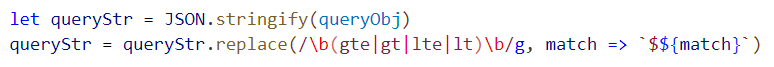
Previous filtering: 1 key is assign to 1 value

Advanced filtering: 1 key is assign more than 1 value, in case we find in the query param contain some complex keywords

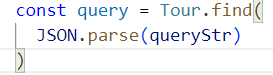
\b make sure the word boundary, all the matches is standalone word

\g flag to make the keyword global, replace all the matching word occurences in the string

Every times found the same string, replace it with the dollar sign



1 more important step is parse that string to JSON type

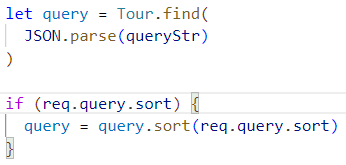


15. Making the API better sorting

We will sort in specific property

If one of the 4 property: page, sort, limit, fields appear in the param request it will be excluded

Check if having sort prop, we will sort it in order from lower to higher

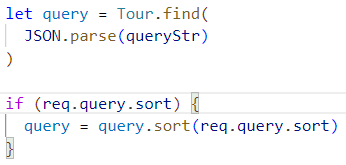
\

Or we can sort in order from higher to lower by adding minus to the price, in case we sort = price

The case we want sort many param at the same request,

Check if have query.sort , plit query by comma, and joinning it by using space

And pass it to sort function



Else we will sort in newest created time

16. Making the API better limiting fields

Allow client get back which field they want

If we want exclude some field right on the schemas, for example, we want hide createAt field, we just simply add select property to flase into Schemas

17. Making the API better pagination

We find if appear the property “page” and “limit” in the param of request



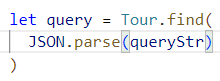
From that 2 property, we will take out exactly what element will appear in the specific page



For example, page = 2, limit = 3, we will skip 3 first element

After that, we use built-in function of the query like this:

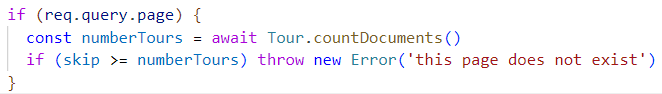
For rehesion, we have query like this:



Just function skip and limit, take in param is number



Add 1 error situation when user reach not found pagination. The idea is if the skip item is greater than number of item in the db, we will throw new error instead of run into the catch block with error 404



18. Making the api better aliasing

Instead of query by passing a lot of parameter to the query, we just use one single route for taking that



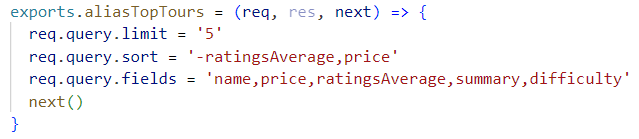
This API want to take out the item have best rating and price is cheapest



Alternative API

The idea is that we will run a middleware like aliasTopTour before reaching getAllTours api

We always write middleware in the tourController



And use it as middleware in the tour route



19. Refactoring API feature

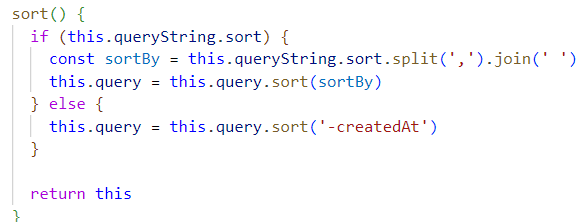
Always remember that the mongoose query and the queryString we get from an express

Create a class in OOP design, for the aim of reusable

Create filter function for the aim of filtering, it take all of the item in the document, second argument is the query string that we take out, and call to method filter. After filter the query string, we take out all of the tour

