International Journal of Computer Science Trends and Technology (IJCST) – Volume 9 Issue 5, Sep-Oct 2021

RESEARCH ARTICLE OPEN ACCESS

CAMPLAND - A Registration and validation portal for Campsites

Koganti Aditya [1], Vinod Rongala [2]
CSE, Vellore Institute of Technology, Vellore, Tamil Nadu - India

ABSTRACT
When we have to book a place for hosting a picnic/ gathering we don't find all the relevant info in one place. After finding a venue for the event, it has to be verified by actually visiting the place or inquiring through third parties. The user has to rely on the information provided by these unknown agents or any other source providers. As we do not proper source, the person has to validate the accuracy of inform by visiting personally. There is also no sophisticated online payment system for booking. An online booking system can solve the above issues and provide a platform where users can view the places that match their requirements, in this paper, was presented an idea of implementing a node js web application using secure authentications and databases used.

Keywords — NodeJS, databases, authentication

I. INTRODUCTION

Many people are showing great enthusiasm towards outdoor activities such as hiking, campaigning, and other adventurous rides they crave for. And there are so many places people want to visit and some people want to share as they meet great places in wild or campaign in regions where no one has gone before and so there is no way no do so. Further, some people have no clue about the certain region they are thinking of visiting as they have no clue to know due to differential geography and language communications and also for the people who want to visit new places to live up to their curiosity.

For these kinds of situations, many tried to find various ways Like hiring a professional tourist for guiding through these kinds of unknown places and other assistances and here comes the problems since the foreigner who hired the guide doesn't know the native workings may end up getting cheated or suffer other kinds of losses, This not only limited to a single person such as guides but also organizations such which tells it provides support for tourists may fall prey to these firms thus a more realistic solution is needed for this problem.

In this paper, we provide a solution to overcome the above problems through a web application and introduce you to the latest technologies that provide better efficiency compared to the prior.

II. PROPOSED WORK

The following web-based application will be drawn upon MEN stack representing M for Mongo dB, E for Express.js, and N for node based on Node.js – A JavaScript runtime environment used for backend application. This was implemented for various reasons such as listed below:

A.A.1 MVC SUPPORT

One of the best reasons for proposing node.js is that it supports the MVC pattern. This is useful for working on each area separately, such as, as the term implies models, views, and control, we can work independently without too much of each component influencing the other. The model module works as the blueprint of database structure and the view component represents the front-end part loaded based on the front-end frameworks from a database or templating engines while the controller part as the name implies it manages the control over all other components where it contains all the server-side logic such as maintaining the database and requests from a user from client-side and all business logic

A.A.2 Solitary Language

Having learned this language has a unique benefit in that it is its whole language is built upon a single language "*JavaScript*". As we generally know how common JavaScript is used and how popular it is, hence having the same language for the backend is a huge benefit such as we can avoid the tiring process of learning a new language and since we work on the frontend on JavaScript and also in other frameworks we can easily inject and switch with the codes.

B. Authentication & Authorization:

As much as we know we have heard technologies such as PHP which use SQL databases for building credential management these generally store data (such as username/email-Id and password) as to how it is passed like any other data it processed into the application database, these considered as high vulnerability which may easily be lead to the data leak, Though there are many ways found to prevent to this hack, they are quite complicated to implement and not flexible enough, and this is where the passport.js comes.

Passport.js is a well-used module among the various other libraries, this plays the role of middleware for authenticating the user and is not just limited to that, can authorize specified users or authorization requirement tasks on specific operations. Since this was written in JavaScript it provides ease of use and is smoothly injected with along with other libraries or frameworks used. Even with all these functionalities these were not the major reason for its use here as there is another huge functionality, was that its ability to encode the provided data (password consider here) to generate into the values of salt and hash.

C. DATABASE(NoSQL)

Though we know how popular the SQL was used, it was inferior to the latest technology based on NoSQL in certain cases. It is MongoDB. Although it may seem too overrated to say it supplanted the traditional database it is spreading across multiple technologies as for its wide known popularity for the great difference in speeds comparatively with SQL

There are certain factors for the cause of implementing this data model for the API over most-used RDBMS for instance for its most popular reason of schema-less form. This alone places a huge difference with SQL which use the RDBMS to manage its database to logic for applying various relations and table. MongoDB is free from such complexity, it completely removed from that concept but its workings are more like a partial schema this is was because the schema applies when the data was read or retrieved.

