



# Practical 3

*Sprint I: Account SignUp*

*Prepared by Sonam Wangmo*

## OVERVIEW & PURPOSE

The purpose of this practical is to start with the sprint I. The practical will cover the basic operations of using and managing projects using asana tools and implementing the subtask of PBI of Account SignUp page.

xx



## Content

OVERVIEW & PURPOSE	1
Content	2
Preparation	3
Scrum: Start a Sprint	3
Scrum: Assigning Task	4
ERD: Database Design	7
Git: Pulling Changes	14
Scrum: Update subtask list	15
Setting up Express	17
MVC Express App	23
Persistence Storage for the application	25
Setting Up the Database	28
Setting Up MongoDB Atlas	29
Getting the Connection String for MongoDB Atlas	34
Setting Up Dependencies for MongoDB	35
Creating an User Model	37
Creating a New Feature Branch	37
Defining Schema	38
Creating Documents with Mongoose	39
Design UI for patient/care-giver Sign-Up and Login	42
Git: Pulling Changes	49
Appendix A: Set Up for Debugging	49
Setup Debugging For nodeJS	49
Starting the Debug Session	50
Port Forwarding	52



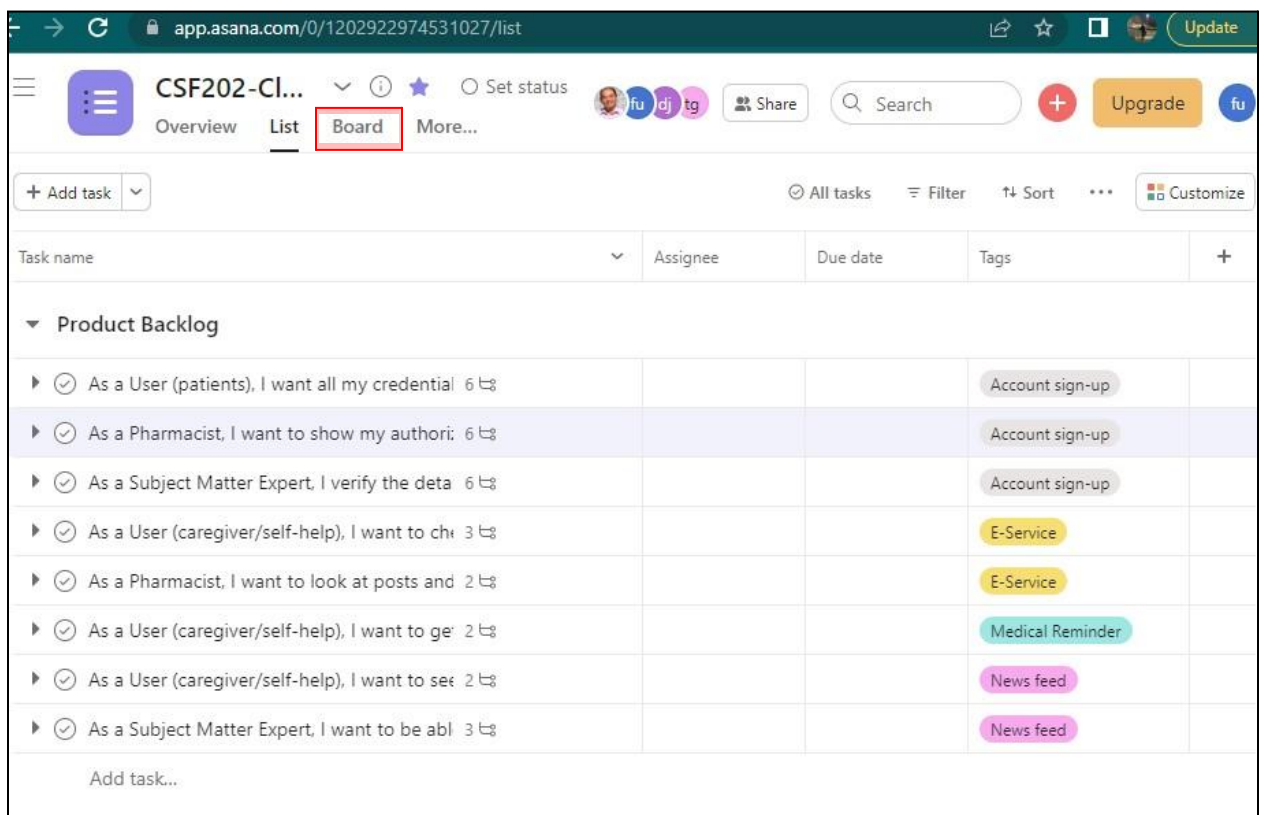
## Preparation

1. Make sure you have Visual Studio Code installed.
2. Run the docker container in the background.

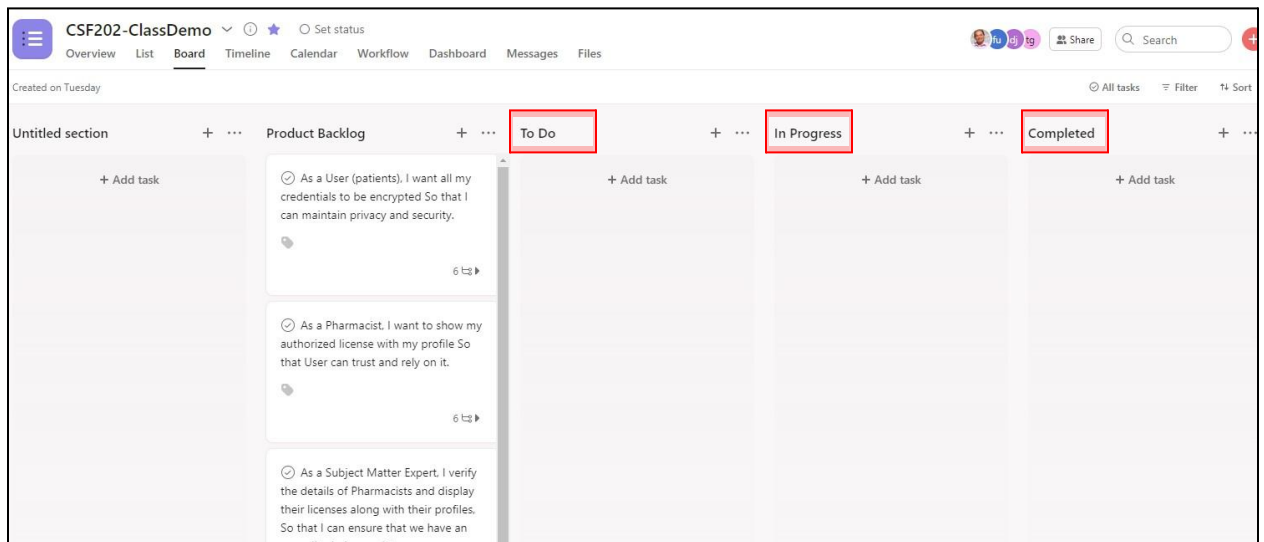
## Scrum: Start a Sprint

From this point onwards, we will start a sprint for our project. Assuming we have completed our *Sprint Planning Meeting*. Let's start a sprint by setting a new Milestone for our project.

