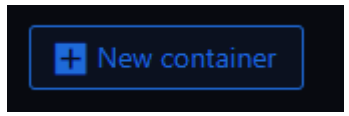
Web Development

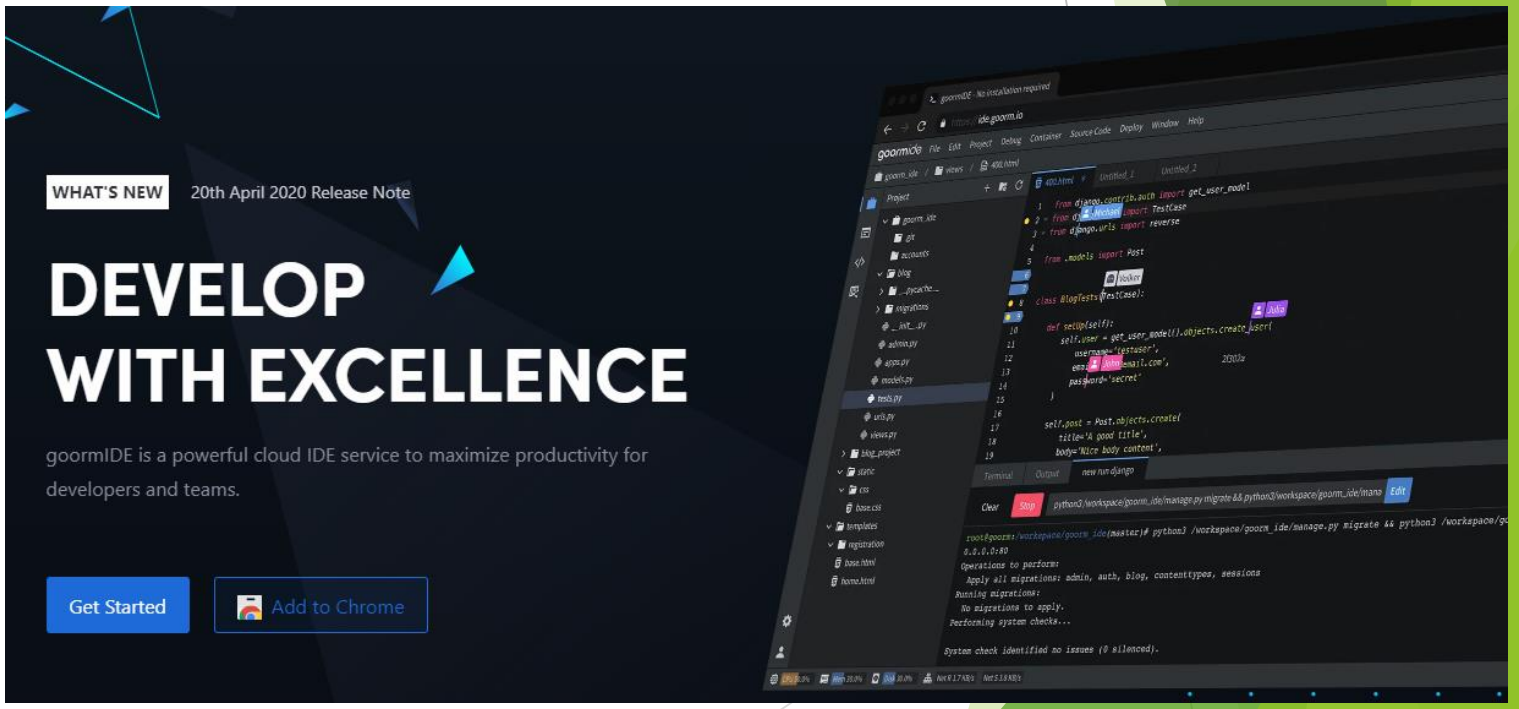
Goorm IDE, Intro Command-Line, NPM, Express, EJS