Further, there are also many others such as its non-compulsory requirement to fill all fields of the model, as such we can leave some blank which shows its flexible nature(like size, data type, etc.,) and also contrasts to the SQL database it does not rely on a relational function to access or modify data, it calls a specified data element through a unique id generated through mongo stored alongside with its other elements in the format of JSON, In this project, we proposed to use mongoose a node.js associated framework built on top of MongoDB as its origin which was designed for ease of use with NodeJS. This provides various advantages such as we can use a predefined model on the collections and clean organization and use in project structure along with all the other functionalities mongo dB provides.

Nodejs has terms called frameworks (such as mongoose and passport) which will be used for building the applications this Is varyingly different from the typical known libraries. That is it is much faster in speeds comparatively with these libraries although they are similar to in the usage with an import code used in applications we can say the difference is "we are calling the function and function calling you", this implies libraries are the whole bunch of code with predefined functions which we will be using for usages whereas frameworks are codes which will be left to write by user mostly and control is steered by the developer rather than already defined functionality in the libraries

CRUD Operations:

As we generally know currently only two requests are capable of working on the web ("GET and POST"). This will severely limit an application's capability; hence we overcame this using a module called "method-override". This does not add new verbs such as put and delete but makes a query that the client asks to perform, which is popularly heard as RESTful routing which is wrapped along with express module and acts as a middleware-like functionality.

Route configuration
We will be using express.js a popular lightweight framework this has been chosen not just because it is immensely rated across people but because there is more community working with this framework as such it will be advantageous it ways such finding solutions and tutorials to all problems and as such will be its ever-increasing development on this framework from the creators.

III. IMPLEMENTATION

Here we will be creating a database on 4 types of collections named campground, comment, to-dos, and users we will be assigning each page dedicated to each collection and some are based on specified data in a collection, there are some referenced objects in another which on whole basis co-dependent on each other.

The plan is as follows:

- On the first page, the user will be welcomed with a reference link to enter the home page
- Every page has a common nav bar for identifying the page he is in and the status of the user for also to log in or register or logout dependent on the status
- At first, a non-user will be having the capabilities to collectively see the campgrounds on the home page and can enter the specified camp on inquiring for more details.
- Here he will be able to see expanded details about the camp, user uploaded and cost of renting the place along with the comments of other users about the camp.
- For more privileges client will have to sign in/up for more capabilities such as the applications that allow for the user to have posted a new campground and also the right to comment upon an existing camp. This implies the permission publish about a campground user knows upon and specify information and price costs it would take and comment for the experience he already visited or something else.
- Ability to also edit or remove the existing campground posted by the user

A. *Routes and configurations:*

Express is used for routing, routes are for listening to what the client-side was requested such as sending the data and also getting the data from the user posted

Here we have deployed the database on the cloud and also could be worked on a local server and I will be explaining you on the local server basis. Routes are addressed to send to receive hence require a dedicated reference to differentiate between each page as we are working on a local basis here the home page will be available to the client through "https:localhost:3000/"(where 3000 is port-number for data-based linkage and app deployment) which implies a get request '/'.

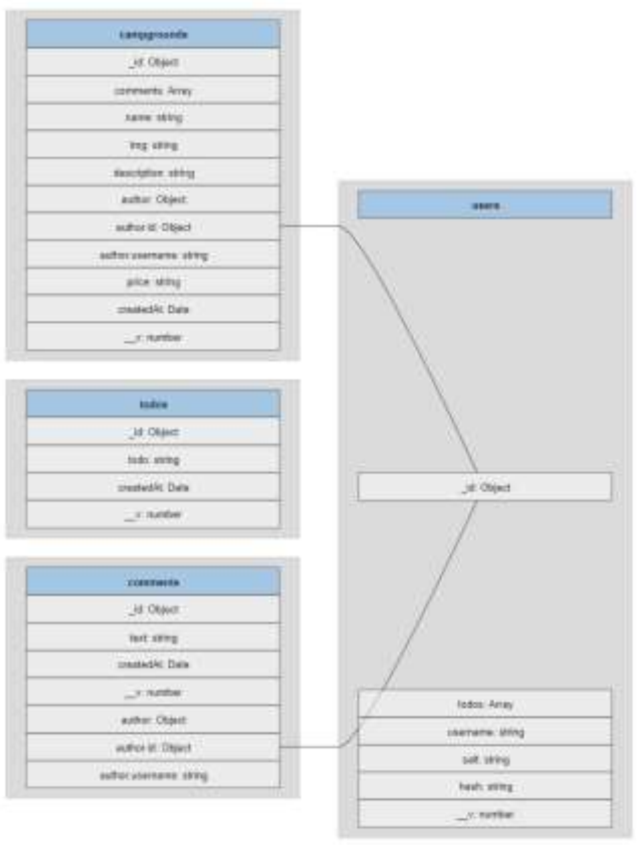


Fig. 0 Representation of how the database is organized

As stated MongoDB is a schema-less database but as we can see from figure 1 it provides a schema on several collections in the database, what type of format the values are taken, and how the data values are interrelated to other collections values but this is not a schema as to how it appears similar to the SQL, we are not mandated to assign all the inputs or restricted to only given values, it is a way to organized way to structure the database and application as we don't constraints as in SQL.

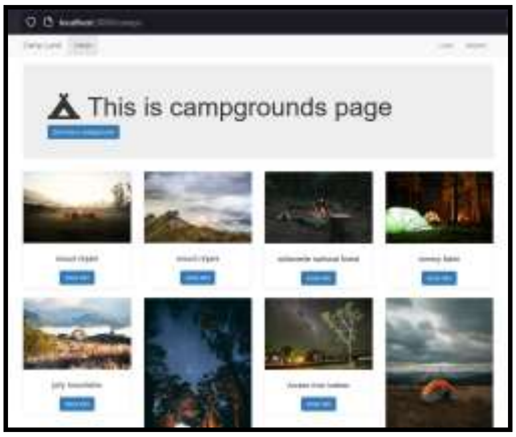
And thus, routes are assigned as follows:

Routes have been classified into the following 3 major categories:

❖ **Campgrounds:**

➢ '/camps' or 'https://localhost:3000/camps' are divided into two categories:

▪ Get request where it will retrieve all campgrounds with title and images loaded as card form



▪ Post request will be enabled only on user login where the user gains the ability to post the form details and redirecting the user to the home page with updated details.

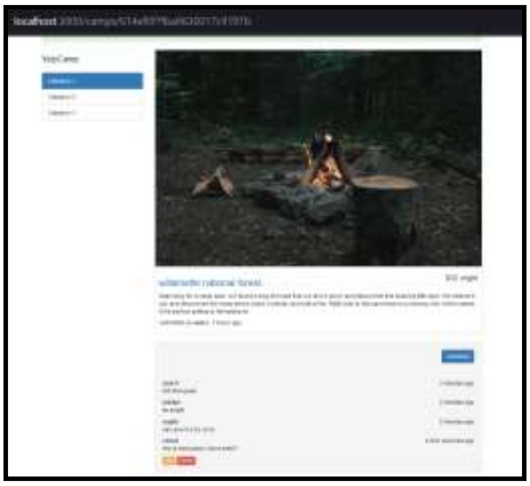
➢ '/camps/new' or 'https://localhost:3000/camps/new

▪ 'Get' has only this one request has only access to signed-in users to display the form for submitting the details that will be later sent through the post request stated prior.

➢ '/camps/:id' or 'https://localhost:3000/camps/:id' while this was divided into 4 broad categories

➢

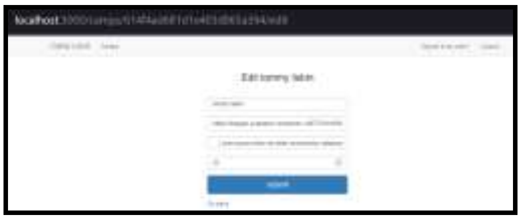
▪ 'Get' – draws over the details of a specific campground on the selection using unique id generated with elaborated details along with the comments



- ‘Put’ – this is from one of the CRUD operations (create, read, update, and delete) where a selective campground is updated
- ‘Delete’ – similarly from method-override functionality deletes the item based on ID.

➢ ‘/camps/:id’ or ‘https://localhost:3000/camps/id’

- As for the above-stated update (PUT) functionality to work, a necessary “form” page similar to the new campground form is requested.



❖ **Comment section:**

Similarly, as stated above, this above stated for the campground is applied to the comment section. It was divided according to the components based on RESTful routing is present in the below table:

URLs are predefined with “https://localhost:3000/camps/:id/comments” then added with the following table:

TABLE I

Request	URL	Action
Get	“/new” (Figure 2.1)	Form submission for a new comment
Post	“/”	Posting logic of the comment submitted through the form
GET	“/:id2c/edit” (Figure 2.2)	Form for updating/editing previous comment
PUT	“/:id2c”	Update logic for edited comment
DELETE	“/:id2c”	Deleting the comment

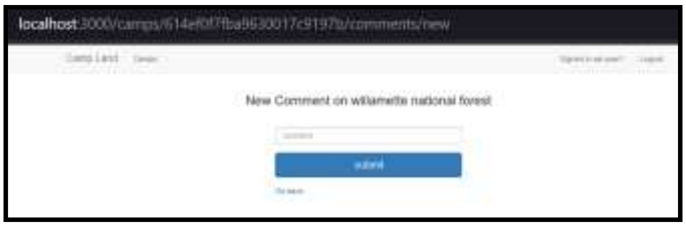

localhost:3000/camps/614ef0f7fba9630017c9197b/comments/new

Fig. 2.1

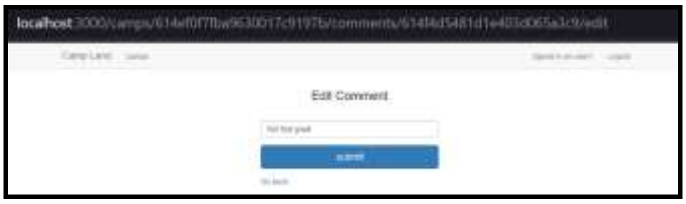


Fig. 2.2

- **User Section** (&Index)**:**

These routes are mostly compromised of authorization for a user. The routes were defined in Table II as shown consists of a total of 5 for login and register excluding the landing page, the routes are priorly added to the link ‘https://localhost:3000’ alike to the campgrounds and comments and exported through the router module and imported and combined in the main app.

TABLE II

Request	URL	Action
GET	‘/’	Shows welcome page
Get	“/login”	Form filling for signing in
Post	“/login”	Posting logic of the login submitted through the form
GET	“/register”	Form for signing up for a new user
POST	“/register”	Post logic for registering and adding the user to the database
GET	“/logout”	Signing out the user

This plays a significant role in the application in the area of authorization and authentication.

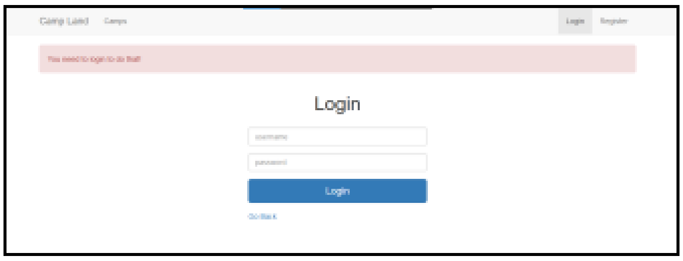


FIGURE 2.1

As for the provided routes above for the campground and comment section, not all routes are accessible but were assigned with certain constraints given below:

- As introduced at the start of the section for non-users can view campgrounds and comments and for posting, users must be signed up or logged in, so is for the commentating upon a campground
- This was added as a middleware functionality imported from another JavaScript file for common functionality for all (such as checking if the user was logged in for posting) As shown in figure 2.1

- It is also secured that no user who does not have ownership over the comments or camps he posted will be not authorized to do so, this is the authorization functionality that was implemented with help of passport.js (i.e., application requests for the user for username crosschecks with the user logged in for granting rights upon which the user has access to modify or delete)

Based on the order of Table II representation of application:

Fig. 3.2

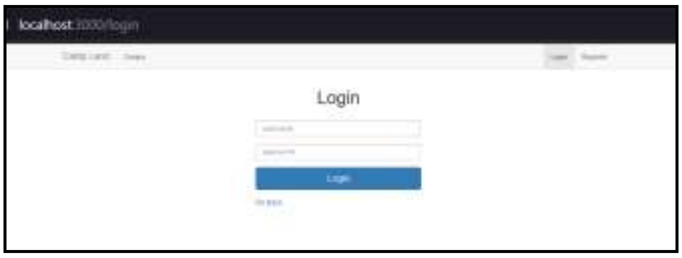


Fig. 3.3

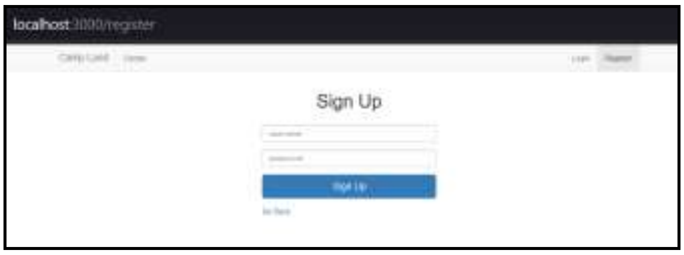


Fig. 3.4

Skeletal Working:

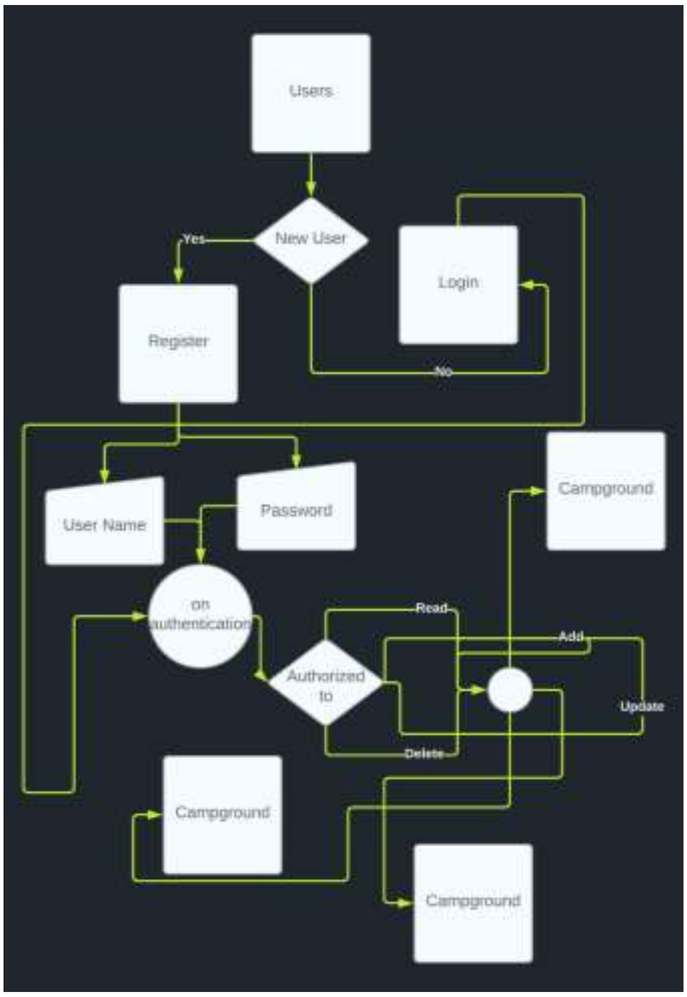
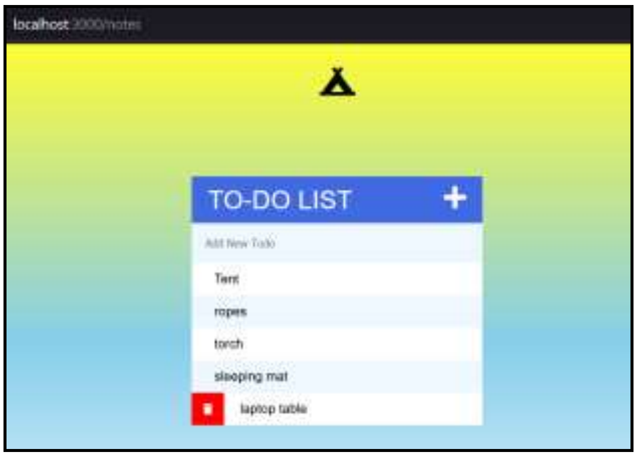


FIG. 4

❖ There is also one more collection named notes that had been added to the application this is only accessed by the user login status and the values displayed are exclusive to the user. This was added as additional functionality for the user for making a to-do type list that can be added or deleted with the neat UI using jQuery.



D. Database

MongoDB has provided cloud support for a developer to host applications and collects different types of metrics/status of database interaction and modifications over the internet by the users

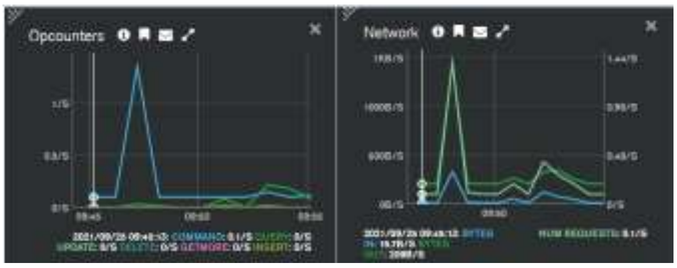


Fig. 5.1

Figure 3 shows certain details about the cluster encounter modifications in the last 1 hour (can be filtered by our requirements) and also network interaction over the same time And also we can even create multiple profiles of the same cluster and use our application as secondary users in the form of API shown in figure 4

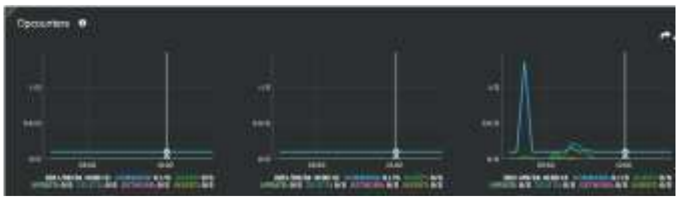


FIG. 5.2

The application was deployed using Heroku and metrics were collected using from the API MongoDB given connected through the deployed app

CONCLUSION

Though there are many sources it is more efficient and accurate for using a web application for these kinds of problems and also the several technologies were indeed implemented in the past, there is always room to get better here NodeJS played the role that overcame the prior technologies through various better structure and solutions for the real-world problems as such is in the paper. In future work, we thought of implementing declarative programming using front-end frameworks (such as react) which enables us to use asynchronous methods to call backend without disturbing the client's UI, such as these will enable us to skip through the post calls rendering pages async.

REFERENCES

[1] Dayley, B., Dayley, B., & Dayley, C. (2017). Node. js, MongoDB, and Angular Web Development: The definitive guide to using the MEAN stack to build web applications. Addison-Wesley Professional.

[2] KUMAR, C. (2021). TOURISM WEB APPLICATION.

[3] Callejo, A. M., & Singun, A. P. Abra iTour: A Semantic Web Recommender Using Hybrid Algorithm.

[4] van Deursen, A., Aniche, M., & Aué, J. (2016). Delft Students on Software Architecture: DESOSA 2016. Delft University of Technology.

[5] Subramani, K., Hemapriya, V., Libertina, A. A., & Yazhini, V. R. (2016). Node JS: Building an High Performance Event Manager in Android Platform. Australian Journal of Basic and Applied Sciences, 10(1), 174-177.

[6] Vu, B. (2016). Developing pilot API application using NODE JS: a case study of 1UP media Oy.

[7] Learn to Create Robust RESTful Web Services with Node.js, MongoDB, and Express.js, 3rd Edition By Valentin Bojinov · 2018

[8] Node.js Security, Verbitskiy, Ilya, January 1, 2017

[9] Developing a User Management Dashboard with Fullstack Javascript Le, Duy (2019)

[10] Anssi Hautaviita DEVELOPING A WEB APPLICATION ON THE MEVN STACK