1. Go to the **Board** section of the asana dashboard as shown.



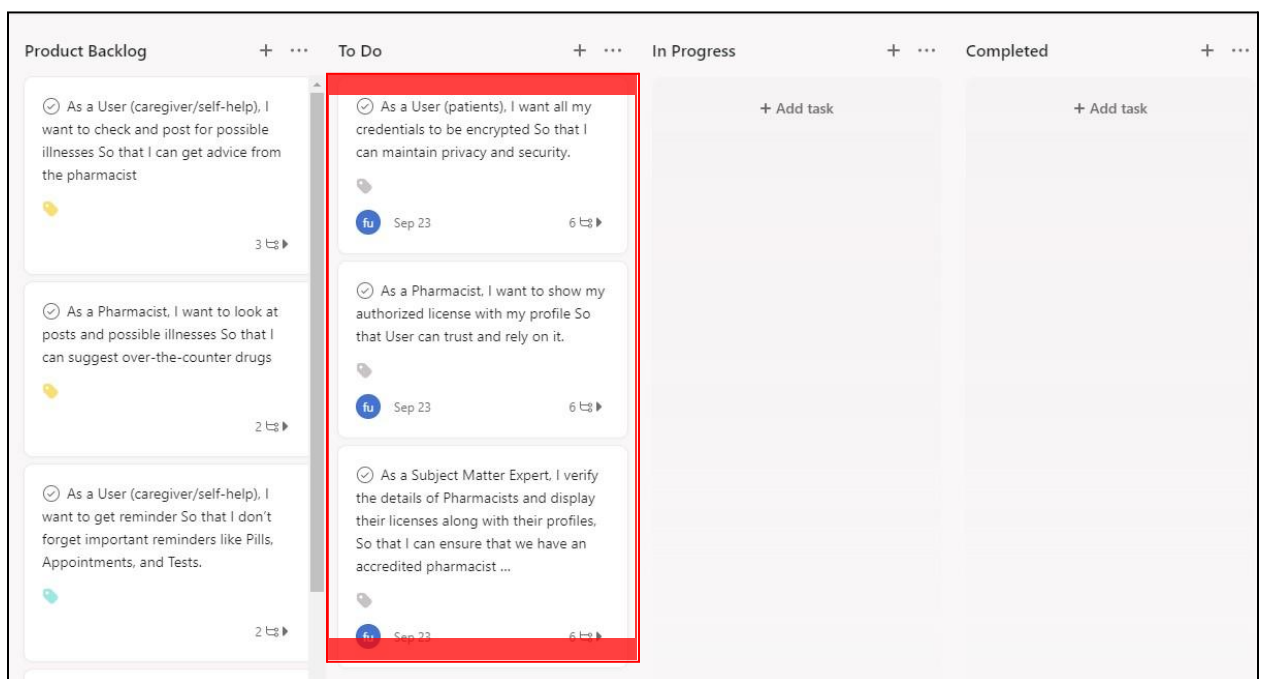
2. Add three sections namely *To Do*, *In Progress* and *Completed* after the Product Backlog section to track your progress.



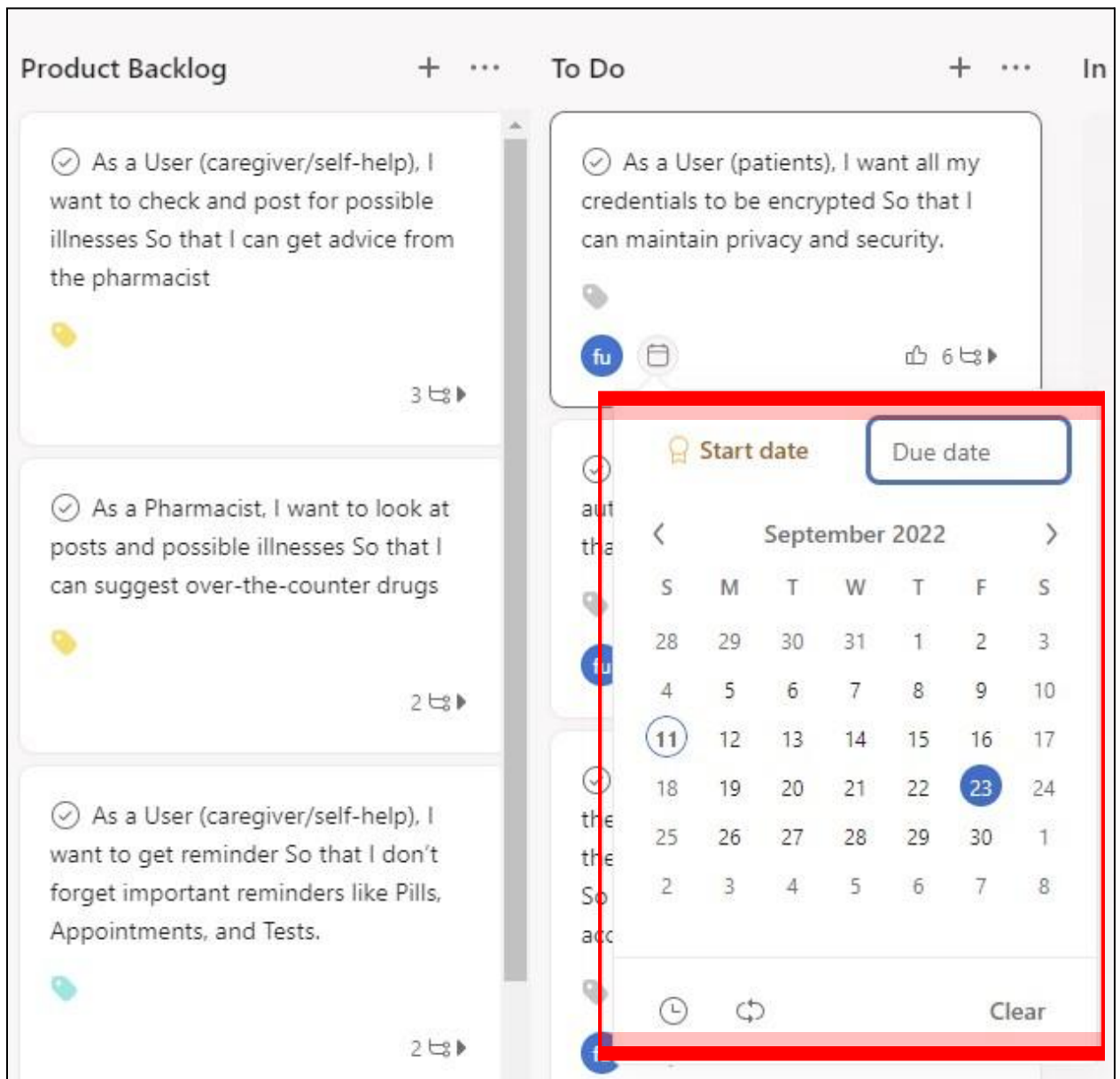
## Scrum: Assigning Task

After the sprint planning meeting, the team should have decided on a Sprint Backlog. The Sprint Backlog consists of all PBI to be completed for this sprint and everyone should have gotten the task assigned to them. In our exercise, you will be doing all the tasks.

3. Goto to the Scrum Board and move all the PBI listed in the Sprint Product Backlog from Product Backlog to *To DO* section as shown. Ensure that all the PBI has been assigned to you.



4. For each PBI set the due date for it as shown.



- Let's take the first PBI/user story 'As a User (patients), I want all my credentials to be encrypted, So that I can maintain privacy and security.' into the Progress list.

Next, start with the first subtask 'Create an entity relationship diagram of the web app'.