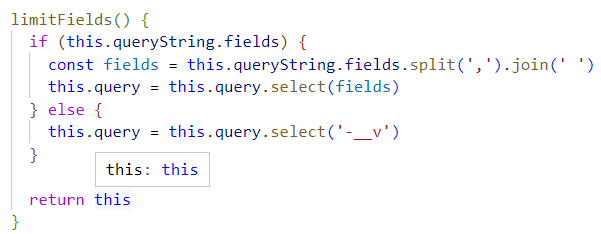


Implement sort method in the class API features

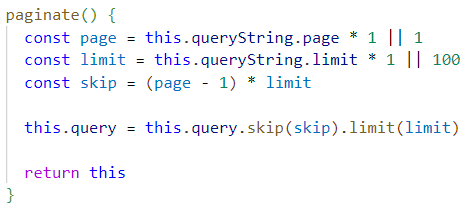


Remember to return this, because for the aim of manipulating with object, we need to return entire object after we filter or sort. If not return, we don’t access to any value

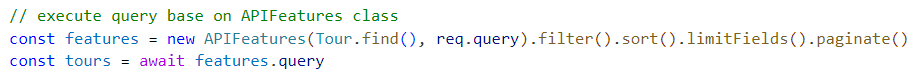
Implement limitField method,



Implement paginate method but don’t need implement throw an error at this point



And chainning all of that function



20. Aggregation pipeline

1. What is aggregation pipeline ?

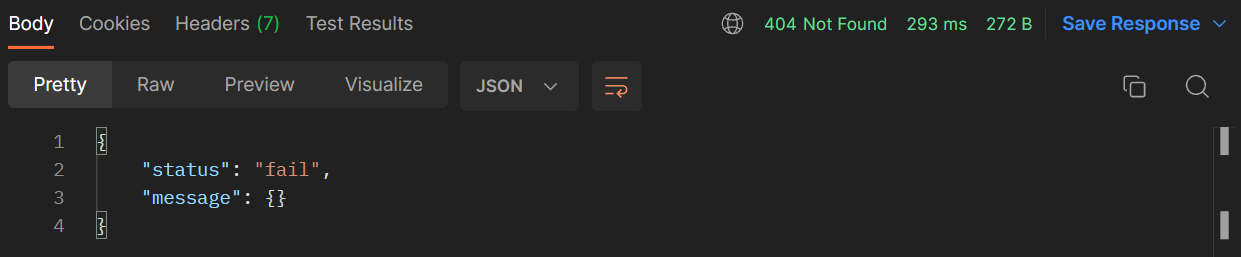
* A powerful feature provided by MongoDB DB that allows to process and transform data in flexible and efficient manner
* Consist of a series of stages that are applied to documents in a MongoDB collection, producing the desired output
* More flexible than normal query
* Each stage in the aggregation pipeline performs a specific operation on the input documents and passes the result to the next stage => perform complex data manipulations: filtering, grouping, sorting,…

2. Application

* like normal query but can manipulate with data in different stage

3. Bug seeker

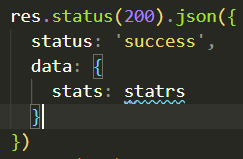
Error in tour-stat endpoint



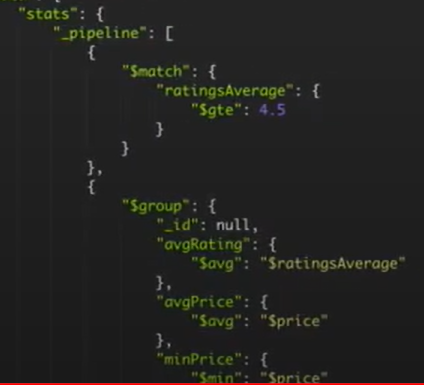
Cause: initialize stats variable but not use



Not assign right data



If lack of await keyword, we returned result will be normal query and we specify the sort state



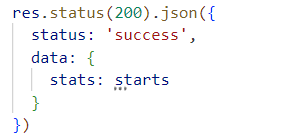
4. Usage

* Go to website for the documentation of the aggregation pipeline stages, there are a lot of stages that we can use for reference, we will use match and group aggregation.
* Aggregation in this route include two stages: match and group.
* With after each stage is matching, its result will come to group stage.