Goorm IDE

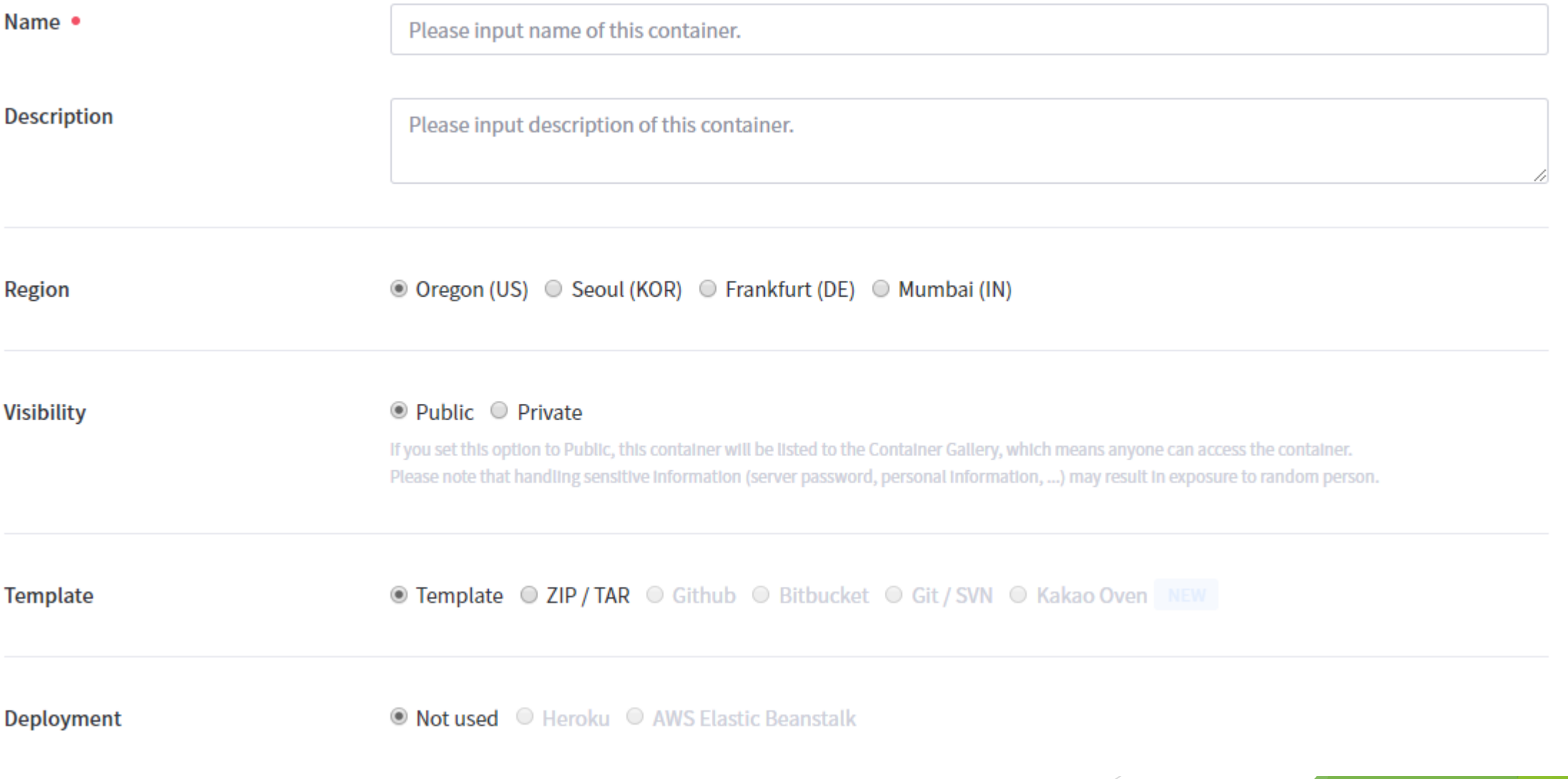
New Goorm ide Container

- Go to https://ide.goorm.io/
- Click on Get Started
- Sign Up for an account.
- Then Login.
- You will be taken to the Dashboard
- Click on +New Container

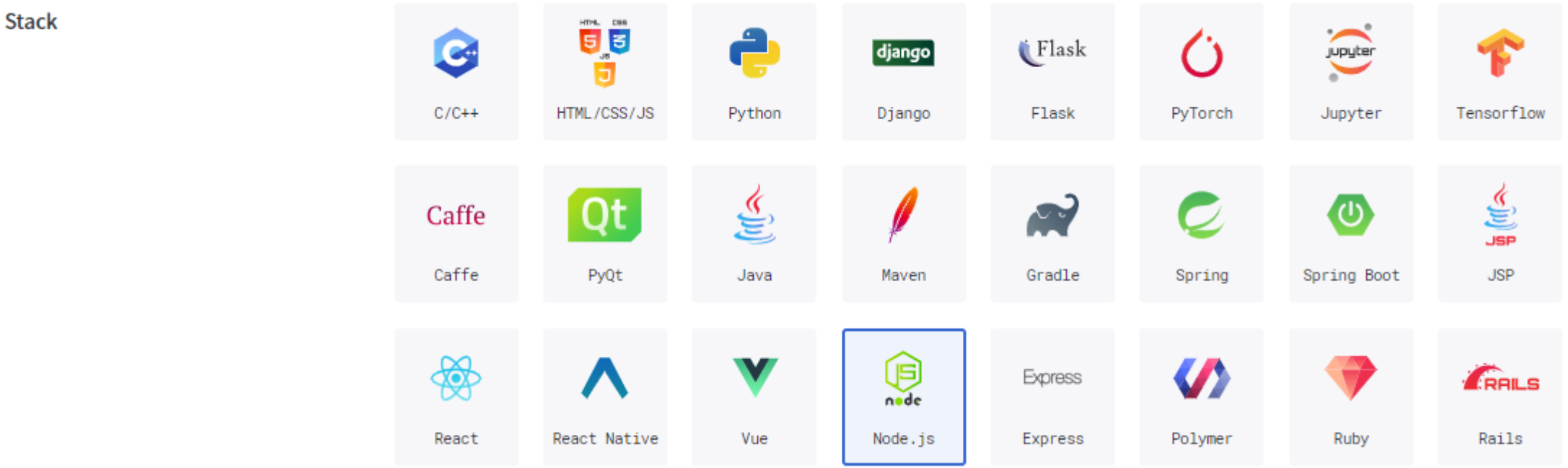




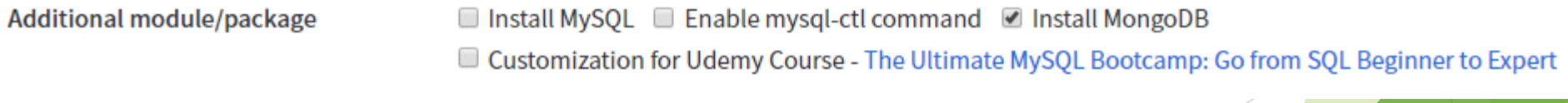
- Give it a name.
- Select Region, Visibility, Template, Deployment or keep it as default.