- Goto your Scrum Board and open up the first PBI and note individual subtasks.

Subtasks

☒ Create an entity relationship diagram of the web app.

☒ Design UI for patient/care-giver Sign-Up and Login

☒ Perform front-end form Validation

☒ Create User Models with roles

☒ Taking Data from forms and inserting them into respective tables

☒ Loading the stored data back into admin panels for verification

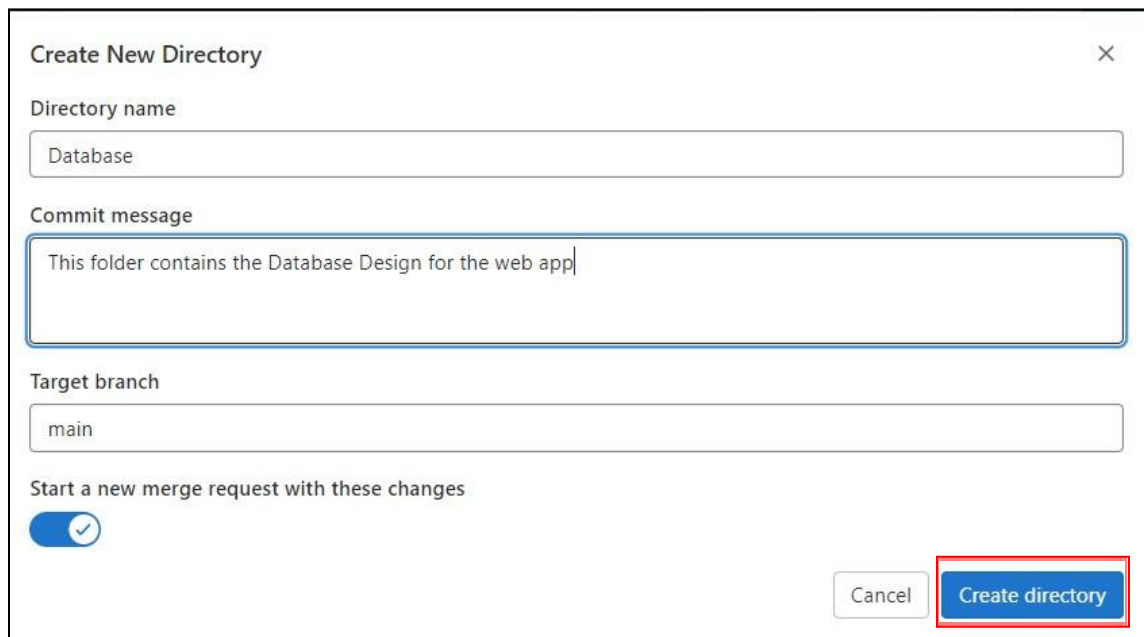
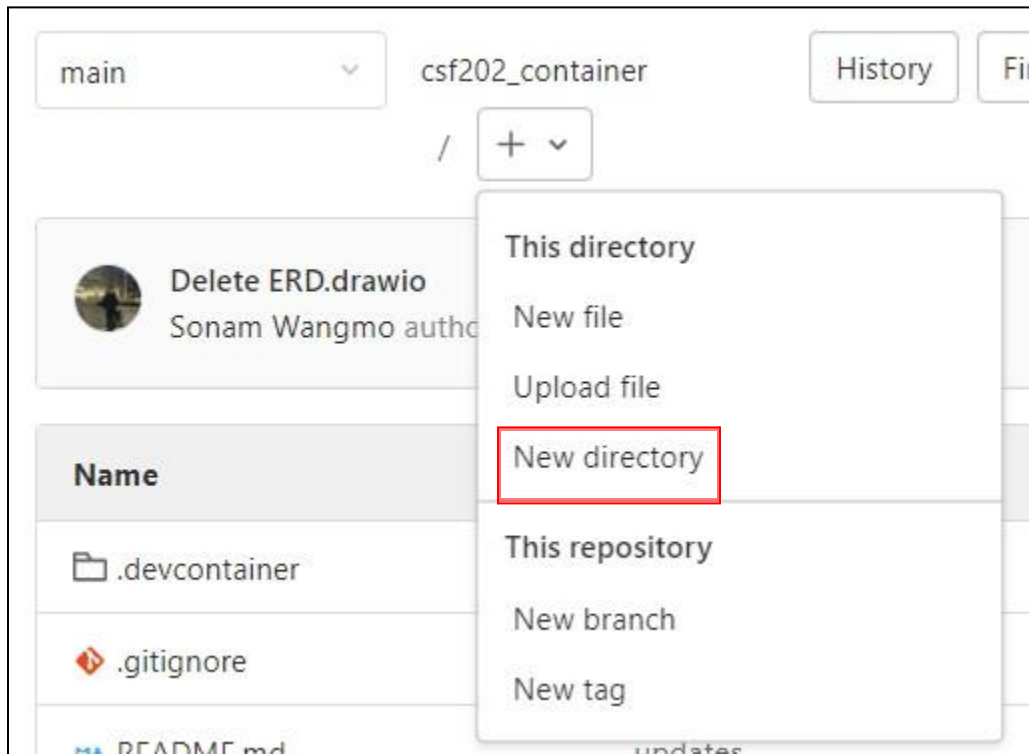
☒ Encrypt the credentials of all the users

+ Add subtask

### ERD: Database Design

In this section, we will be using an external online tool to create a simple ERD for our web application. This tool is called diagram.net and it will save directly into the GitLab repository.

7. Goto your Repository ►Files to create a new directory called “Database” with the fields as shown. We need to use this method to create empty folders so that they can be tracked by git.



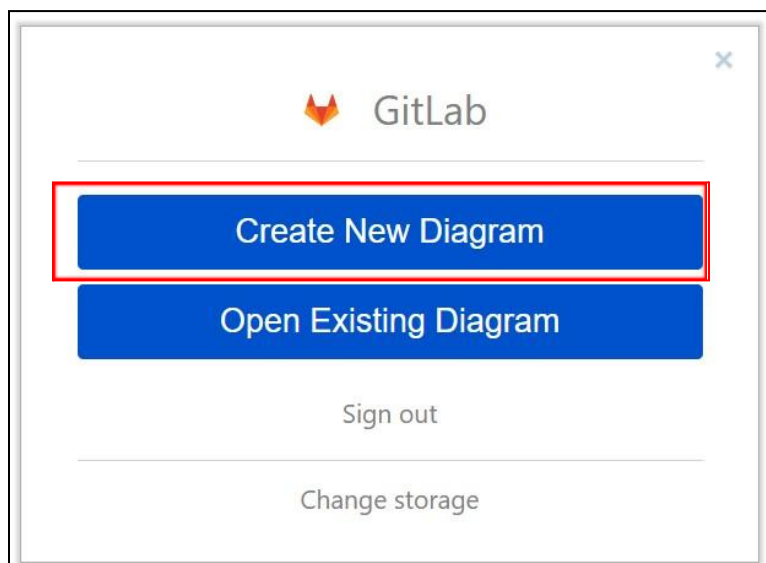
The screenshot shows a 'Create New Directory' dialog box. The 'Directory name' field contains 'Database'. The 'Commit message' field contains 'This folder contains the Database Design for the web app'. The 'Target branch' field contains 'main'. There is a checkbox labeled 'Start a new merge request with these changes' which is checked. The 'Create directory' button is highlighted with a red box.