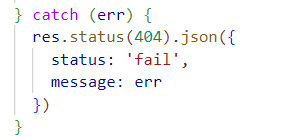


Use variety mongodb operator: [gte, toUpper, sum, avg, min, max]

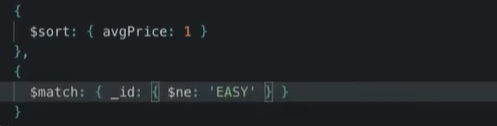
Specify code status



And catch the error



We also can repeat stages by match again



5. Working with unwinding and projecting pipeline

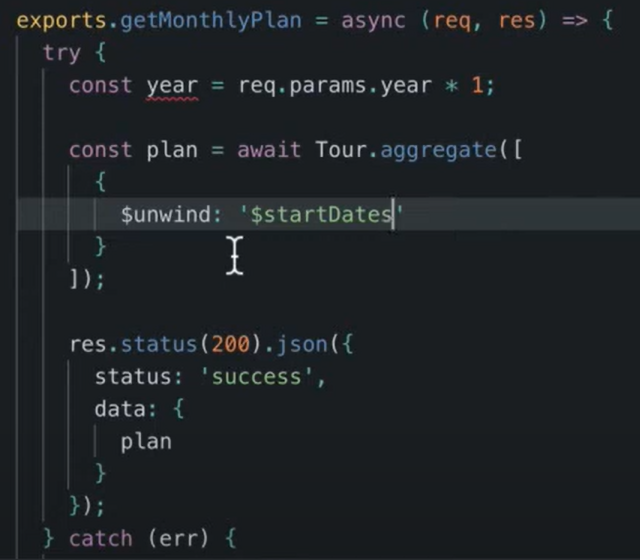
Split array and form an array of json form

In this api we will search tour in single year, in each year how many tour it have and the start month using 7 different stage, the result will look like this:

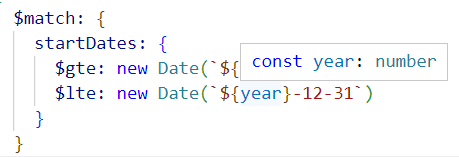


We combine all duplicate tour in to 1 single date and function getMonthlyPlan like this:

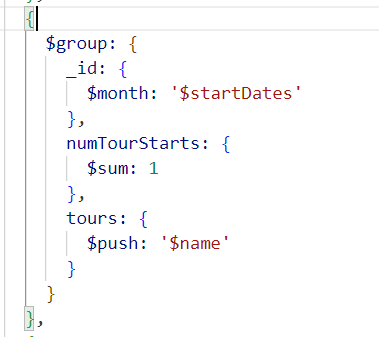
**Stage unwind**: deconstruct an array field from input document and then output document for first document of the array



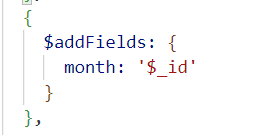
**Stage match**: Select document for the year that we pass in, we use match stage, startDate will start from first day of the year and last dat of the current year



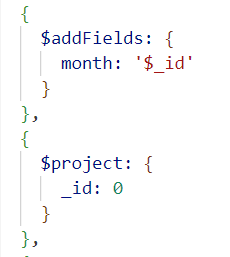
**Stage group**: group by month of the startDate field, sum operator will increase by 1 for each document is searched, push operator will push the “name” is found to an array have property key: tours



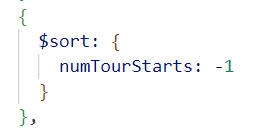
**addField stage**: to add 1 more field return from API



**Stage project:** to make visible id only or the returned group, set to rezo will specify not show \_id property



**Sort stage:** use arbichary prop to sort in order or inverse order, -1 will sort from high to low



**Limit stage:** will limit the item that we take out from API:



|  |  |  |
| --- | --- | --- |
|  | Aggregation (n) | Sự tập hợp, nhóm lại |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |