# LEARN ENGLISH AND GROW

**MINI PROJECT REPORT**

Submitted by

**AMIT KUMAR SINGH**
**ROLLNO-1901320130013**

Submitted in partial fulfillment of the Requirements for the award of

**Degree of Bachelor of Technology in information technology**

**SUBMITTED TO:**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

Greater Noida Institute of Engineering

and technology, Greater Noida

## CERTIFICATE

Certified that report ‘***LEARN ENGLISH AND GROW*** ’ is the report of mini project presented by **AMIT KUMAR SINGH**, **1901320130013** during year **2020-2021** in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology in Information Technology of the Dr . A.P.J Abdul Kalam Technical University.

**Mrs. Jasneet Kaur**
Designation
Dept. of Information Technology
Greater Noida Institute of Technology, Greater Noida

## DECLARATION

I, hereby declare that, this project entitled ‘Learn english and grow’ is the bonafide work of mine carried out under the supervision of
**Mrs. Jasneet kaur** I declare that, to the best of my knowledge,
The work reported here in does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion to any other candidate. The content of this report is not being presented by any other student to this or any other University for the award of a degree.

Signature- Amit kumar

Roll no-1901320130013

DATE  18-Jan-2021

## ACKNOWLEDGEMENT

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

I am highly indebted to **Mrs. Jasneet kaur**, Professor, Department of Information Technology), for her excellent guidance, positive criticism and valuable comments. I am greatly thankful to **Dr. Ramveer Singh**, Head of Information Technology for his support and cooperation.
Finally, I thank my parents and friends near and dear ones who directly and indirectly contributed to the successful completion of my mini project.

**Amit kumar singh**

Date 18-JAN-2021

# ABSTRACT

There are lots of English learning apps in Google play store. you can download hundreds of them but still you will lag in speaking English when you face the real world. where you need confidence for talking with humans.  Now, the question comes how they can start?

If you will start now speaking broken English then everyone will laugh on you and hence confidence will be degraded by this. Also I have seen many of my classmates who can score like 80+ in English subject but still they can't speak English in person.

So, here comes the REAL solution. I am working upon a project which can solve this real life problem. there you will find a huge space which may help you to enhance your confidence while speaking.

# CONTENTS

# CHAPTER 1

## INTRODUCTION

English language plays a crucial role in our day to day life. it is an official language which is used in many different countries across the world. So it's very important to know English language in this era. Now, with learning something new, there comes some problems in our path.

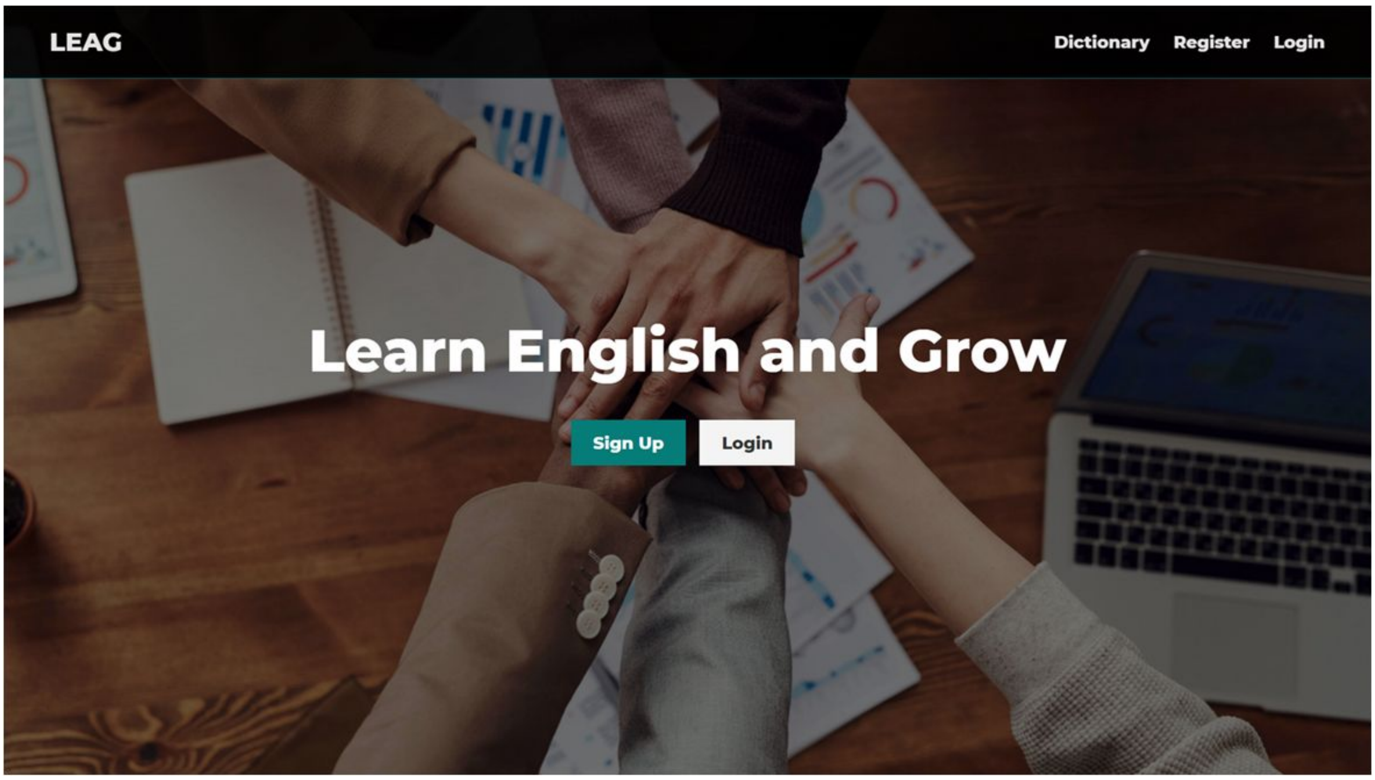
We can spend lots of money and time to learn bookish knowledge but how can we conquer the real world ? Is it sufficient to learn all that present tense, past tense, future tense to speak so confidently in any interview, in any meeting, in public speaking events? Answer is a straight no. Until or unless you get the confidence to face the real world and speak whatever comes to mind in English with fluency. Here comes the role of this mini project. there you will not find notes, quizzes, or any sort of bookish knowledge. But people just like you who want to learn English and build that confidence by interacting with you. By speaking some broken English with you and  they will not laugh at you ,rather they will be learning there with you.

# CHAPTER 2

## FEATURES

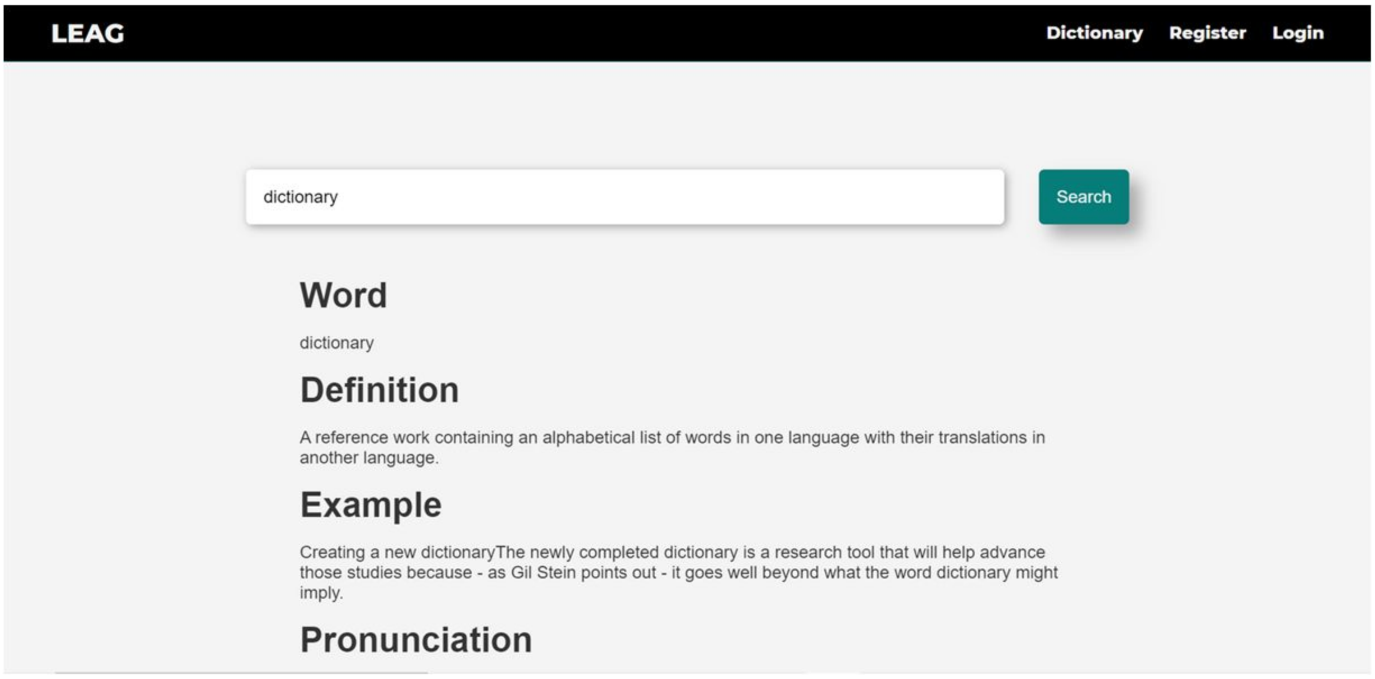
### 2.1 Easy Interaction

You just have to create an account here and then you can interact with people across the globe. And don't worry about what to say. Just consider that here you are to improve your speaking skills.



## 2.2 Dictionary

We have a feature through which you can have a dictionary in your hand. So that you can correct and improve your english vocabulary while interaction.



Word

dictionary

Definition

A reference work containing an alphabetical list of words in one language with their translations in another language.

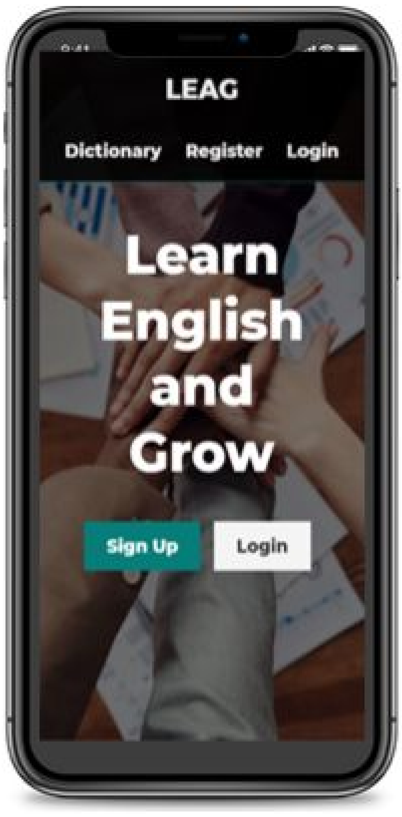
Example

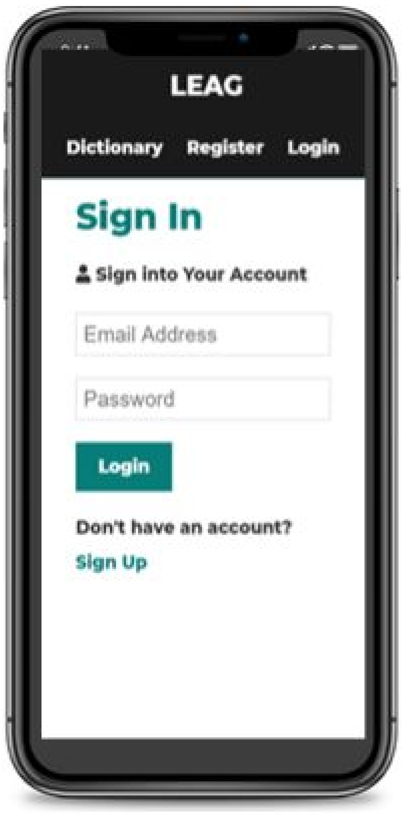
Creating a new dictionaryThe newly completed dictionary is a research tool that will help advance those studies because - as Gil Stein points out - it goes well beyond what the word dictionary might imply.

Pronunciation

## 2.3 Responsive web design

It's responsive web design can adapt to the size of the visitor's viewport. The primary benefit of responsive web design is that sites load quickly without any distortions, so users don't need to manually resize anything to view content.

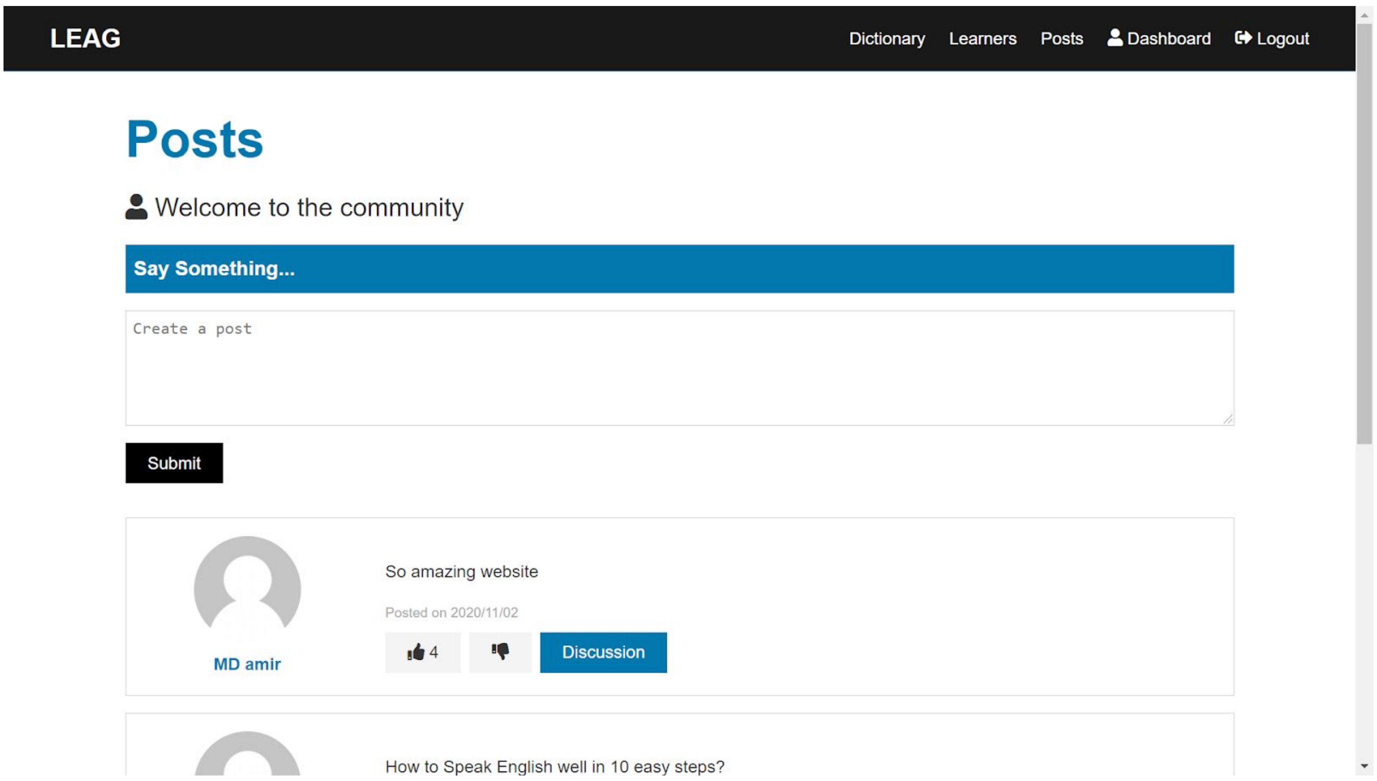




## 2.4 Write posts,Comment, like, delete posts/comment,reply

Once you get familiar, you will find a space of posts where you can tell your story by writing posts. It can be an inspiration for other people.

Telling your story through posts will help others. Once you succeed to overcome your fear and are able to speak with fluency then there comes the time to share your whole journey from coping up problems and achieving goals.



Posts

Welcome to the community

Say Something...

Create a post

Submit

So amazing website

Posted on 2020/11/02

4

Discussion

MD amir

How to Speak English well in 10 easy steps?

# CHAPTER - 3

## TECHNOLOGIES USED

### 3.1 Frontend

Back in the day, websites were simple, static text sites with a bit of formatting and maybe even some animation. That was all thanks to HTML and CSS.

It’s important to note that front-end development has changed significantly over the past 10 to 15 years with the explosive growth of JavaScript, which wasn’t as ubiquitous on the front end as it is now, or even as common on the back end.

Some of the core technologies used in front-end web development include:

**HTML: The Organizer**

- HTML is how every site on the web is organized, so it’s a big one you can’t live without in front-end development.

- HTML5 is the latest HTML specification. Get a full run-down of this evolving backbone of front-end web technology

## CSS: The Stylist

Cascading Style Sheets (CSS) is how developers add styling and effects to a website. Styles can be added globally, then layered on without changing that fundamental styling that gets applied to a whole site.

- CSS is always evolving. Get to know what’s new with the latest and greatest version, CSS3.
- Many front-end developers use a CSS preprocessor to make writing lots of lines of CSS even faster.
- CSS front-end frameworks like Foundation or Bootstrap can help you create polished websites in a snap.

## JavaScript: The Multitasker

What was once a supplementary tool to make web pages more interactive is now the most ubiquitous client-side technology. JavaScript is more than just a language — it’s an entire ecosystem that spans frameworks, task runners, server-side development, and more.

### 3.2 Backend

Back-end development can be much more varied than front-end development, which is largely driven by JavaScript, HTML, CSS, and various front-end frameworks using these languages.

To simplify things, we’ll break the server-side down into four main components of a “software stack”: the server, the database, the operating system, and the software.

## 1. Languages & Frameworks

A variety of programming languages and frameworks are involved in building the software aka back-end. Frameworks are libraries of pre-written code with a pre-imposed structure that a back-end developer can use according to the requirements and needs. Whereas, a programming language is a superset of scripting languages like Ruby, Java, Python, PHP, Perl, Erlang, and **Node.js** which can be used to write instructions for execution.

## 2. Web Servers

Web servers are computer programs that store, process and deliver web pages to the users.there are many web servers out there but I found heroku is the best for deployment.

## 3. Database Management Systems

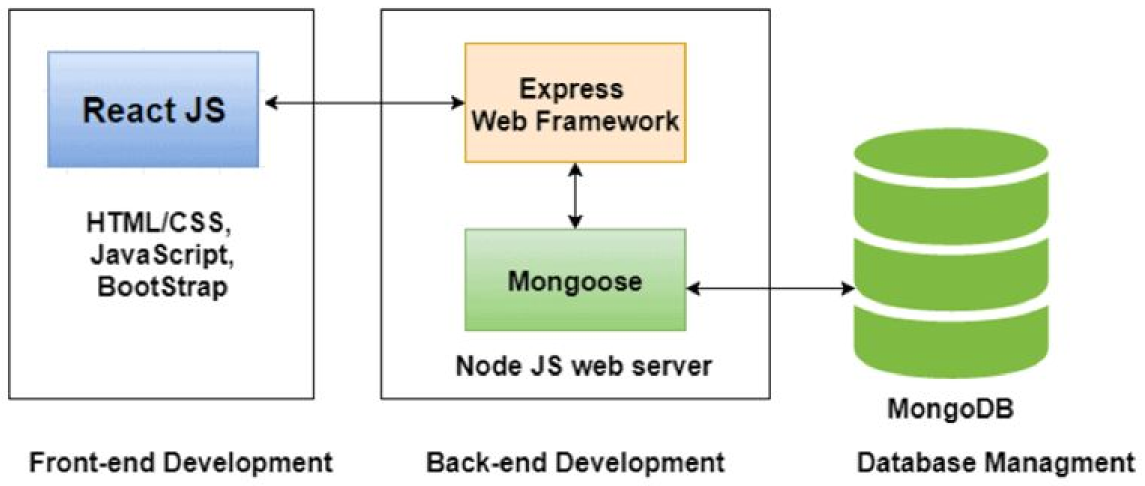
A Database Management System (DBMS) is a collection of programs that enables its users to access a database, manipulate, interpret and represent data. MySQL is the world's most popular open-source relational database. It's not only accessible but also free. Its ease of setup and speedy performances make it a favorite among many backend developers. On the other hand, **MongoDB** is an open-source NoSQL database system which is closely associated with a JavaScript-based set of technologies like **ExpressJS** and **NodeJS**.