8. Launch a new window in your browser and navigate to: <https://app.diagrams.net/>. It will show you the following:



Select GitLab

9. This brings you to the next section

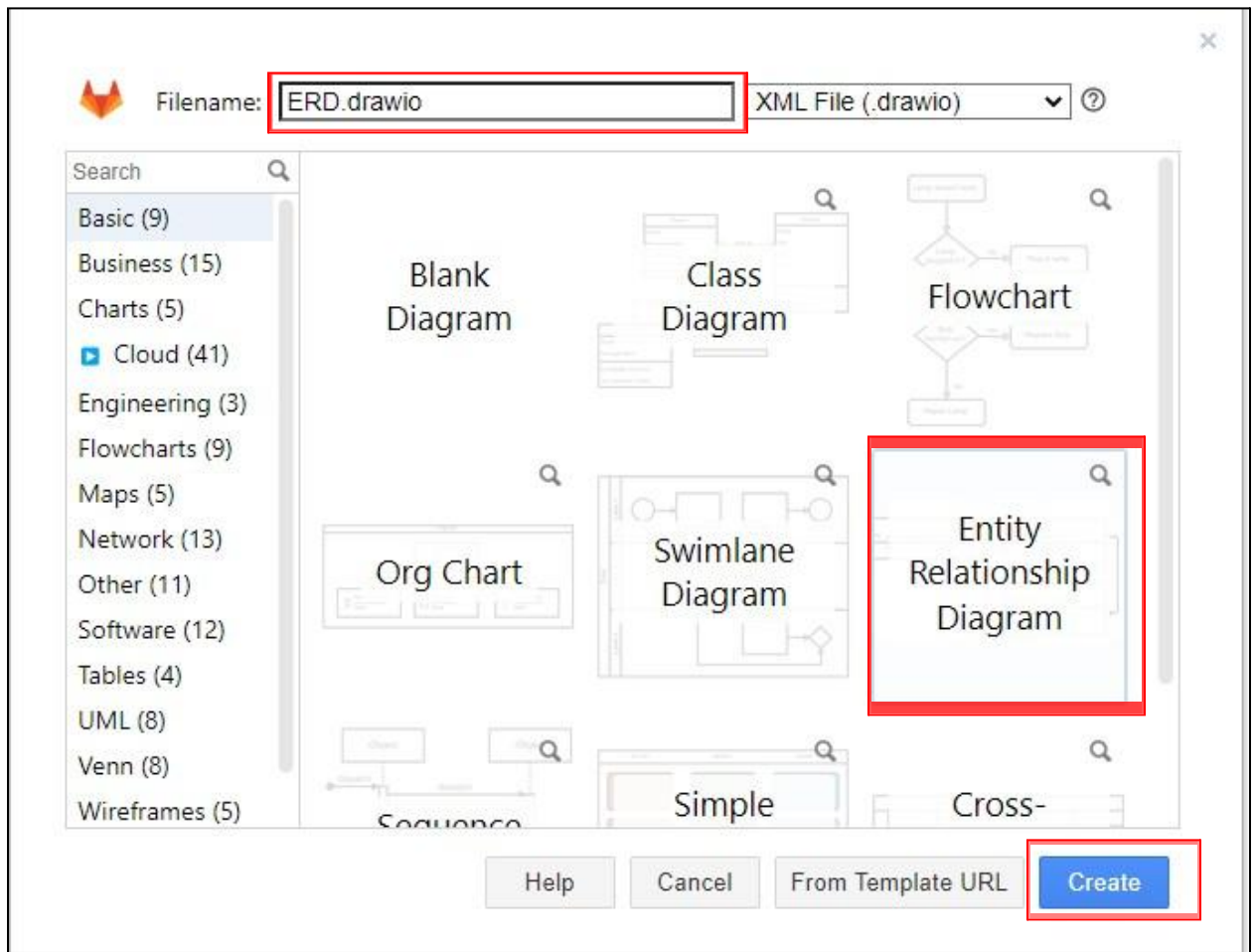


Select **Create New Diagram**

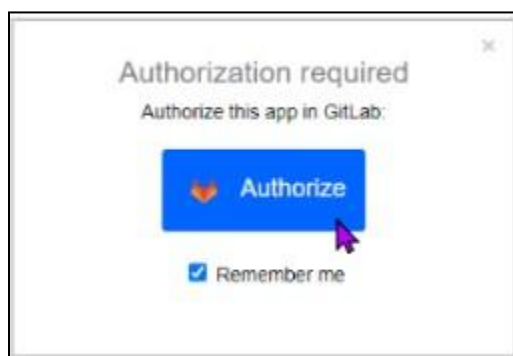
10. Here you will see more selection. Do the following:

- Give a name for the design, try starting with the word “ERD” so in GitLab it is easier to identify the file.
- Select “Database” category.
- Choose the template as shown below.
- Click Create for the new diagram






11. If this is the first time you are using GitLab for storing diagram.net, you will need to authorise it so that file can be saved in your GitLab.



12. Next, you need to sign into GitLab. Use your developer GMail (google account) to sign in as shown below.



# GitLab.com

Username or email

Password

☐ Remember me


[Forgot your password?](#)


[Sign in](#)

By signing in you accept the [Terms of Use](#) and acknowledge the [Privacy Policy](#) and [Cookie Policy](#).

Don't have an account yet? [Register now](#)

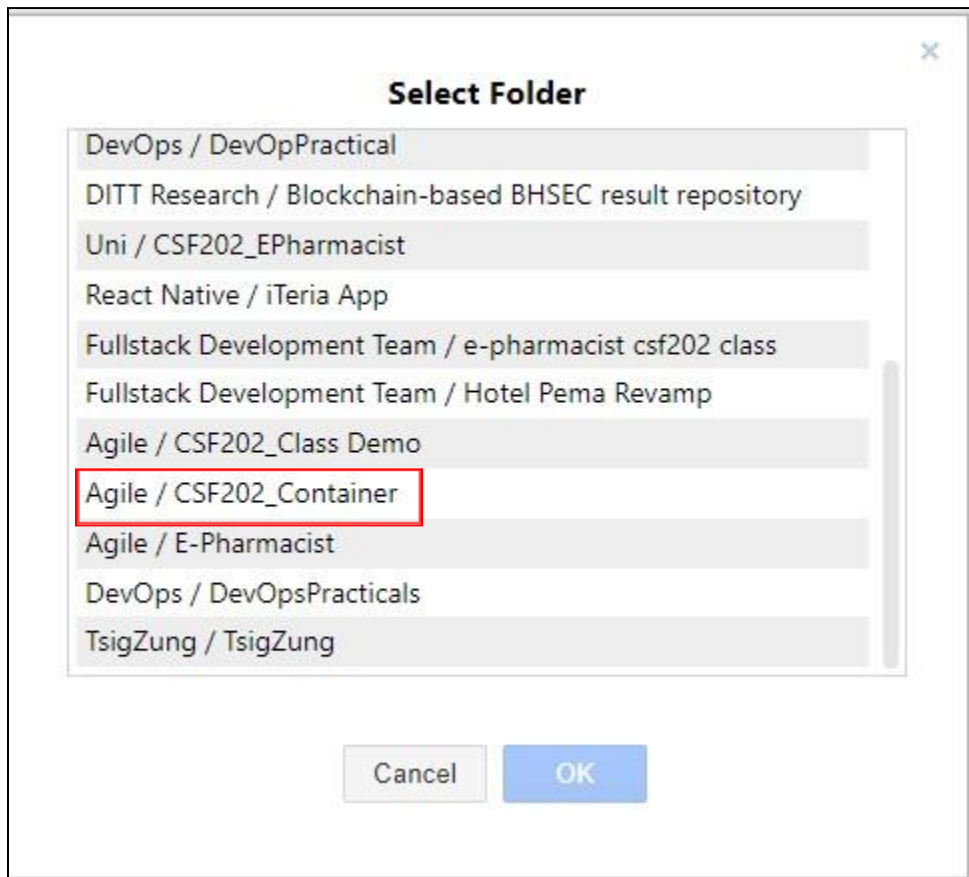
Sign in with

 Google

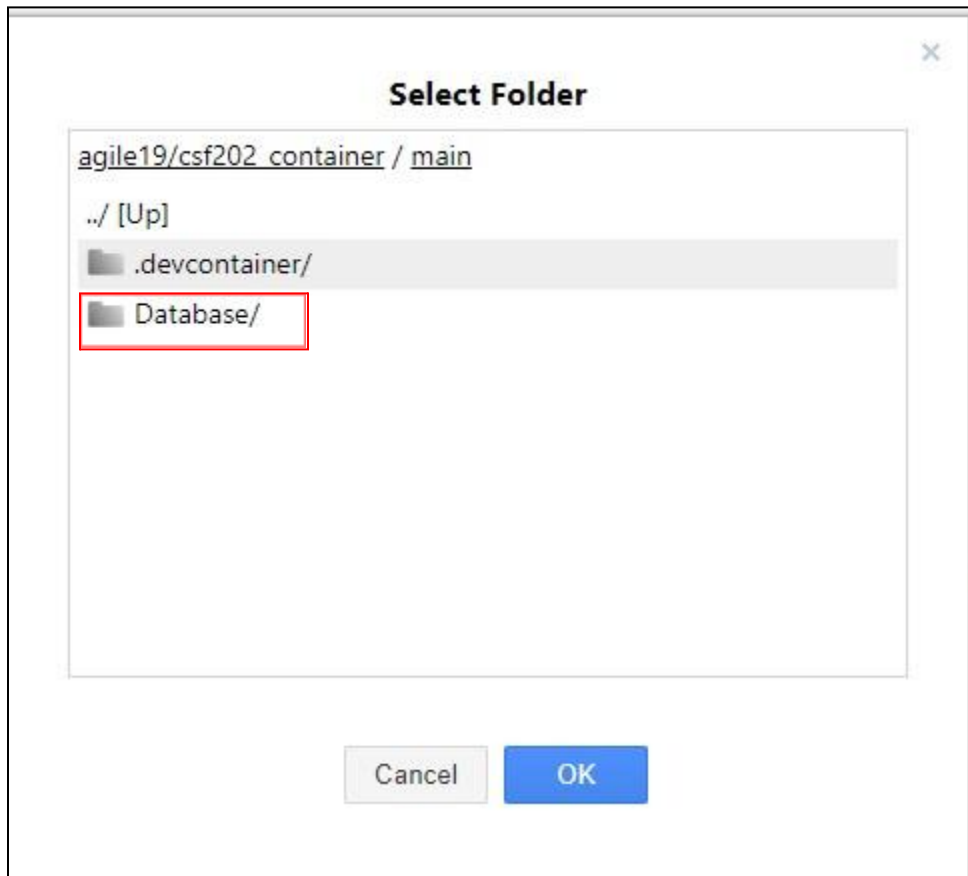
 GitHub

Enter your credential to proceed.

13. The next screen shows you the folder that you can save the file to.



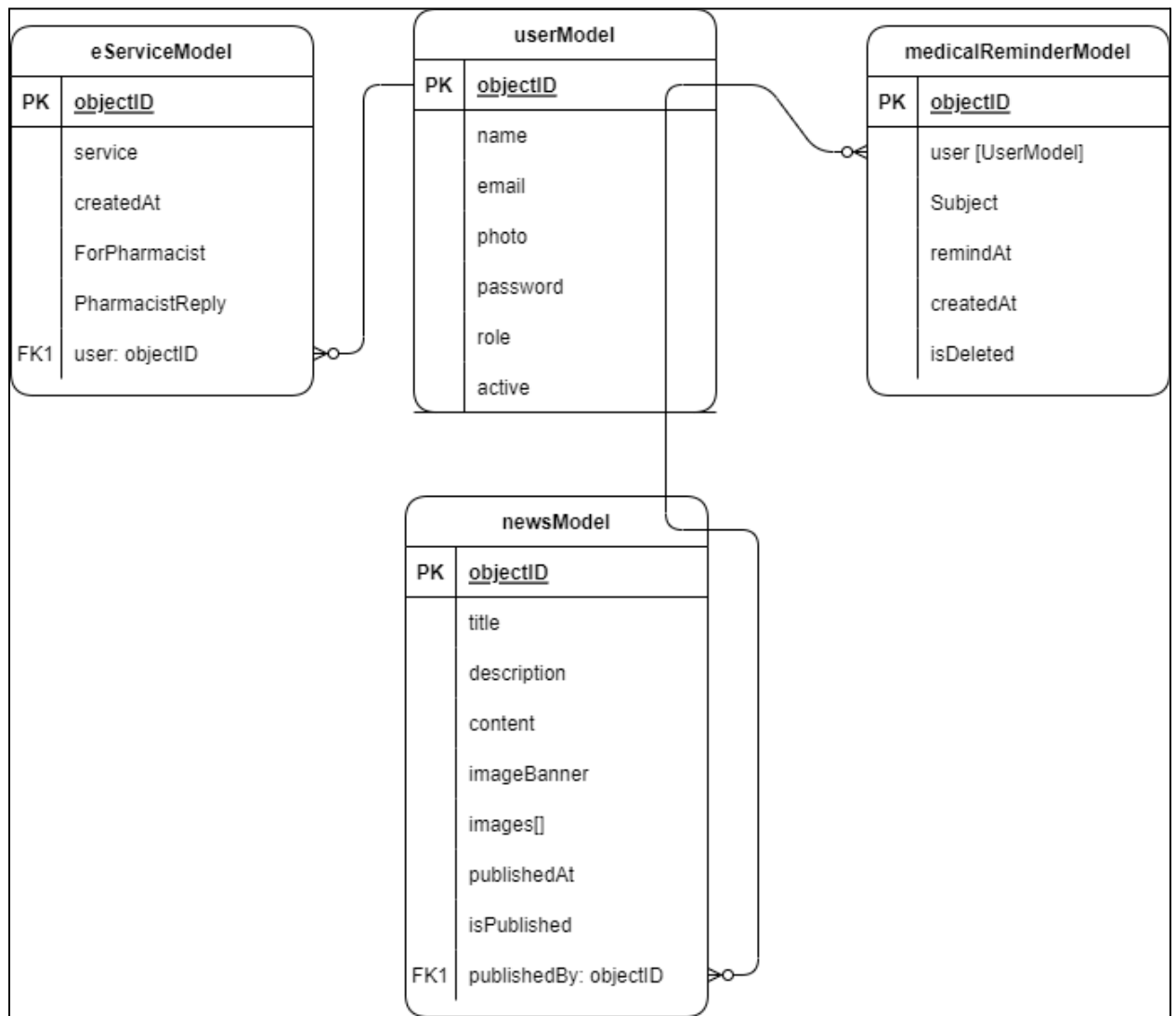
14. Select the current project and **Database** folder to proceed. Click OK



15. Accept default commit message. Click OK



16. Based on the template created, modify the ERD to the one shown below.



17. Once you have finished, click on Save and commit using the default message.  
Click OK



Commit Message:

Cancel OK

## Git: Pulling Changes

Git is a collaborative tool. When you are working with git, there is a high chance your team members or even yourself can add and commit files into the repository from another location.

Here, we have simulated such changes by creating the database file for the project from an external party. Note that our local repository has not been updated yet. Let's pull the changes from the origin repo (GitLab).

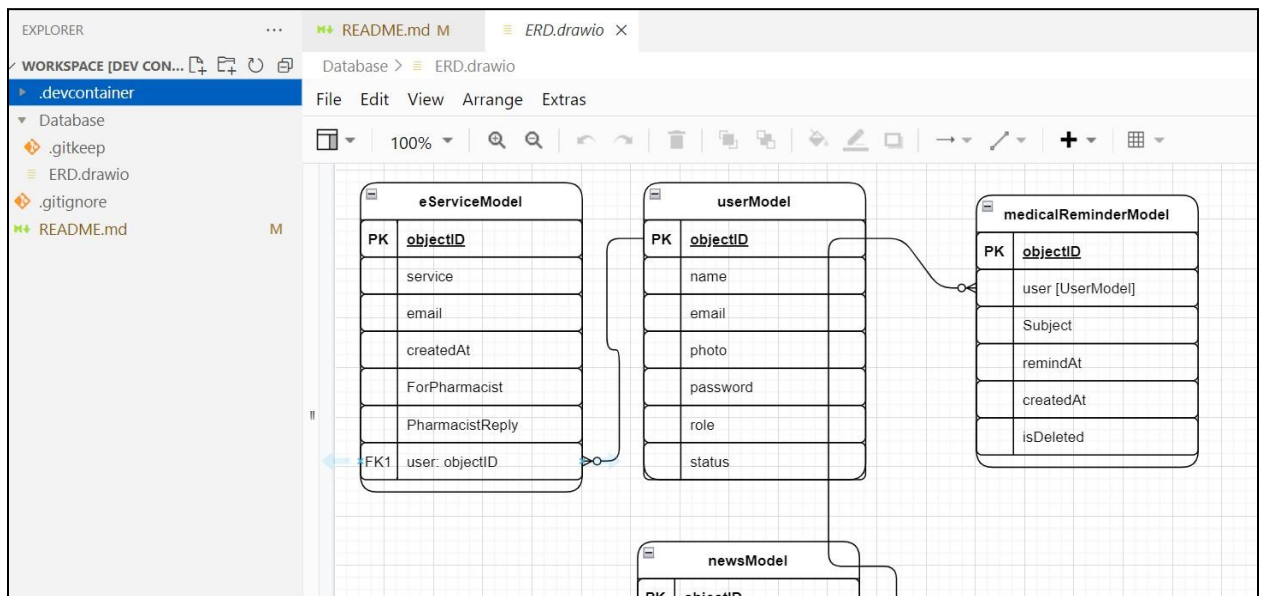
18. In your VSCode terminal, use the command:

```

▼ TERMINAL

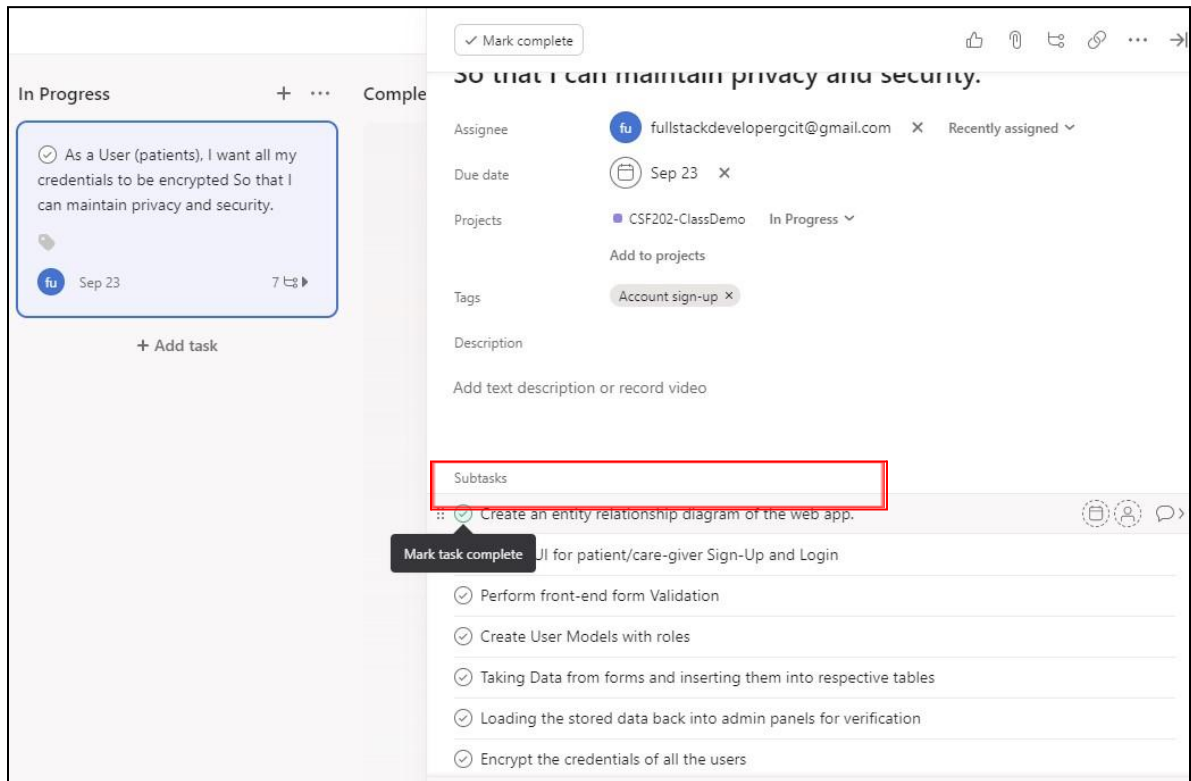
● node → /workspace (main) $ git pull
Updating e68b8b7..9ef9a07
Fast-forward
 Database/.gitkeep      |    0
 Database/ERD.drawio    | 417 ++++++
 2 files changed, 417 insertions(+)
 create mode 100644 Database/.gitkeep
 create mode 100644 Database/ERD.drawio
○ node → /workspace (main) $

```

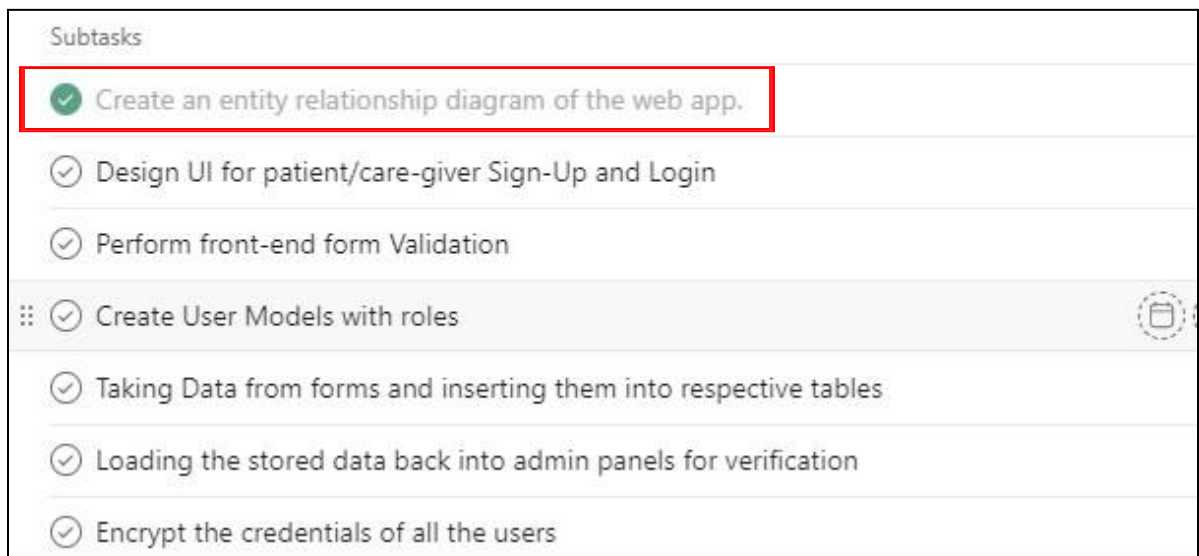


## Scrum: Update subtask list

19. Once you have completed this ERD task, go to the Scrum Board, open the PBI associated with this and mark the completion of the Database task as shown below:



The screenshot shows a Jira Scrum Board with a task in the 'In Progress' column. The task is titled 'As a User (patients), I want all my credentials to be encrypted So that I can maintain privacy and security.' The task is assigned to 'fullstackdeveloper@gmail.com' and has a due date of 'Sep 23'. The task is part of the 'CSF202-ClassDemo' project. The task is in the 'In Progress' column. The task is marked as 'Mark complete'.



The screenshot shows the 'Subtasks' section of the task. The first subtask, 'Create an entity relationship diagram of the web app.', is checked and highlighted with a red box. Other subtasks include 'Design UI for patient/care-giver Sign-Up and Login', 'Perform front-end form Validation', 'Create User Models with roles', 'Taking Data from forms and inserting them into respective tables', 'Loading the stored data back into admin panels for verification', and 'Encrypt the credentials of all the users'.

## ACTIVITY



20. Design a new database schema that meets your project requirements and commit as per the steps shown above in your Gitlab repository. Update your Scrum Board in asana when you are done.

## Setting up Express

The purpose of this section is to configure the setup of Express Js and its basic routing folders. *Express.js*, is a lightweight Node.js framework for creating web servers. Many other Node.js frameworks are built on top of Express.js. This tells you how convenient and powerful it is.

21. Before we begin, make sure you have **Node.js** installed on your computer. To check if you have Node.js installed run this command in your terminal `node -v`

A screenshot of a terminal window with three tabs: 'OUTPUT', 'TERMINAL', and 'PORTS'. The 'TERMINAL' tab is active. The terminal shows a prompt 'node → /workspace (main) \$' followed by the command 'node -v' which has been executed, resulting in the output 'v14.20.0'. The command 'node -v' is highlighted with a red box. Below the first prompt, there is another prompt 'node → /workspace (main) \$' with a cursor.

If you have Node.js installed it will return the version you have.

22. Next, run this command on the terminal `npm init`. After running this command you will have to answer some questions, for the most part the answers to these questions are not very important.



```

node →/workspace (main) $ 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.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (workspace) epharmacist
version: (1.0.0) 1
invalid version: 1
version: (1.0.0) 1.0.1
description: o ensure that the medicines prescribed to patients are suitable and advise patients about medicines, including how
to take them, what reactions may occur, and answer patients' questions.
entry point: (index.js) app.js
test command:
git repository: (git@gitlab.com:agile19/csf202_container.git)
keywords:
author: Ms. Sonam Wangmo
license: (ISC)
About to write to /workspace/package.json:

```

```

{
  "name": "epharmacist",
  "version": "1.0.1",
  "description": "o ensure that the medicines prescribed to patients are suitable and advise patients about medicines, including how to take them, what reactions may occur, and answer patients' questions.",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "git+ssh://git@gitlab.com:agile19/csf202_container.git"
  },
  "author": "Ms. Sonam Wangmo",
  "license": "ISC",
  "bugs": {
    "url": "https://gitlab.com/agile19/csf202_container/issues"
  },
  "homepage": "https://gitlab.com/agile19/csf202_container#readme"
}

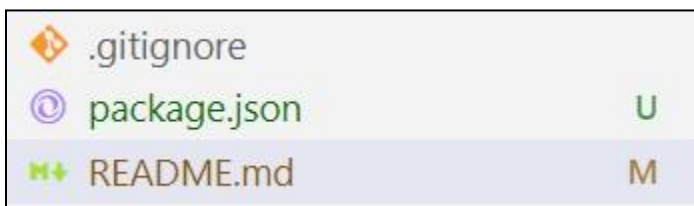
```

```

Is this OK? (yes) yes
node →/workspace (main X) $ 

```

23. After answering or skipping the questions a *package.json* file is going to be created. Below is an image of the file.





```
package.json U X
package.json > ...
1  {
2    "name": "epharmacist",
3    "version": "1.0.1",
4    "description": "o ensure that the medicines prescribed to patients are",
5    "main": "app.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "repository": {
10     "type": "git",
11     "url": "git+ssh://git@gitlab.com/agile19/csf202_container.git"
12   },
13   "author": "Ms. Sonam Wangmo",
14   "license": "ISC",
15   "bugs": {
16     "url": "https://gitlab.com/agile19/csf202_container/issues"
17   },
18   "homepage": "https://gitlab.com/agile19/csf202_container#readme"
19 }
```

*Note:* The `package.json` file holds some basic information about your application as well as *metadata* about your app. It will also manage all the dependencies of your application. Any additional package you install using *NPM* (Node Package Manager) will be managed here as well.

24. Let's install Express.js by running the following command in the terminal ***npm i express***.



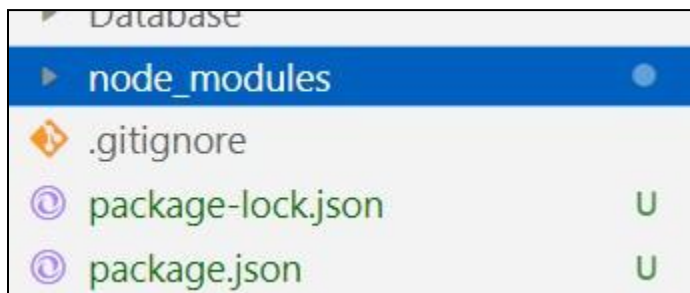
```
node → /workspace (main X) $ npm i express
npm notice created a lockfile as package-lock.json. You should commit this file.
+ express@4.18.1
added 57 packages from 42 contributors and audited 57 packages in 7.845s

7 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

```
New major version of npm available! 6.14.17 → 8.19.1
Changelog: https://github.com/npm/cli/releases/tag/v8.19.1
Run npm install -g npm to update!
```

25. This will install the Express.js framework, refresh and you will see a folder appear called **node\_modules**, do not edit this folder. This folder contains all the dependencies for *Express.js* and the dependencies of those dependencies.



26. In addition, we want to install another package called **nodemon** as development dependency, meaning that if our application ever goes into production this package will not be included. Run this command to install *nodemon* **npm i -D nodemon**.