- In stack select Node.js

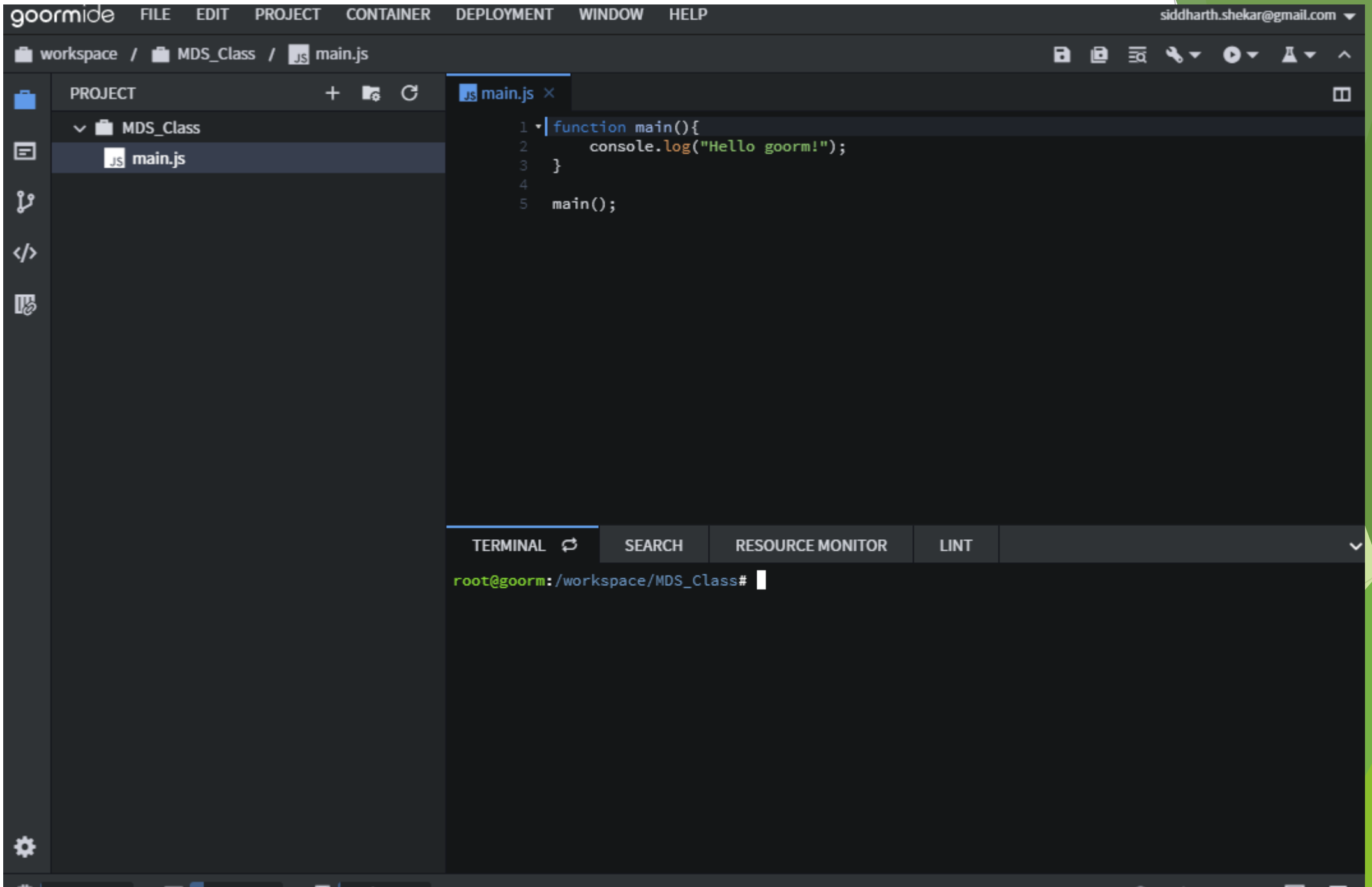


- In Additional module/ Package select installMongoDB



- Click Create on Top right

- Once Container is ready click Run Container


goormide
FILE
EDIT
PROJECT
CONTAINER
DEPLOYMENT
WINDOW
HELP
siddharth.shekar@gmail.com
workspace / MDS_Class / main.js
PROJECT
MDS_Class
main.js
main.js
function main(){
console.log("Hello goorm!");
}
main();
TERMINAL
SEARCH
RESOURCE MONITOR
LINT
root@goorm:/workspace/MDS_Class#

Gooorm IDE interface

- You have your Project Directory on the left.
- Goorm automatically includes a main.js file in the main directory.
- The view on the right shows the content of the file.
- And at the bottom right is you Terminal.
- In the terminal type ls this will give the list of files in the current directory

```
root@goorm:/workspace/MDS_Class# ls
goorm.manifest  main.js
```

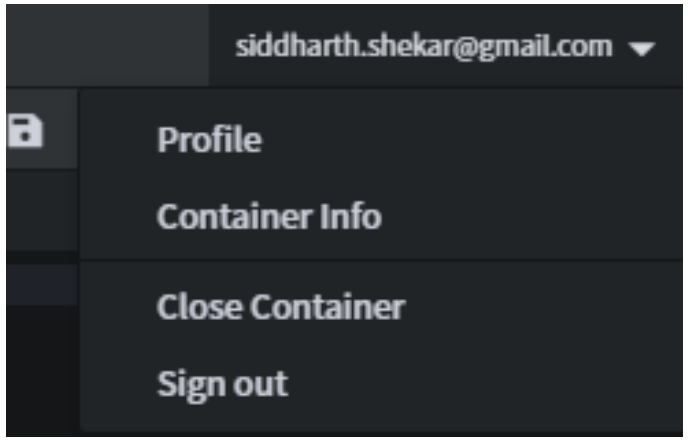
- To run the main.js file, in the terminal type node main.js to see the output printed to the console.

```
root@goorm:/workspace/MDS_Class# node main.js
Hello goorm!
```

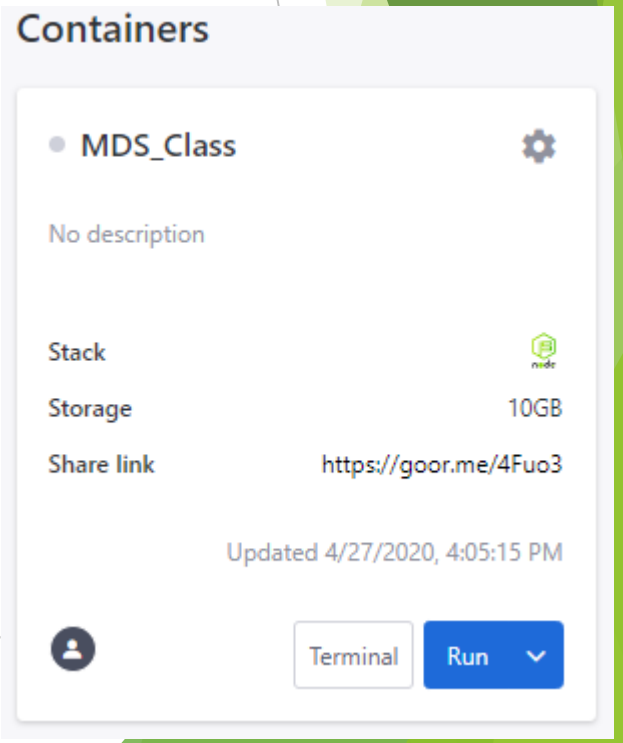
- And that's node js ☺

Goorm IDE interface

- Select your email id on top right and select Close Container to close.



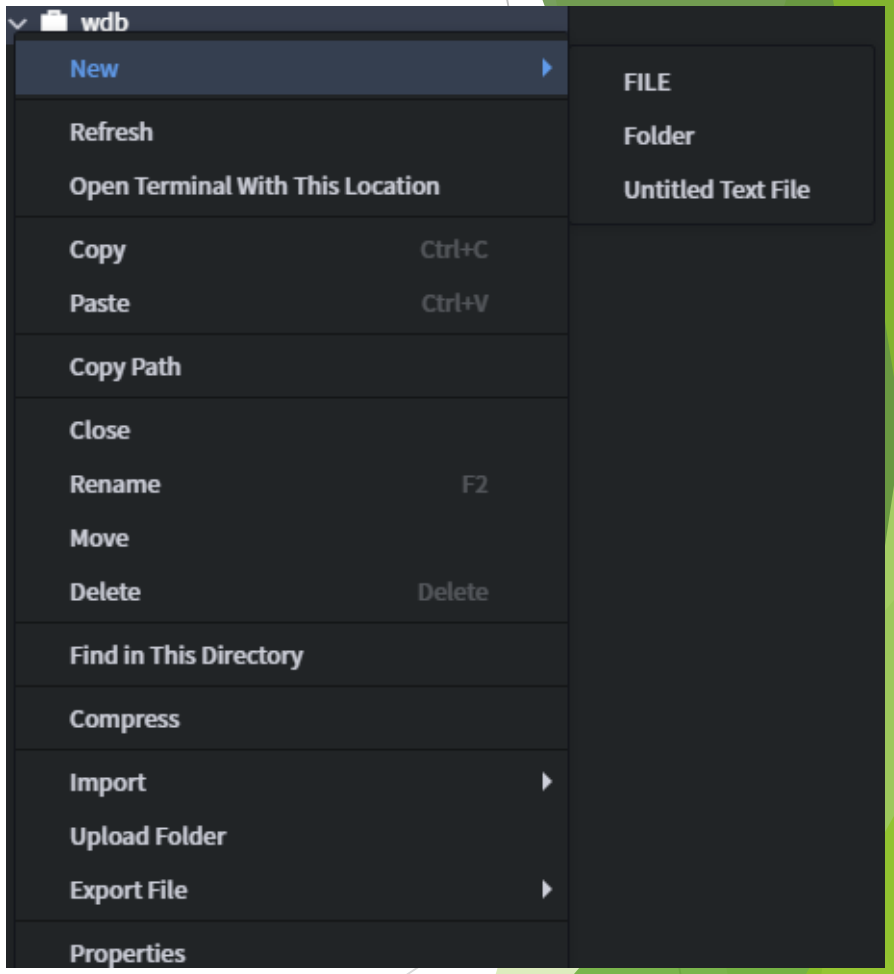
- Once closed it will take you back to the dashboard.
- Click on the Run to run the container again.



Command Line

Command Line – ls and mkdir

- Although you can add/ delete directory, file using by right clicking on directories in Goorm Projects panel.
- It is faster to do the same using Command Line in the Terminal.
- Go to terminal in Goorm and type *ls*
- This will list all the files in the directory
- To create a new directory type *mkdir* followed by the directory name.
- Call it webdev. Press Enter.
- Now a new directory is created under the *root* directory.

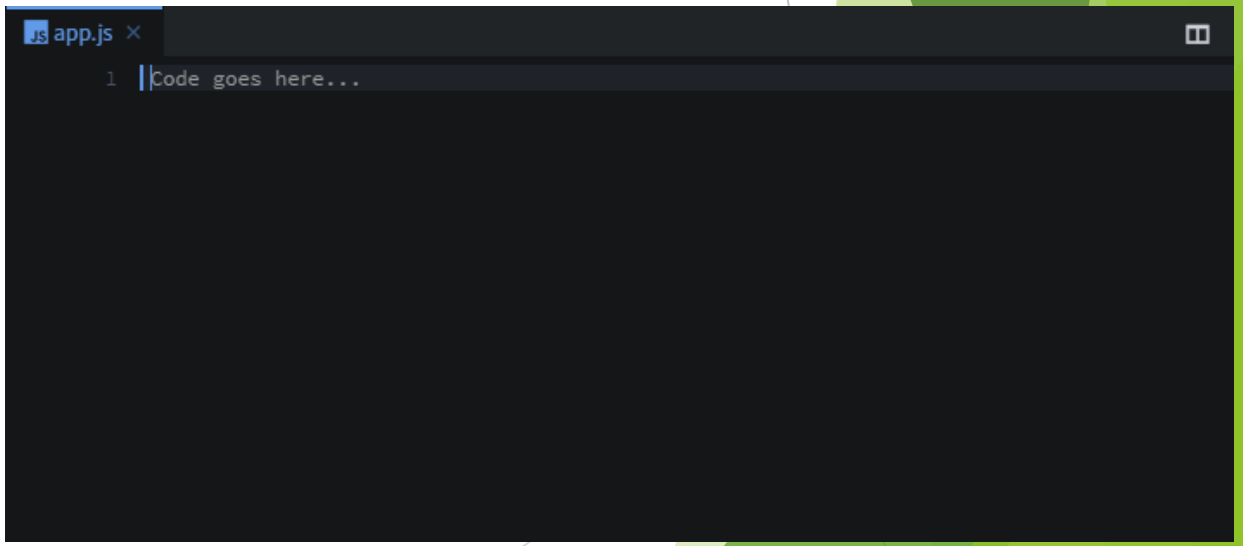



```
root@goorm:/workspace/wdb(master)# ls
APIs  Associations  AuthDemo  DeployDemo  databases  demoApp  firstExpressApp  goorm.manifest  intermediate
Express  mongod  myShop  nodeJS  packageJSNDemo  readme.md  yelpCamp
root@goorm:/workspace/wdb(master)# mkdir webdev
```

Command Line – cd & touch

- To get into the webdev directory type ***cd*** followed by the directory name. Press enter.
- We will be using the webdev directory for the remainder of the course for developing the website.
- Once inside the webdev directory, to create a new file type *touch* followed by the filename and extension and press enter.
- It is a standard to call the file *app.js*.
- This is file we the base file you will be using to develop the app.
- You can use cd space and two dots to got back to go back to the root directory.
- It is recommended to use the Project panel to delete or move directories.
- Double click on app.js to view the file

```
root@goorm:/workspace/wdb(master)# cd webdev
root@goorm:/workspace/wdb/webdev(master)# ls
root@goorm:/workspace/wdb/webdev(master)# touch app.js
```



NPM

NPM - Basics

- NPM – Is Node Package Manager.
- npm makes it easy for JavaScript developers to share and reuse code, and makes it easy to update the code that you're sharing, so you can build amazing things.
- npm is distributed with Node.js- which means that when you download Node.js, you automatically get npm installed on your computer.
- NPM is free and anyone can contribute a package.
- In the terminal type npm –v to see the npm version installed

- Documentation: https://docs.npmjs.com/
- Website: https://www.npmjs.com/

NPM - init

- Make sure you are in the webdev directory.
- Type *npm init*
- This will add a package.json file in the directory.
- This file will keep a track of all the dependency packages that are installed into our package.
- It will ask questions about things like pacjkage name, version, description, entry point, author, etc.
- For most things you can leave it as defult but make sure entry point is app.js.
- This will create a pakage.json file which will have all the details provided.

```
root@goorm:/workspace/wdb/webdev(master)# npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.
```

```
package name: (webdev)
version: (1.0.0)
description: MDS Webdev Lectures
entry point: (app.js)
test command:
git repository:
keywords:
author: Siddharth Shekar
license: (ISC)
```

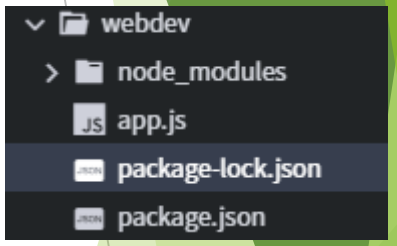
```
{
  "name": "webdev",
  "version": "1.0.0",
  "description": "MDS Webdev Lectures",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Siddharth Shekar",
  "license": "ISC"
}
```

NPM - install

- Next we will install our first package.
- The package is called express.
- Tin the terminal type

  npm install express –save
- This will install the new package called express.
- And the –save will make an entry into the package.json file with the package installed along with the version number of the package.
- The webdev directory will have a new node_modules directory within which all the packages installed will go into.
- The package-lock file will have all the installed packages dependencies.

```
root@goorm:/workspace/wdb/webdev(master)# npm install express --save
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN webdev@1.0.0 No repository field.

+ express@4.17.1
added 50 packages from 37 contributors and audited 126 packages in 14.001s
found 0 vulnerabilities
```




```
{
  "name": "webdev",
  "version": "1.0.0",
  "description": "MDS Webdev Lectures",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Siddharth Shekar",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1"
  }
}
```

Express

Express - app

- Express is a unoptimized, minimalist web framework for Node.js.
- Express is a web application framework that provides a robust set of features for web and mobile applications.
- https://expressjs.com/
- To include express in our application, at the top of the app.js file type
- var express = require(“express”)
- then var app = express();
- Now the app variable has all the functions of the express package.
- We will use in our web application.
- Run app.js to check if the express is installed and running without errors.

```
1  var express = require("express");
2  var app = express();
3
4  console.log("running the app.js file");
```

```
root@goorm:/workspace/wdb/webdev(master)# node app.js
running the app.js file
root@goorm:/workspace/wdb/webdev(master)#
```

Express – get route

- The get route method of express is used to show the a web page the get route is sending a request to.
- This is a get request to a route.
- The get route method takes 2 parameters.
- The first is the path to which the get function is pointing to.
- And the second is a callback function which takes two parameters request and response.
- Request is used to request something from this route.
- Response is used to send something to the route.
- Using the response.send() function we are sending HTML code to be rendered on the root route.
- Also known an the landing page.

- checkout
- copy
- delete
- get
- head
- lock
- merge
- mkactivity
- mkcol
- move
- m-search
- notify
- options
- patch
- post

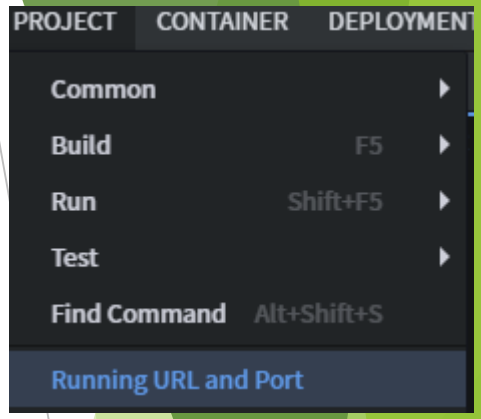
```
app.get("/", function(req, res){

    res.send("<h1>WELCOME TO THE HOMEPAGE!<h1>");


});
```

Express - Listen

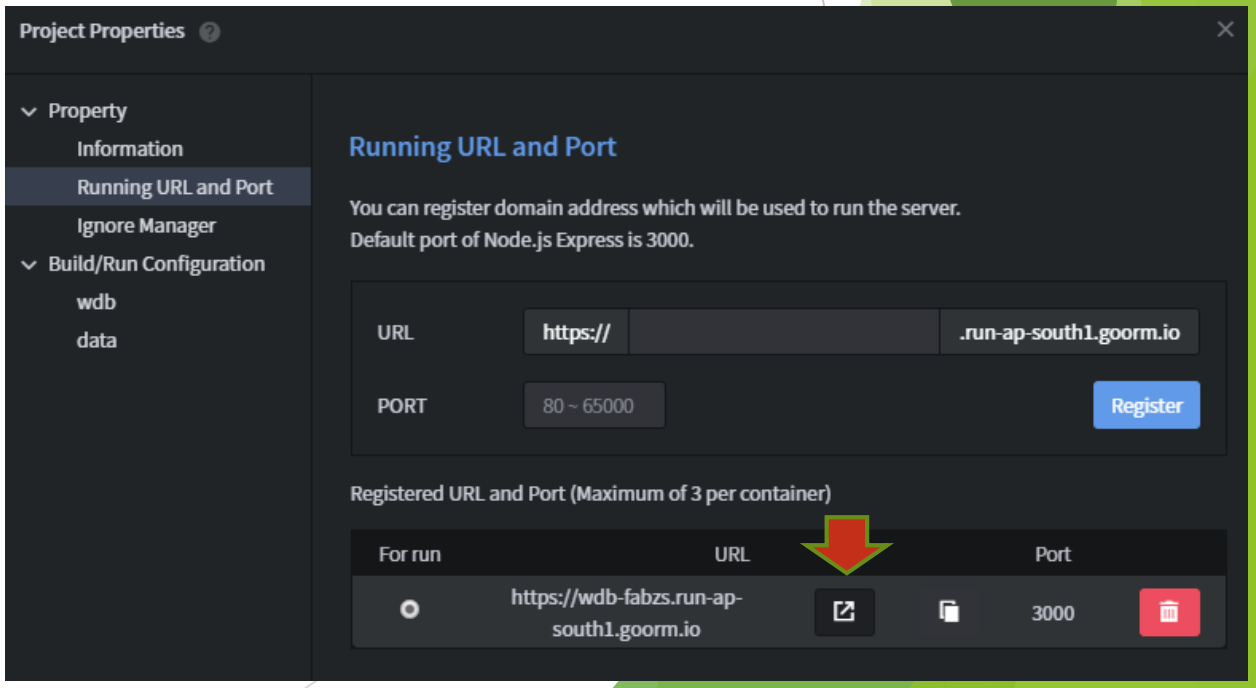
- Binds and listens for connections on the specified host and port.
- The method takes three parameters.
- The first is the port to connect to.
- The second is the host ip address.
- The third is a callback function.
- We will use the callback function to check if the server has started and the server is listening.

```
app.listen(process.env.port || "3000",
        process.env.ip,
        function(){

         console.log(" SERVER IS LISTENING")
         }
);
```

Viewing the page

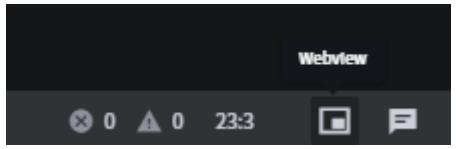


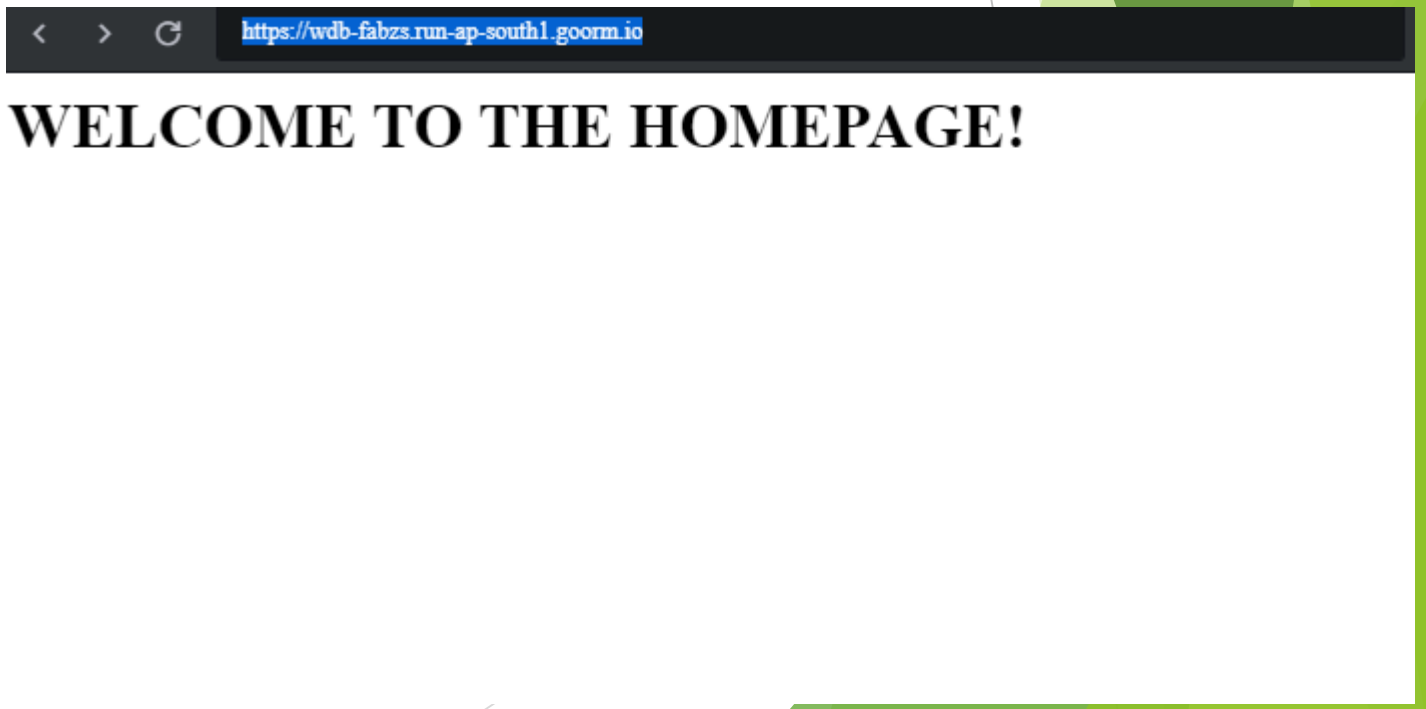
- Now run the application by running the app.js file
- You should see the server running now.
- To view the web page go to Project->Running URL and Port
- And click the link button next to the URL.
- This will open the page in a new tab on Chrome.
- Now you can preview your webpage



Viewing the page

- You can even copy the url and view the webpage in goorm webviewer.
- Click the webviewer icon on the bottom right of the ide which is next to the chat box below the terminal.
- Paste the URL link in the text box and press enter to view the preview.
- This way you don’t have to view the webpage in a separate tab.





EJS

- Now this is all looks good but we cannot be typing a whole lot of HTML into the send function when we want to render the page.
- Also we want the page to be dynamic which enables us to send data to the webpage.
- For dynamic pages we cant use HTML.
- So for that use EJS or Embeded Javasciprt Templating.
- ejs.co
- EJS is a simple templating language that lets you generate HTML markup with plain JavaScript.

EJS

- Installing EJS as a package
- If you are still running the server close it by typing ctrl+c in the terminal.
- Next make sure you are in the webdev directory.
- Type npm install ejs --save to install.
- Once installed you will also see in your package.json file ejs is added as a dependency.

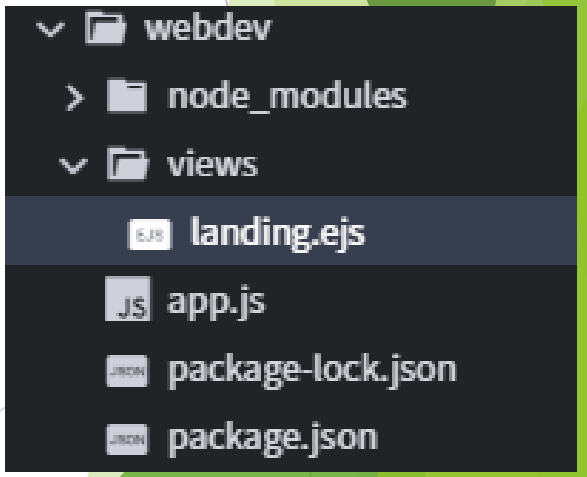
```
root@goorm:/workspace/wdb/webdev(master)# npm install ejs --save

> ejs@3.1.2 postinstall /workspace/wdb/webdev/node_modules/ejs
> node --harmony ./postinstall.js

Thank you for installing EJS: built with the Jake JavaScript build tool (https://jakejs.com/)

npm WARN webdev@1.0.0 No repository field.

+ ejs@3.1.2
added 15 packages from 8 contributors and audited 145 packages in 6.674s
found 0 vulnerabilities
```

```
{
  "name": "webdev",
  "version": "1.0.0",
  "description": "MDS Webdev Lectures",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Siddharth Shekar",
  "license": "ISC",
  "dependencies": {
    "ejs": "^3.1.2",
    "express": "^4.17.1"
  }
}
```

Creating the EJS file

- Webpages that are drawn by calling the res.render() function and in it we will be passing the page to be drawn.
- We will call this file landing.ejs.
- And express will automatically look for this file in the **views directory placed in the webdev directory.**
- Create a new directory in the webdev directory called views.
- And in the views directory create a new file called landing.ejs

```
root@goorm:/workspace/wdb/webdev(master)# ls
app.js  node_modules  package-lock.json  package.json
root@goorm:/workspace/wdb/webdev(master)# mkdir views
root@goorm:/workspace/wdb/webdev(master)# touch views/landing.ejs
```



EJS - Landing.ejs

- In the ejs file you can type your HTML code in it.
- Save the file by pressing ctrl + s

```
app.js   landing.ejs
1  <h1> THIS IS THE HOMEPAGE <h1>
2
3      <img src ="https://store.storeimages.cdn-
apple.com/8756/as-images.apple.com/is/iphone-11-pro-select-
2019-family?
wid=441&amp;hei=529&amp;fmt=jpeg&amp;qlt=95&amp;op_usm=0.5,0.
5&amp;.v=1567812930312"></img>
```

- In the app.get() function comment out the res.send() function.
- Replace it with res.render() and pass in the landing.ejs file in it in quotes.
- Start the server again by calling node app.js
- Preview the updated webpage with the new link to the image

```
app.get("/", function(req, res){

    //res.send("<h1>WELCOME TO THE HOMEPAGE!<h1>");

    res.render("landing.ejs");

});
```

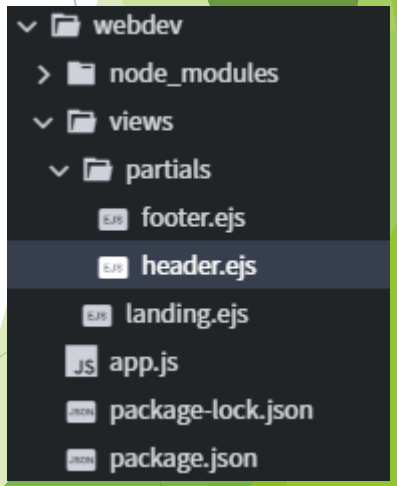
https://wdb-fabzs.run-ap-south1.goorm.io

THIS IS THE HOMEPAGE

EJS - Partials

- It is good we have the page working.
- But this HTML file doesn't have DOCTYPE, head tag or body tag.
- We could add it to all the ejs files, but if we will have a lot of ejs files we cant be typing it over and over.
- With partials we can have part of the HTML files written and included in our ejs files.
- These partial files will be created in the views/partials directory.
- Create a partials directory in views.
- Add 2 ejs files in it called header and footer.

```
app.js  node_modules  package-lock.json  package.json  views
root@goorm:/workspace/wdb/webdev(master)# mkdir views/partials
root@goorm:/workspace/wdb/webdev(master)# touch views/partials/header.ejs
root@goorm:/workspace/wdb/webdev(master)# touch views/partials/footer.ejs
```



EJS – Header.ejs

- In the header.ejs file add the code which includes the DOCTYPE, HTML, HEAD
- And leave the body and HTML tags and **don't** close it.
- We can also add a styling sheet in here which we will using.
- In the webdev directory **create a new directory called public and add a style.css** file in there.
- In the style.css file add a body color and background style.

```
<!DOCTYPE html>
<html>
<head>
    <title>Hello Web Dev</title>


<!-- we didnt speciy that the style.css is in public -->
<link rel="stylesheet" href="/style.css">


</head>
<body>
```

app.js | landing.ejs | header.ejs | style.css

```
body{
    background: yellow;
    color: purple;

}
```

EJS - Footer.ejs

- In the footer.ejs file add a trademark and close the body and html tags.
- In the landing.ejs file add the header.ejs and footer.ejs files at the top and bottom of the body of the page.
- In the app.js file we will also set the public directory as static so that we don't have todo pubic/style.css everytime.

```
<p>
    Trademark 2020
</p>

</body>
</html>
```

```
<%- include("partials/header.ejs")%>

<h1> THIS IS THE HOMEPAGE <h1>

    <img src ="https://store.storeimages.cdn-
images.apple.com/is/iphone-11-pro-select-2019
wid=441&amp;hei=529&amp;fmt=jpeg&amp;qlt=95&a
</img>


<%- include("partials/footer.ejs")%>
```

```
var express = require("express");
var app = express();

console.log("running the app.js file");

app.use(express.static("public"));
```

Preview

- Now run the server again and view that the Header partial is applied properly showing the correct title and applying proper style.
- We see the Footer getting applied as well with the Trademark included.

EJS - Sending data to EJS

- Data can be sent to a path with :/ after the path.
- Then the data can be retrieved using the req.params property.
- So if path is “/someData/:data” then the data can be retrieved by req.params.data and stored in a local var called data
- This data can be further sent to an ejs file in the res.render() function as below
- Res.render(“data.ejs”, {data: data}) ;
- Here the variable **data** will be sent as data into the site where you can access it there.
- It is common practice to have the same name for the variable being sent from the route and the variable received in the ejs file.

```
app.get("/someData/:data", function(req, res){

    var data = req.params.data;

    res.render("data.ejs", {data: data})


})
```

EJS - Sending data to EJS

- Create a new ejs file called data in the views directory.
- In the data.ejs file include the header and footer.
- Since we specified the variable to be received in the ejs file is called data, we use **data** in the ejs file to get the information stored in it.
- While writing JS code in HTML in the ejs file, the code needs to be enclosed in <%%>
- If the data needs to be received we use the = sign between <% and %>
- So we type <%= data%> to render the data in the ejs file.

app.js | landing.ejs | data.ejs

```
1  <%- include("partials/header.ejs")%>
2
3  <h1>
4      Received Data
5  </h1>
6
7  <%= data %>
8
9
10 <%- include("partials/footer.ejs")%>
```

EJS - If Else Statement

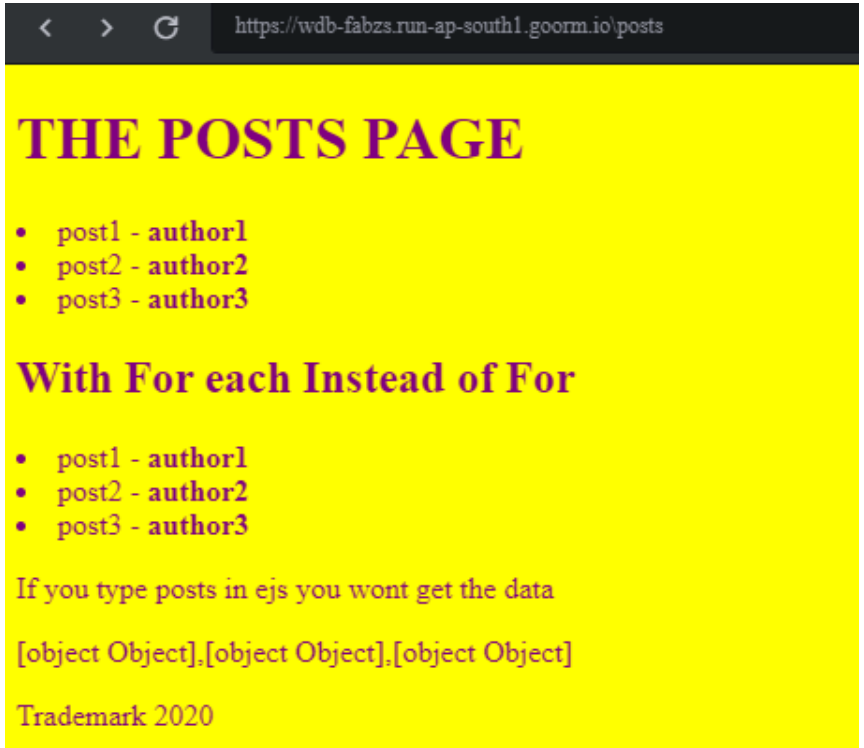
- You can type if else statements as well in ejs.
- For if statement you don't need to add = after <%.
- But all js code is encloded in <% and %>.


```
<%- include("partials/header.ejs")%>

<h1>
    Received Data
</h1>

<%= data %>

<p></p>
<p></p>

<%if(data === "iphone"){ %>

<p> GOOD CHOICE!</p>

<% } else { %>

<p> BAD CHOICE!</p>

<% } %>


<%- include("partials/footer.ejs")%>
```

EJS – Loops

- Suppose you have a get route called posts and you are sending an array of posts with object title and author to the posts.ejs file.
- Create a posts.ejs file in the views directory
- Run server and goto the /posts route to see output

```
app.get("/posts", function(req, res){

    var posts = [
        {title: "post1", author: "author1"},
        {title: "post2", author: "author2"},
        {title: "post3", author: "author3"}

    ];

    res.render("posts.ejs", {posts: posts});

});
```



app.js | landing.ejs | data.ejs | posts.ejs *

```
<%- include("partials/header.ejs")%>

<h1> THE POSTS PAGE  </h1>

<%for(var i=0; i < posts.length; i++){ %>

<li>
    <%=posts[i].title%> - <strong><%=posts[i].author%></strong>
</li>

<%} %>

<h2> With For each Instead of For </h2>

<%posts.forEach(function(post){%>

<li>
    <%=post.title%> - <strong><%=post.author%></strong>
</li>

<%})%>

<p>If you type posts in ejs you wont get the data</p>

<%=posts%>

<%- include("partials/footer.ejs")%>
```

Post Request

Post Request

```
var friends = ["Tony", "Miranda", "Justin"];

app.get("/friends", function(req,res){

    res.render("friends.ejs", {friends: friends});

});
```

- Till now we have only done GET request, where we request information from the server.
- Using the POST request we can send data back to the server.
- Suppose we have a GET friends route which renders friends.ejs file.
- We keep the friends array global so that we can access it later in the POST request function.

app.js | landing.ejs | data.ejs | posts.ejs | friends.ejs

```
1  <%- include("partials/header.ejs")%>
2
3  <h1>Friends List<h1>
4
5      <%friends.forEach(function(friend){%>
6      <li>My friend -   <%=friend%>   </li>
7      <%})%>
8
9  <%- include("partials/footer.ejs")%>
```

https://wdb-fabzs.run-ap-south1.goorm.io\friends

Friends List

- **My friend - Tony**
- **My friend - Miranda**
- **My friend - Justin**

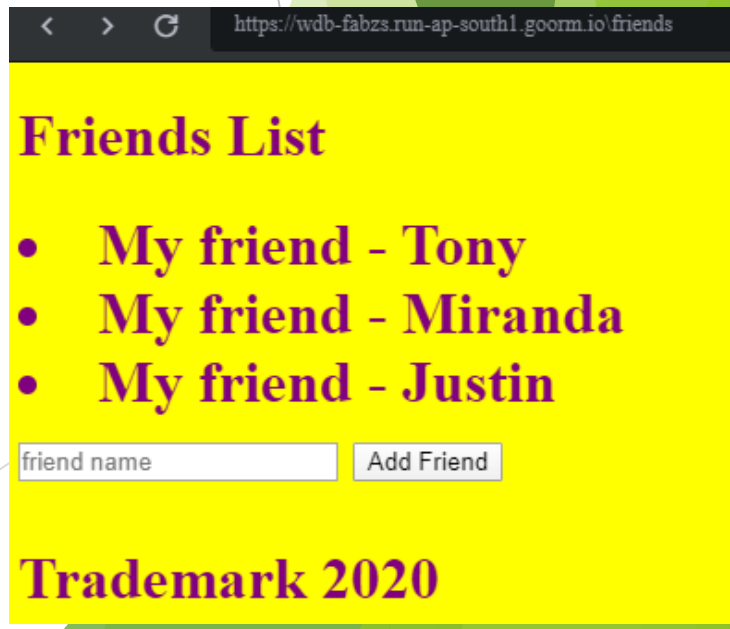
Trademark 2020

Post Request- Add friends form

- In the friends.ejs we add a form to add a new friend.
- The form has action and method attributes.
- The action tells once the submit button is pressed to which route the data should be sent to.
- The method specifies the route type which is POST.
- In the input we add a name attribute and set the value equal to **friend**.
- The name entered in the text box will be set as the value for the **friend.** We can then access the value stored for the **friend** in the post route.
- Finally we **Add Friend** button is added at the end of the form.

app.js | friends.ejs

```
1  <%- include("partials/header.ejs")%>
2
3  <h1>Friends List<h1>
4
5      <%friends.forEach(function(friend){%>
6      <li>My friend -    <%=friend%>    </li>
7      <%})%>
8
9      <form action="/friends" method="POST">
10         <input type="text" placeholder="friend name" name="friend"></input>
11         <button>Add Friend</button>
12     </form>
13
14
15 <%- include("partials/footer.ejs")%>
```



Post Request Route

- Now in the app.js file we will add the post route
- The POST request usually has the same path as the GET request. But instead of app.get() you are using app.post().
- This also takes a path and a callback function.
- In the callback function we console.log the req.body to get the body of the page.
- But when we initialize the server and go to friends page add a friend and click submit the console outputs as *undefined*.
- This is because the data is not understood.
- So we need a body-parser package to get the body information to be parsed correctly.

```
app.post("/friends", function(req,res){

    console.log(req.body)

})
```

```
running the app.js file
 SERVER IS LISTENING
undefined
```

Post Request - Body parser

- Add the body-parser package to the package
- npm install body-parser --save

```
root@goorm:/workspace/wdb/webdev(master)# npm install body-parser --save
npm WARN webdev@1.0.0 No repository field.

+ body-parser@1.19.0
updated 1 package and audited 177 packages in 4.357s
found 0 vulnerabilities
```

- In the app.js require body-parser and save it to a bodyParser variable.
- And also add **app.use(bodyParser.urlencoded({extended: true}));**
- As it is required.
- Now when you restart app.js and add a new friend you get the new friend name printed out in console instead of undefined.

```
var express = require("express");
var bodyParser = require("body-parser");

var app = express();

console.log("running the app.js file");

app.use(express.static("public"));
app.use(bodyParser.urlencoded({extended: true}));
```

```
root@goorm:/workspace/wdb/webdev(master)# node app.js
running the app.js file
 SERVER IS LISTENING
{ friend: 'newfriend' }
```

Post Request

- We can now get the friend value using the req.body.friend property.
- We store the value in a new variable called newFriend.
- Then we push the new name in the friends array.
- Finally we redirect back to the /friends GET route so the user can see the updated friends page.

- **Obviously when you restart the server the new friend will be lost.**

```
app.post("/friends", function(req,res){

    console.log(req.body.friend)

    var newFriend = req.body.friend;

    // new friend is added to the friends array
    friends.push(newFriend);

    // Redirect to the GET /friends route
    res.redirect("/friends");

})
```

```
root@goorm:/workspace/wdb/webdev(master)# node app.js
running the app.js file
 SERVER IS LISTENING
pammy
```

Exercise

- Create a Goorm login and create a new node JS and mongoDB container.
- Run javascript on it.

- Create a newShop project directory on Goorm
- Init npm.
- Install Express and create respective routes.
- Create a shop app by installing **faker** package and using the faker library populate 10 fake products and prices.
- Using express create a website to take the 10 products and display the names and prices on the website each time the application is run.