## 4. Local Development Environments

All back-end developers will swear by the importance of a local test environment. The advantage of using a local site that's visible only to you gives you the liberty to try codes and experiments before the site goes live. XAMPP and WampServer are examples of open source windows development environments that allow users to use web applications with Apache, PHP, and MySQL database.

# CHAPTER - 4

## WORKFLOW



# CHAPTER 5

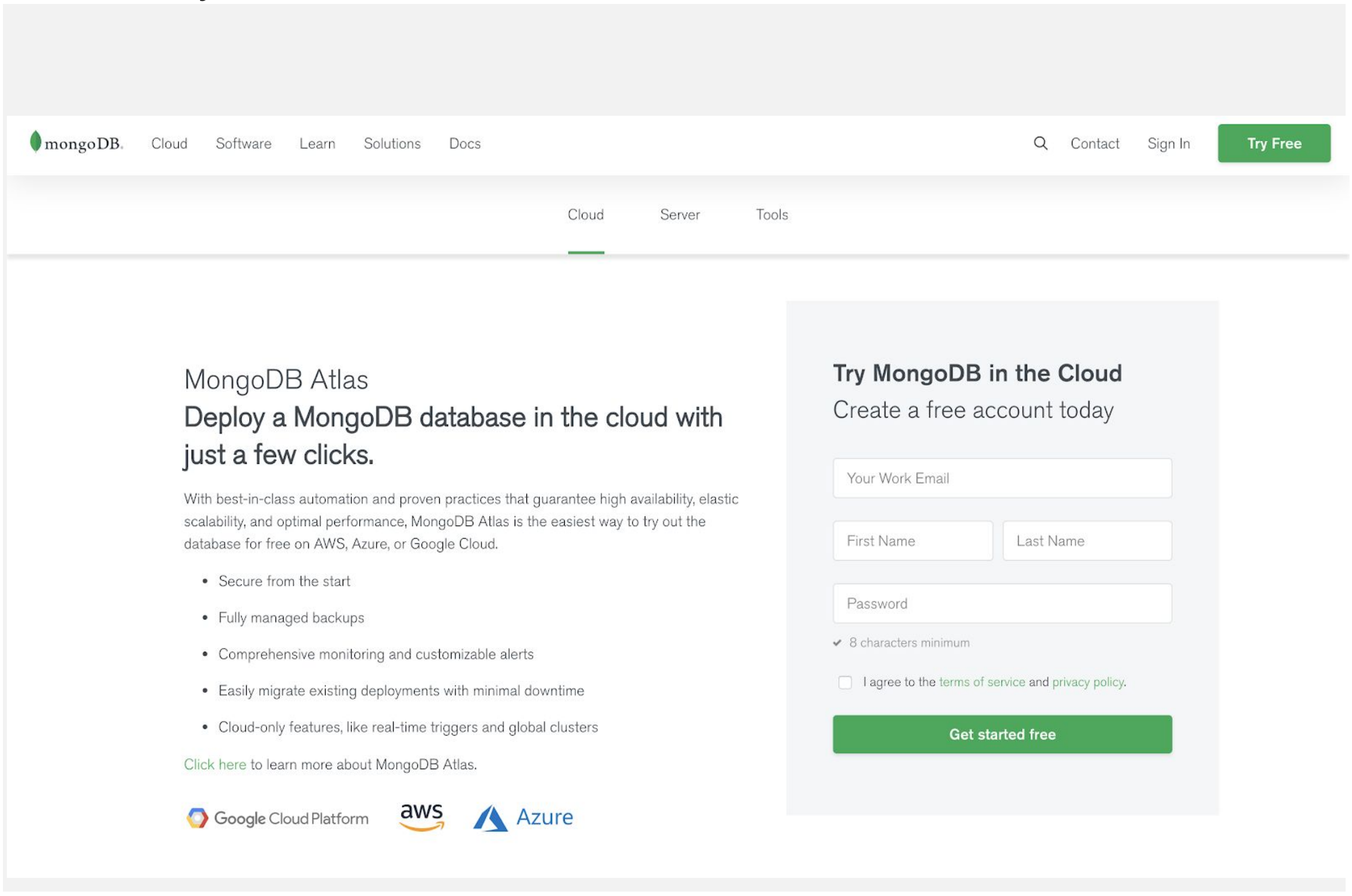
Connecting to MongoDB Atlas

## What is MongoDB Atlas?

It is a global cloud database service developed by MongoDB. You are given the option to choose the cloud service provider. Atlas will handle the complexity of deploying, managing, and healing your deployments.

## Getting Started

First of all you will have to make an account on their site.



MongoDB Atlas
Deploy a MongoDB database in the cloud with just a few clicks.

With best-in-class automation and proven practices that guarantee high availability, elastic scalability, and optimal performance, MongoDB Atlas is the easiest way to try out the database for free on AWS, Azure, or Google Cloud.

- Secure from the start
- Fully managed backups
- Comprehensive monitoring and customizable alerts
- Easily migrate existing deployments with minimal downtime
- Cloud-only features, like real-time triggers and global clusters

Click here to learn more about MongoDB Atlas.

Try MongoDB in the Cloud
Create a free account today

Your Work Email
First Name
Last Name
Password
8 characters minimum
I agree to the terms of service and privacy policy.

Next up is creating a cluster. Click on the green button that says “Build a Cluster”. It will then ask you what type of cluster you want to create.

-Starter Cluster

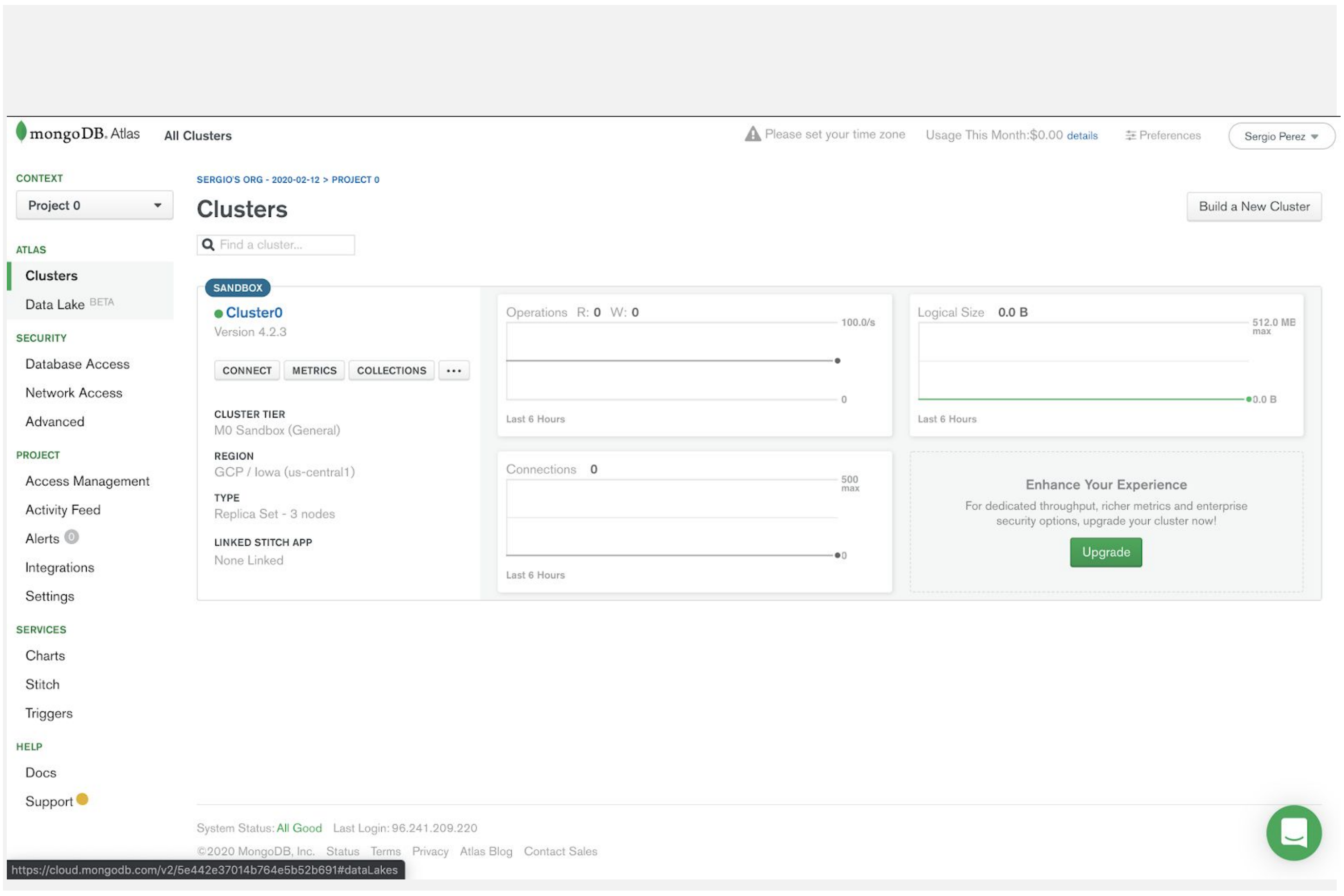
-Single-Region Cluster

-Multi-Region Cluster

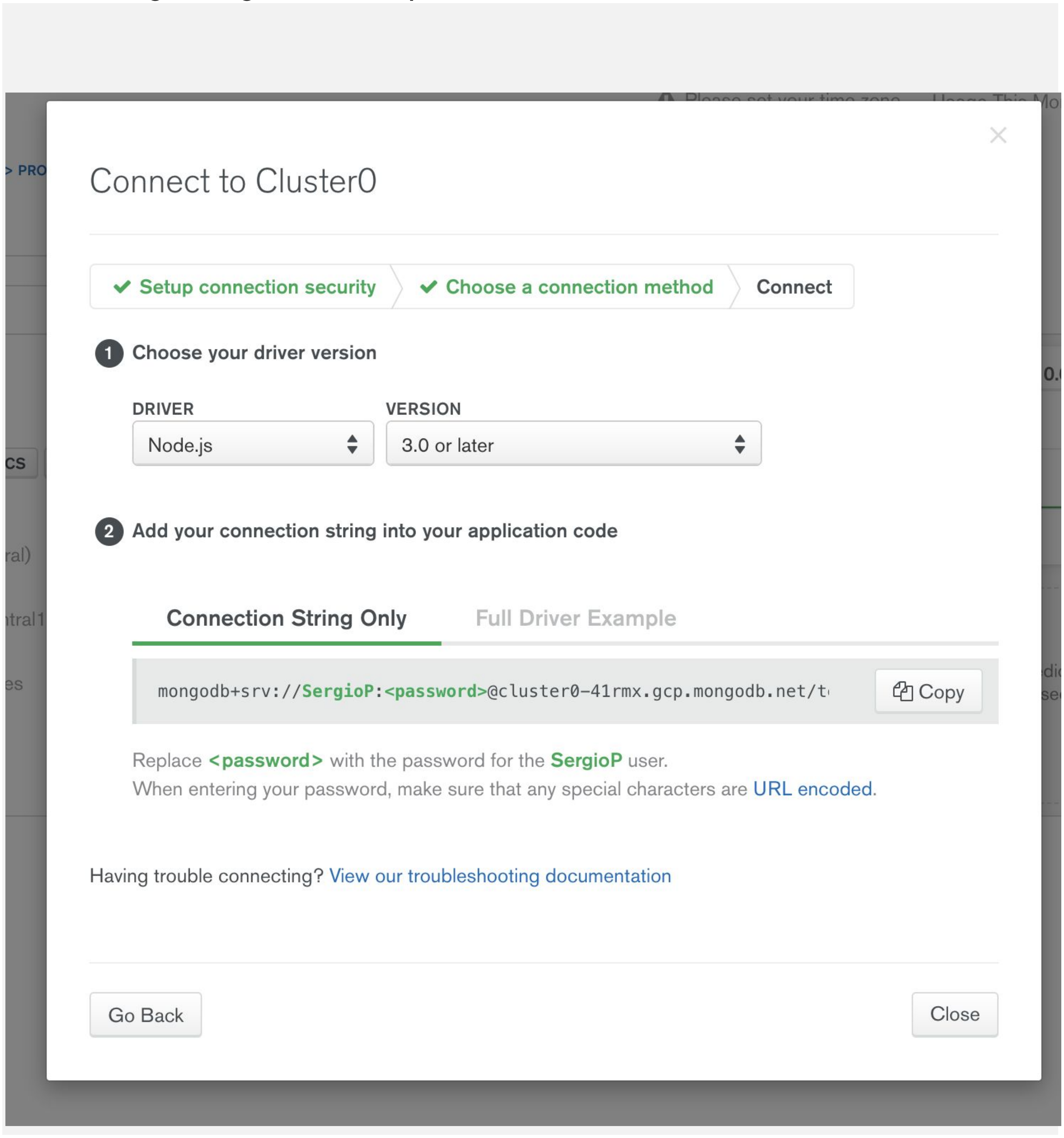
We will go with a Starter Cluster since it is free and we will only be using it for a personal project.

Then it will ask what cloud provider you would like to use. For my case I will be using Google’s Cloud Platform, you have the option to choose AWS or Azure. I will be using the “Iowa” region since it has the “free tier available” tag. You do not need to touch the Cluster Tier or the Additional Settings. Feel free to change the Cluster Name .

It may take 1–3 minutes for the cluster to be created. Once it is done you should be greeted with the following screen.


mongoDB. Atlas
All Clusters
Please set your time zone
Usage This Month:$0.00 details
Preferences
Sergio Perez
CONTEXT
Project 0
ATLAS
Clusters
Data Lake BETA
SECURITY
Database Access
Network Access
Advanced
PROJECT
Access Management
Activity Feed
Alerts 0
Integrations
Settings
SERVICES
Charts
Stitch
Triggers
HELP
Docs
Support
SERGIO'S ORG - 2020-02-12 > PROJECT 0
Clusters
Build a New Cluster
Find a cluster...
SANDBOX
Cluster0
Version 4.2.3
CONNECT
METRICS
COLLECTIONS
CLUSTER TIER
M0 Sandbox (General)
REGION
GCP / Iowa (us-central1)
TYPE
Replica Set - 3 nodes
LINKED STITCH APP
None Linked
Operations R: 0 W: 0
100.0/s
0
Last 6 Hours
Logical Size 0.0 B
512.0 MB max
0.0 B
Last 6 Hours
Connections 0
500 max
0
Last 6 Hours
Enhance Your Experience
For dedicated throughput, richer metrics and enterprise security options, upgrade your cluster now!
Upgrade
System Status: All Good Last Login: 96.241.209.220
©2020 MongoDB, Inc. Status Terms Privacy Atlas Blog Contact Sales
https://cloud.mongodb.com/v2/5e442e37014b764e5b52b691#dataLakes

Click on the “Connect” button right under the cluster name. It will give you 3 options to connect to it from your application. I will be going with the 2nd one “Connect Your Application”, this option will give you a connecting string and examples on how to use it.


Connect to Cluster0
✓ Setup connection security
✓ Choose a connection method
Connect
1 Choose your driver version
DRIVER
Node.js
VERSION
3.0 or later
2 Add your connection string into your application code
Connection String Only
Full Driver Example
Copy
Go Back
Close

```
mongodb+srv://SergioP:<password>@cluster0-41rmx.gcp.mongodb.net/t
```

Replace <password> with the password for the SergioP user.
When entering your password, make sure that any special characters are URL encoded.

Having trouble connecting? View our troubleshooting documentation

Driver option should be set to Node.js and the version should be 3.0 or later. You will need to copy the connection string.

Example of what my connection string looks like:

*“mongodb+srv://leag:<password>@cluster0–41rmx.gcp.mongodb.net/test?retryWrites=true&w=majority”*

You will need to change the <password> text with your password that you set up when creating the MongoDB Atlas user.

## Connecting it to your application

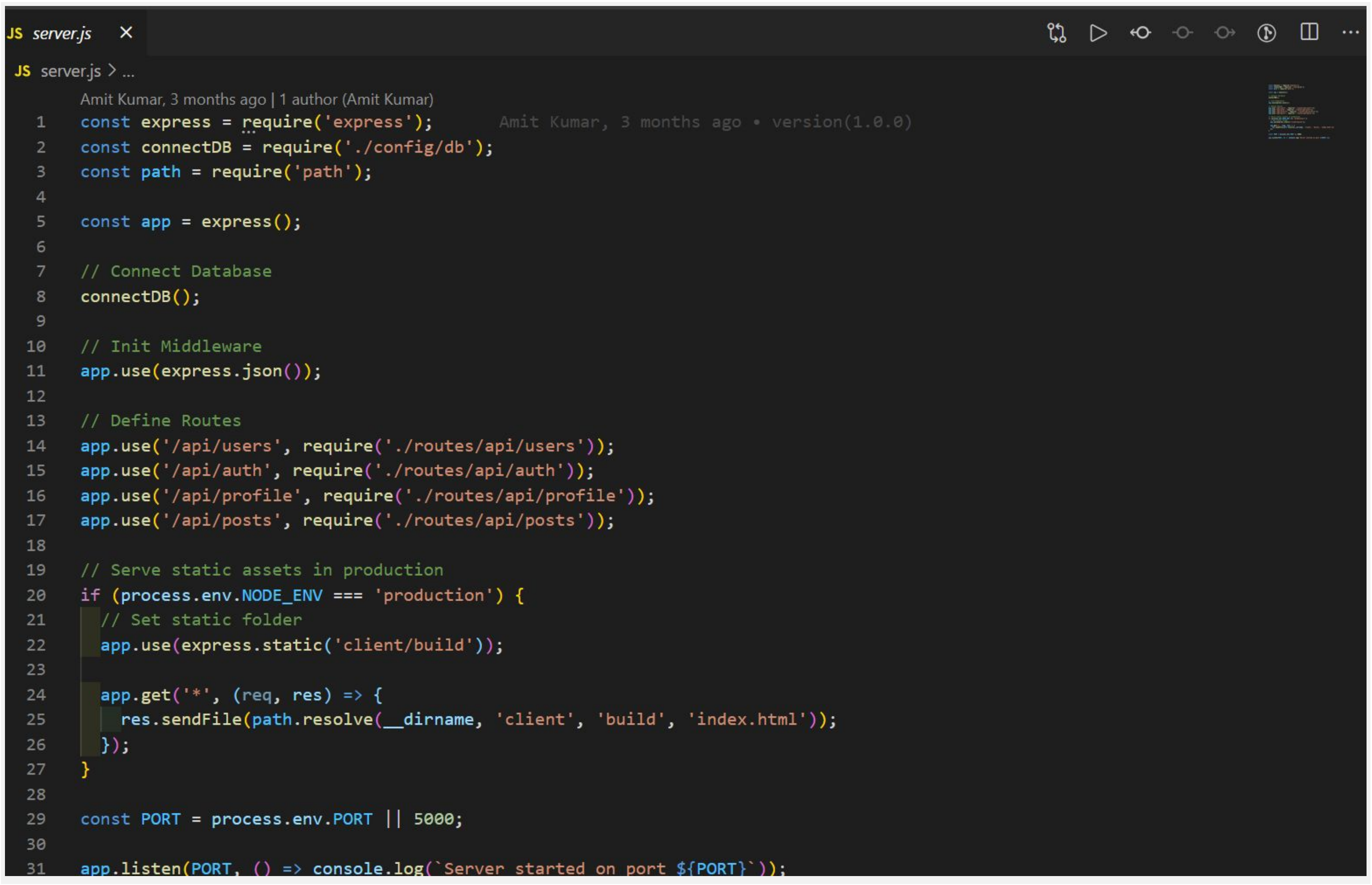
I have a simple NodeJS application with one route for example purposes. Preferably you will want to connect to the MongoDB Atlas cluster before Node listens for your server.

Also keep in mind that we will be using mongoose to connect. So if you need to install it you can do so with the following command lines.

```
npm install mongoose
```

And require it in your server js file.

```
const mongoose = require('mongoose')
```



JS server.js ×

JS server.js > ...

Amit Kumar, 3 months ago | 1 author (Amit Kumar)

```
const express = require('express');      Amit Kumar, 3 months ago • version(1.0.0)
const connectDB = require('./config/db');
const path = require('path');

const app = express();

// Connect Database
connectDB();

// Init Middleware
app.use(express.json());

// Define Routes
app.use('/api/users', require('./routes/api/users'));
app.use('/api/auth', require('./routes/api/auth'));
app.use('/api/profile', require('./routes/api/profile'));
app.use('/api/posts', require('./routes/api/posts'));

// Serve static assets in production
if (process.env.NODE_ENV === 'production') {
  // Set static folder
  app.use(express.static('client/build'));

  app.get('*', (req, res) => {
    res.sendFile(path.resolve(__dirname, 'client', 'build', 'index.html'));
  });
}

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => console.log(`Server started on port ${PORT}`));
```

Now with the connection string we copied we will insert the following code:

const uri = “YOUR CONNECTION STRING“;

mongoose.connect(uri, {

 useNewUrlParser: true,

 useUnifiedTopology: true})

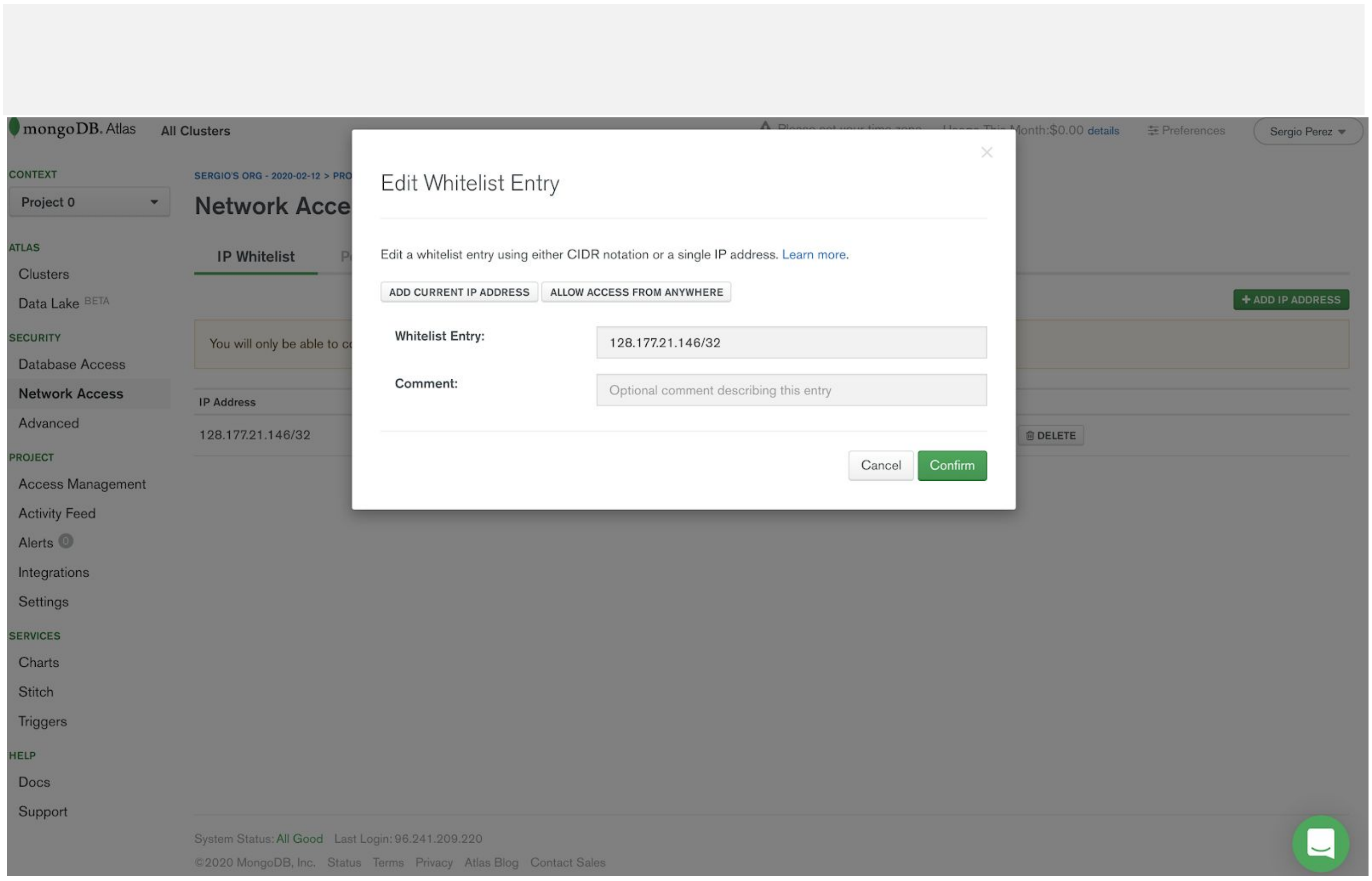
```
.then(() => {

 console.log('MongoDB Connected…')

})

.catch(err => console.log(err))
```

Using the connect method that mongoose provides us with we are first passing in the connection string. Then we are passing some parameters useNewUrlParser and useUnifiedTopology.

The connect method will return a promise, we will need to handle that promise. Attaching a ".then" after the method we can console.log a success message if all goes well. We have also attached a ".catch" after that will catch any errors if we cannot connect to our MongoDB Atlas Cluster.

## Last Step Network Access

This last step is still a bit tricky for me and I assume it acts as an extra step of security. Back on the MongoDB Atlas dashboard click on "Network Access" under "Security".



Edit Whitelist Entry

Edit a whitelist entry using either CIDR notation or a single IP address. Learn more.

Whitelist Entry: 128.177.21.146/32

Comment: Optional comment describing this entry

Usage This Month: $0.00 details

You will only be able to c

128.177.21.146/32

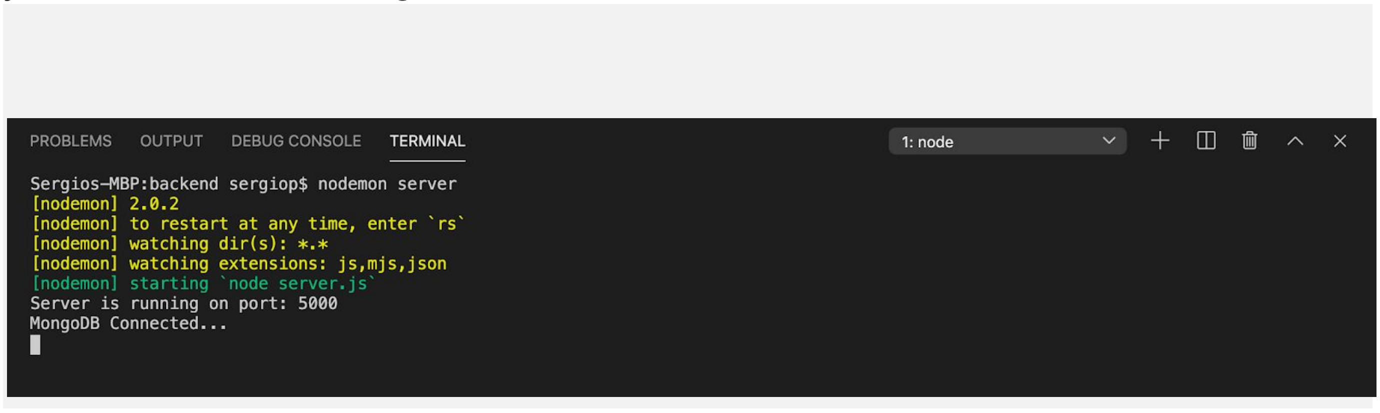
System Status: All Good Last Login: 96.241.209.220

©2020 MongoDB, Inc. Status Terms Privacy Atlas Blog Contact Sales

You want to click on “Add Current IP Address” and it will automatically generate your IP address into the field, then click “confirm”. You also do have the option to allow access from anywhere. I used this a couple times when I was testing out my connection. MongoDB will email you and tell you how dangerous this is. So I do not recommend it unless you are quickly testing it.

## Conclusion

You should be able to run your node server now and should be receiving your success message in the console.



```
Sergios-MBP:backend sergiop$ nodemon server
[nodemon] 2.0.2
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
Server is running on port: 5000
MongoDB Connected...
```

I have not deployed my project where I am using this MongoDB Atlas Connection so I am uncertain if it will break once it gets deployed.

Also, I have had to log back into the MongoDB dashboard and update my IP address when I work in a different location.

# CHAPTER-6

## Passport Authentication in Node.js application

Authentication in Node.js using Passport.js is quite common. There are a number of mechanisms in Passport.js for authenticating requests, like authentication using a username and password, single sign-on authentication using Facebook, Twitter, and more.

### What is Passport.js?

It is an authentication middleware for Express-based applications using Node.js. Being a middleware, it has access to request, and response objects, and can manipulate them in request-response cycle. It is extremely easy to integrate into any application.

Put simply, Passport has only one thing to do- authenticate requests.

I have assumed that:

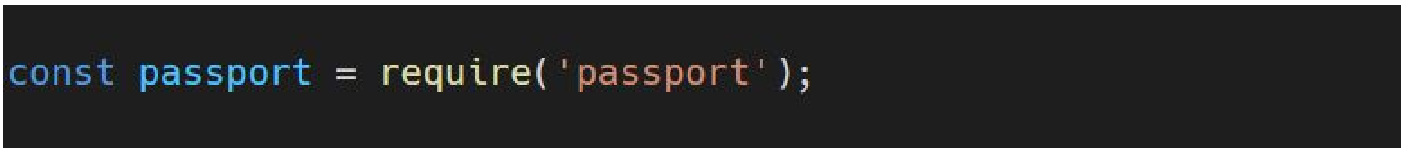
i) you have set up an Express server (and that you are familiar with Routing in Express).

ii) you have connected your application to a database (I am using MongoDB, as its document-oriented structure is easy to keep track of all data).

Now, as said earlier, there are a number of strategies in Passport to employ for authentication. We will be using the Local Strategy to authenticate requests using username and password. A strategy is an "authentication mechanism" or a "way to verify", and packaged as individual modules, so that only required dependencies are installed in projects.

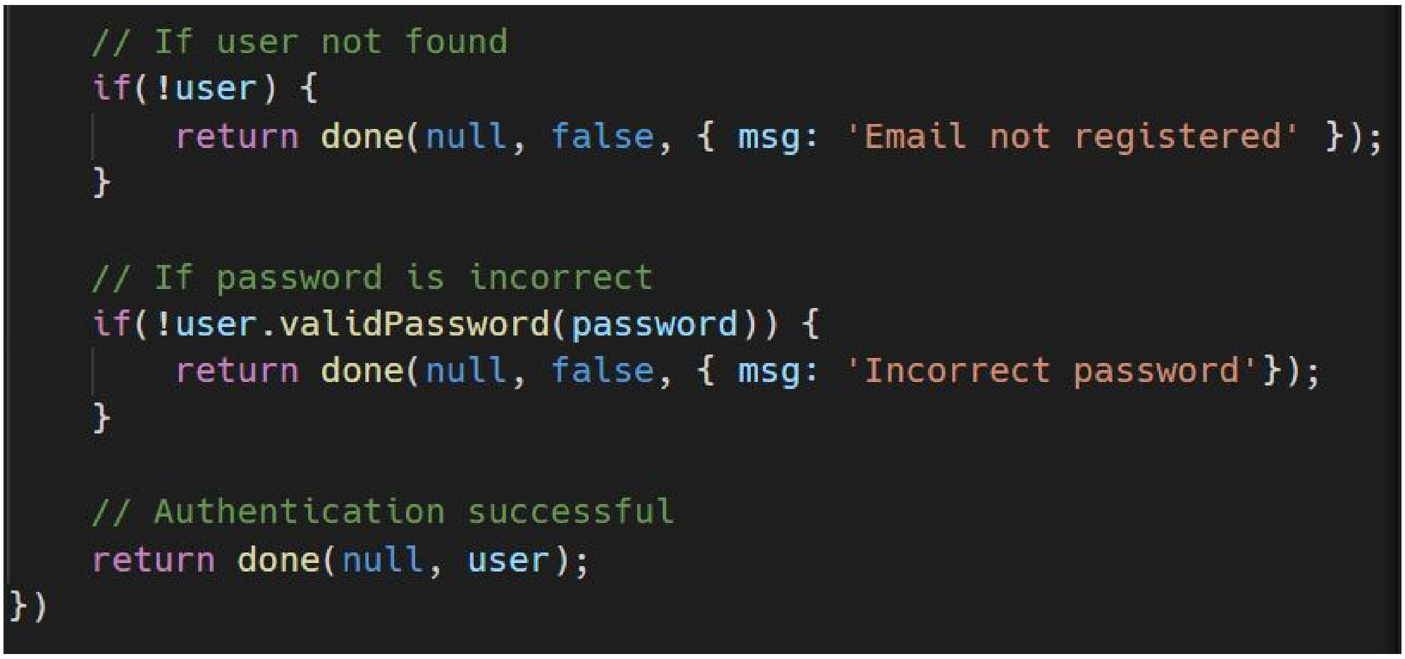
To add an authentication layer to the application, install passport and passport-local, and integrate these modules in the "root" file or "entry point" file.

**npm install passport passport-local**

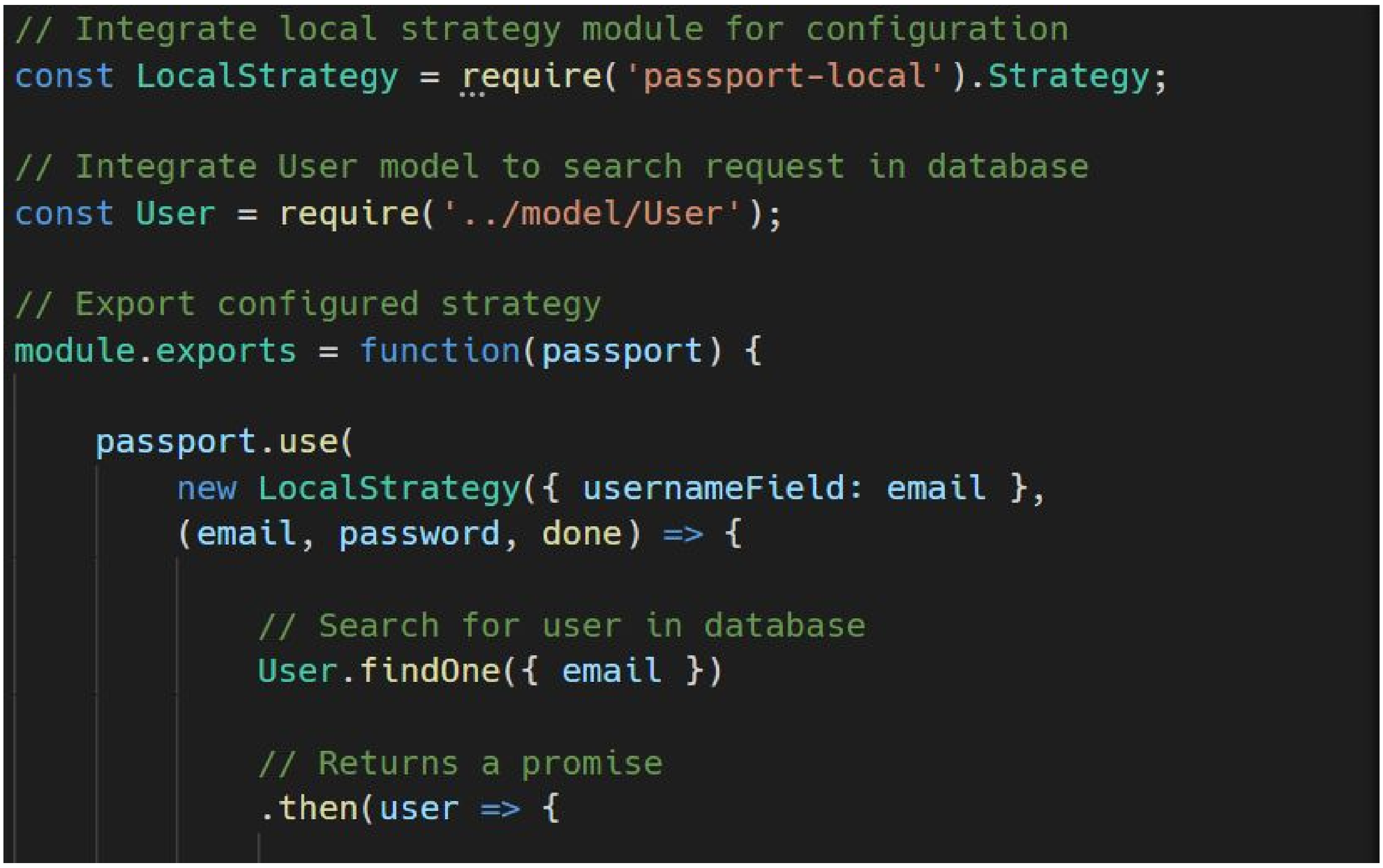


```
const passport = require('passport');
```

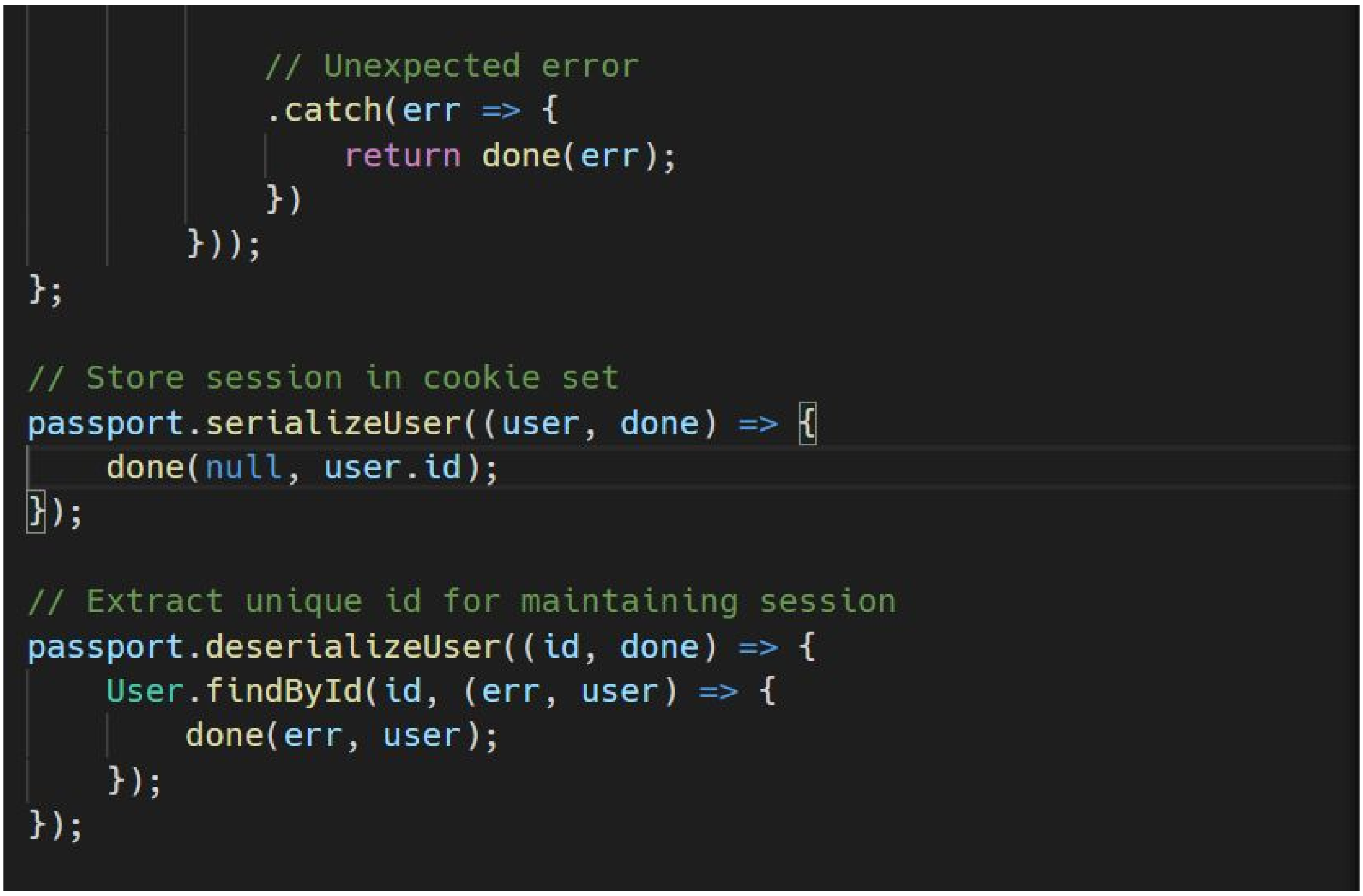
Next, we have to configure our Local strategy to be able to use it. The configuration of the strategy involves a **verify callback** which takes the username, and password entered by the user as arguments, and verify these credentials with the database (in our case, MongoDB)



```javascript
    // If user not found
    if(!user) {
        return done(null, false, { msg: 'Email not registered' });
    }

    // If password is incorrect
    if(!user.validPassword(password)) {
        return done(null, false, { msg: 'Incorrect password'});
    }

    // Authentication successful
    return done(null, user);
})
```



```javascript
// Integrate local strategy module for configuration
const LocalStrategy = require('passport-local').Strategy;

// Integrate User model to search request in database
const User = require('../model/User');

// Export configured strategy
module.exports = function(passport) {

    passport.use(
        new LocalStrategy({ usernameField: email },
        (email, password, done) => {

            // Search for user in database
            User.findOne({ email })

            // Returns a promise
            .then(user => {
```



```
                // Unexpected error
                .catch(err => {
                    return done(err);
                })
            }));
};

// Store session in cookie set
passport.serializeUser((user, done) => {
    done(null, user.id);
});

// Extract unique id for maintaining session
passport.deserializeUser((id, done) => {
    User.findById(id, (err, user) => {
        done(err, user);
    });
});
```

Let’s understand the above code snippet, which is in a file called **passport.js**, residing in directory **config**.

**(i)** The passport-local module is made available for strategy configuration.

**(ii)** The User model of MongoDB is integrated next (stored in a directory **model** as **User.js**) The model facilitates the handling of data (like searching, adding, retrieving etc.). In our case, we want the model to search for a given set of credentials in MongoDB.

**(iii)** Since modularity in programming leads to a more clean, and maintainable code, we have configured the strategy in a separate file, and then exported it in the "root" file as a function.

**(iv)** The strategies, and their configurations are supplied via **use()**. Therefore, in passport.use(), an object LocalStrategy is instantiated, with the first parameter specifying the type of username (here, email), and the second parameter being the verify callback, which actually verifies the credentials.

**(v)** The credentials (inputted by user i.e., email and password) are passed automatically to the verify callback once the request to authenticate the user is made.

**(vi)** The next few lines of code performs the task of verifying the passed email, and password with those stored in MongoDB.

Note that in the verify callback, another callback **done()** is passed. It is sufficient to know that this function is called internally (by Passport) once the verification process is completed, or an error is generated (like server error). This function **done()** tells whether the authentication succeeded, or failed!

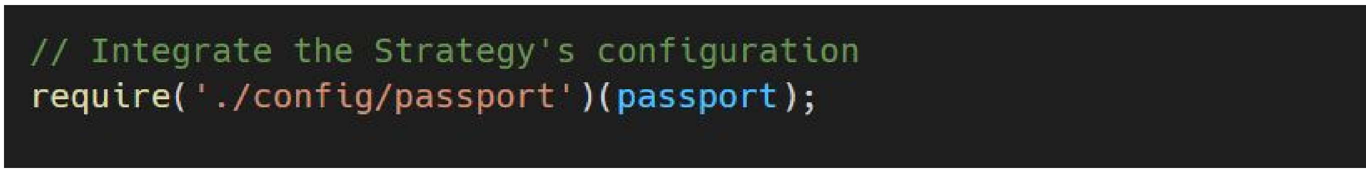
Once the authentication process is successful, a session can be created very easily. Creating a session means the user doesn't need to login every time he visits a web app or service on the web. This is achieved using **serializeUser** and **deserializeUser** in Passport. Once a request is authenticated, a constant session is maintained via a cookie set in the user's browser. The cookies only contain a small amount of data i.e., user ID (which is generated itself in MongoDB).

So, next time the user visits the application, the appropriate ID is retrieved by the browser, and the user, thus, does not need to login again.

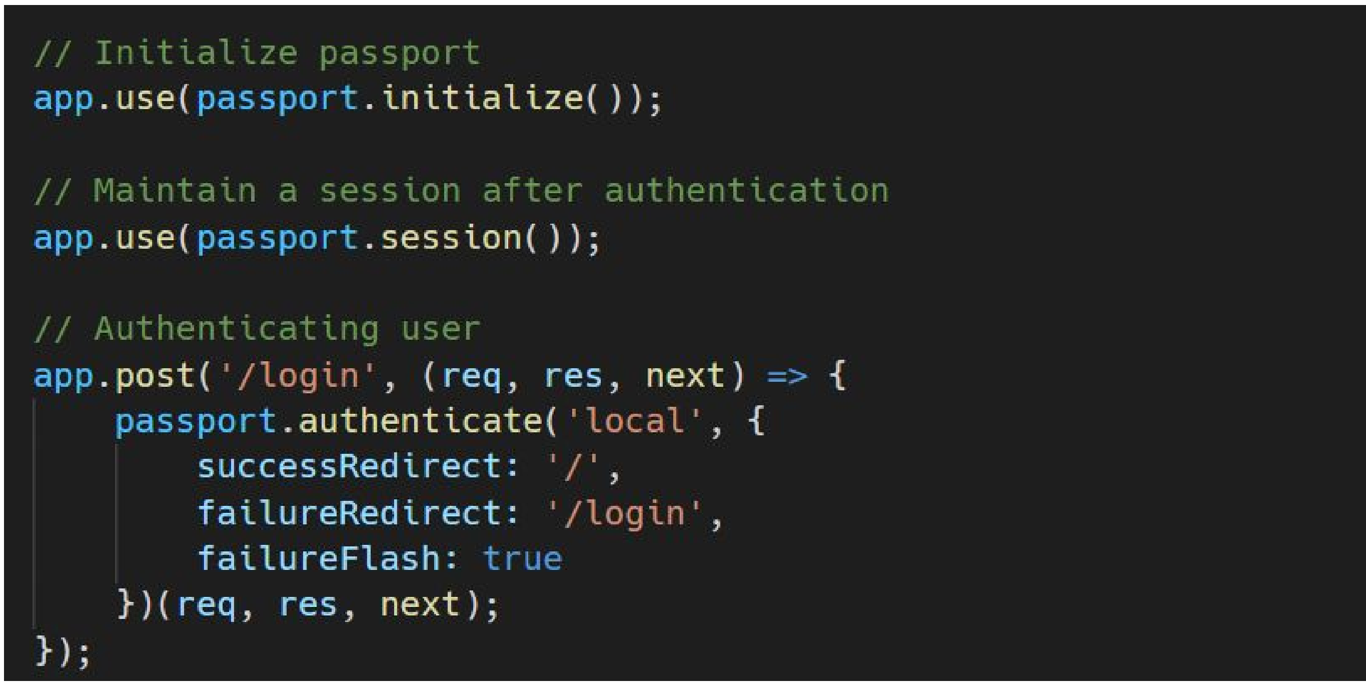
**IMPORTANT:** It is not always necessary to set up a session, because while working on an API layer in application, the user needs to verify at every stage to protect access (normally done using tokens, like JWTs).

The only thing left is to set up a route handler. Setting up a route handler essentially means instructing the application to start the authentication process once credentials are filled and submitted via a form present on a certain route (example, /login).

This step is quite easy, because we have already configured the strategy.



```
// Integrate the Strategy's configuration
require('./config/passport')(passport);
```



```
// Initialize passport
app.use(passport.initialize());

// Maintain a session after authentication
app.use(passport.session());

// Authenticating user
app.post('/login', (req, res, next) => {
    passport.authenticate('local', {
        successRedirect: '/',
        failureRedirect: '/login',
        failureFlash: true
    })(req, res, next);
});
```

Let’s demystify the above code:

**(i)** First the Local Strategy config. is made available (stored in /config/passport.js).

**(ii)** The Passport layer should be initialized first before use. So, passport.initialize() middleware is used.

**(iii)** Since we have chosen to maintain an active session after successful authentication, the passport.session() middleware is also needed.

**It is important to note** that the passport session must be used after the express.session(), to ensure sessions in proper order!

**(iv)** At the end, we have set up a route handler, which uses passport.authenticate() for verification. The first argument to authenticate() is the strategy used, and the second argument is an object specifying where to redirect, should the authentication succeeds or fails.

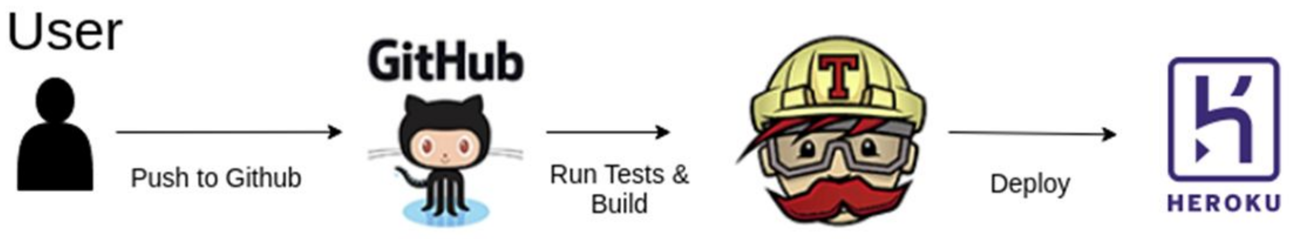
The failureFlash property ensures an error message is shown on the redirected route in case the authentication fails.

Remember the message to be shown is provided by the verify callback i.e., **{ msg }**.

COMPLETE SOURCE CODE

# CHAPTER-7

## Deployment

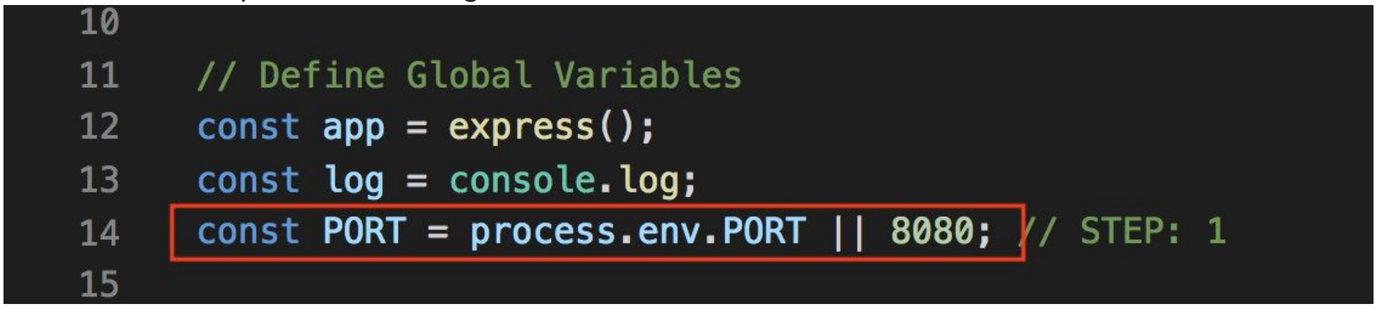


### Step 1

`server.js`

We can't assume that Heroku will have PORT 8080 available. We need to set `process.env.PORT` to our PORT variable to use Heroku PORT. Then add 8080 as a fallback port for our localhost development

We should end up with something like this



```
// Define Global Variables
const app = express();
const log = console.log;
const PORT = process.env.PORT || 8080; // STEP: 1
```

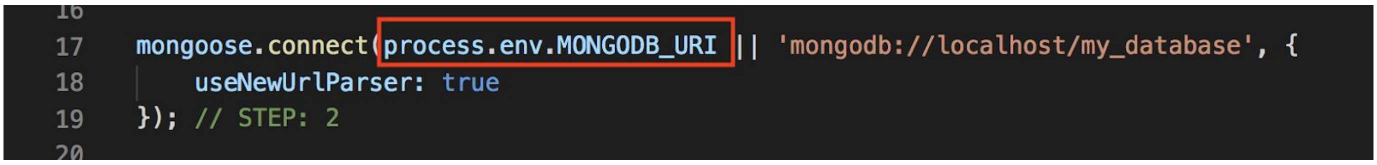
;

### Step 2

`server.js`

Heroku's going to generate an environment variable called `MONGODB_URI` for us to connect to mlab (MongoDB). In order to utilize this variable, we need to set our `mongoose.connect` with that variable.

It should look like this



```
mongoose.connect(process.env.MONGODB_URI || 'mongodb://localhost/my_database', {
    useNewUrlParser: true
}); // STEP: 2
```

;

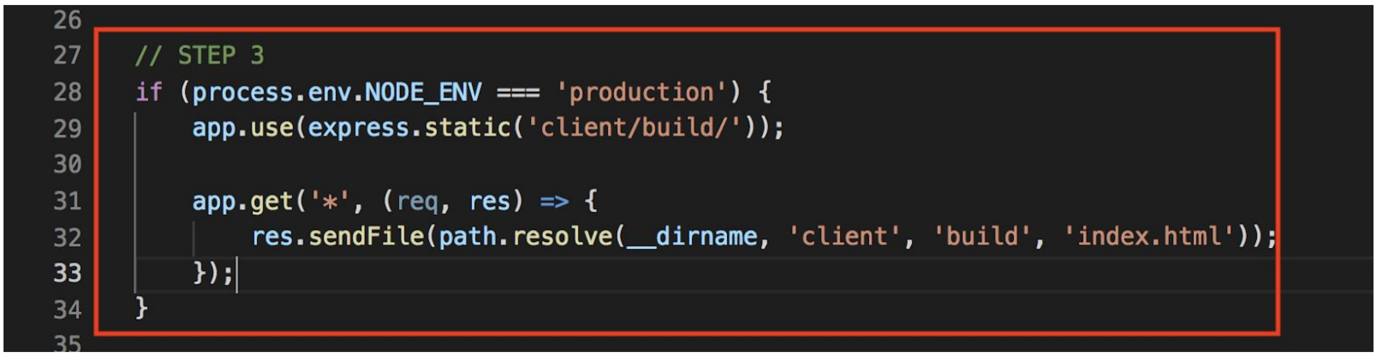
## Step 3

`server.js`

Once our app is on Heroku, we need to send the static build files on our server so that Heroku can serve it.

How do we know our app is on Heroku? Well, by default Heroku has this environment variable called `NODE_ENV` with a value set to production. We can write a conditional logic to check if `NODE_ENV` has the value of production, if so, then we know for sure that our app is on Heroku. Then we serve the static files generated by React after we have successfully run `npm run build` in the client folder.

We should end up having something like this



```
// STEP 3
if (process.env.NODE_ENV === 'production') {
    app.use(express.static('client/build/'));

    app.get('*', (req, res) => {
        res.sendFile(path.resolve(__dirname, 'client', 'build', 'index.html'));
    });
}
```

;

## Step 4

`package.json`

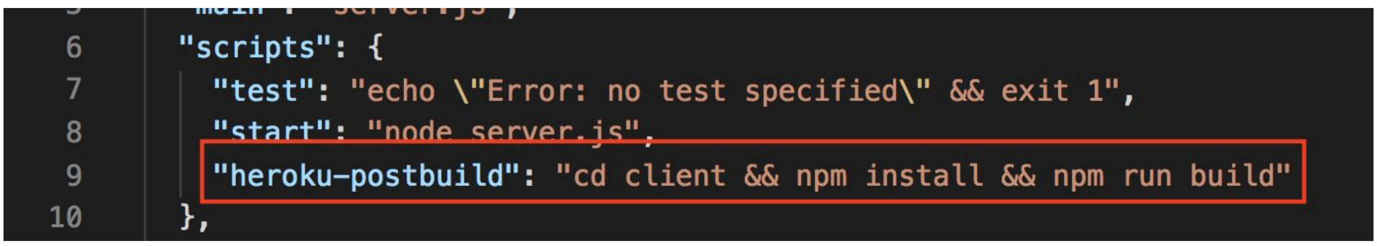
You don't want to always keep doing `npm run build` to generate the build folder for you every time you do a change. What if Heroku can do it for us? Well, the good news is, yes it can :)

Heroku has 2 builds scripts that you can run either before or after the build.

- heroku-prebuild
- heroku-postbuild

We're going to use the `heroku-postbuild` one. This is a change that we need to do inside the package.json, under the scripts section.

It should look like this



```
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
  "start": "node server.js",
  "heroku-postbuild": "cd client && npm install && npm run build"
},
```

;

Note: In order to run step 5, you need to make sure you already have `git` initialized into your project. A quick way to check is by doing `git status` in your terminal. If you see the following message `fatal: Not a git repository` . That means you do not have git initialized into your project. You can run `git init` to initialized git and continue to step 5

## Step 5

```
open your terminal
```

Once we have the above steps completed, we can run the following commands to create a heroku app, configure mlab, and push our code to heroku

```
$ heroku create app_name

$ heroku addons:create mongolab:sandbox

$ git add -A

$ git commit -m "add_message"

$ git push heroku master

$ heroku open
```

## REFERENCES-

w3schools

MDN Web Docs

Reactjs official Web Docs

npm Docs

Node.js v15.6.0 Documentation

Express

MongoDB Docs