```
✓ TERMINAL

node → /workspace (main X) $ npm i -D nodemon

> nodemon@2.0.19 postinstall /workspace/node_modules/nodemon
> node bin/postinstall || exit 0
```

The purpose of this package is to watch for any changes in our files and restart the server instead of us having to do that manually ourselves.

27. Now our *package.json* file looks like this:

```

package.json U X
package.json > ...
1  {
2    "name": "epharmacist",
3    "version": "1.0.1",
4    "description": "o ensure that the medicines prescribed to patients a
5    "main": "app.js",
6    "scripts": {
7      "start": "nodemon app.js"
8    },
9    "repository": {
10     "type": "git",
11     "url": "git+ssh://git@gitlab.com/agile19/csf202_container.git"
12   },
13   "author": "Ms. Sonam Wangmo",
14   "license": "ISC",
15   "bugs": {
16     "url": "https://gitlab.com/agile19/csf202_container/issues"
17   },
18   "homepage": "https://gitlab.com/agile19/csf202_container#readme",
19   "dependencies": {
20     "express": "^4.18.1"
21   },
22   "devDependencies": {
23     "nodemon": "^2.0.19"
24   }
25 }
26

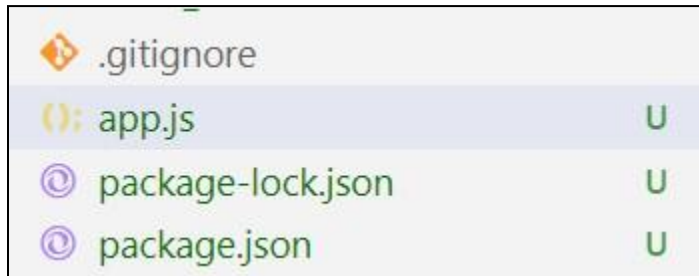
```

On *line 7*, We added the script *start* which will use our *nodemon* package to watch our *app.js* file. Therefore when we run the command *npm start* in the terminal this will start our nodemon package.

*Lines 19–21* shows all the dependencies we have installed, in this case *express*.

Then **lines 22–24** are the development dependencies in our case that point to *nodemon*.

28. Now that we have all the packages we need let's start building our express server. Create a **new file** and call it ***app.js***, this is going to be the entry point for our application and also where in our case we are going to write the bulk of our code.



29. Implement the code below in *app.js* file. In Node.js, *app.js* file is a javascript file that contains the express app object.



*Line 1:* We are bringing in the express framework and storing it in a constant.

*Line 3:* We are initializing the express framework and saving it into another constant.

*Line 6:* We are saving the port number of our server as a constant. In our



case we are going to use port number 4001.

*Line 7:* We use the built-in listen method and it expects at least one argument, which is the port *number* that our is located. You can pass a callback function as a second argument to the listen method, in this case, our function prints a log to the console indicating that the server is running and showing which port is used.

30. Now if you run **npm start** in your terminal you should see the message from *line* 7, that your server is running and the port number in your terminal. Alternatively, if you have set up the debugging environment, you can also press F5 to start the process.

A screenshot of a terminal window with tabs for OUTPUT, TERMINAL, and PORTS. The TERMINAL tab is active. The terminal shows the command 'npm start' being executed, which runs 'nodemon app.js'. The output shows 'nodemon' version 2.0.19, instructions to restart with 'rs', and the server starting on port 4001. It also shows a restart triggered by changes.

```
node → /workspace (main X) $ npm start

> epharmacist@1.0.1 start /workspace
> nodemon app.js

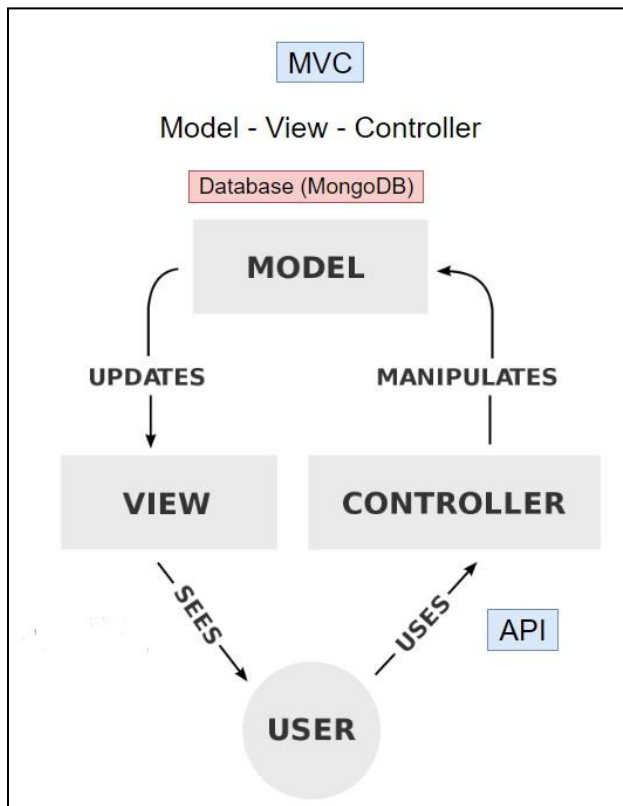
[nodemon] 2.0.19
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
App running on port 4001 ..
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
App running on port 4001 ..
```

At this moment you have a working express server.

## MVC Express App

The purpose of this section is to build Web Servers that focus on reliability and making the development process much simpler and easier since MVC composes the Web Server into three separate parts.



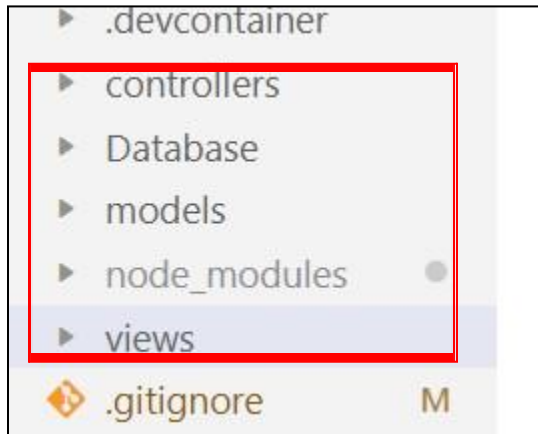


- ★ MVC is simply a design or architectural pattern used in software engineering. While this isn't a hard rule, but this pattern helps developers focus on a particular aspect of their application, one step at a time.
- ★ The main goal of MVC is to split large applications into specific sections that have their own individual purpose.
- ★ It also allows developers to logically structure applications in a secure way, which we will show in this tutorial. But first, let's break down what each aspect of the pattern provides.

31. *Model:* As the name implies, a model is a design or structure. In the case of MVC, the model determines how a database is structured, defining a section of the application that interacts with the database. This is where we will define the properties of a user that will be stored in our database. The controller accesses the database through the model. You could say that the model is the heart of the application.
32. *View:* The view is where end users interact within the application. Simply put, this is where all the HTML template files go.
33. *Controller:* The controller interacts with the model and serves the response and functionality to the view. When an end user makes a request, it's sent to the

controller which interacts with the database. You can think of the controller as a waiter in a restaurant that handles customers' orders, which in this case is the *view*. The waiter then goes to the kitchen, which is the *model/database*, and gets food to serve the customers, which is the *controller* handling the request.

34. Now create three folders to represent MVC: *models*, *views*, and *controllers*.

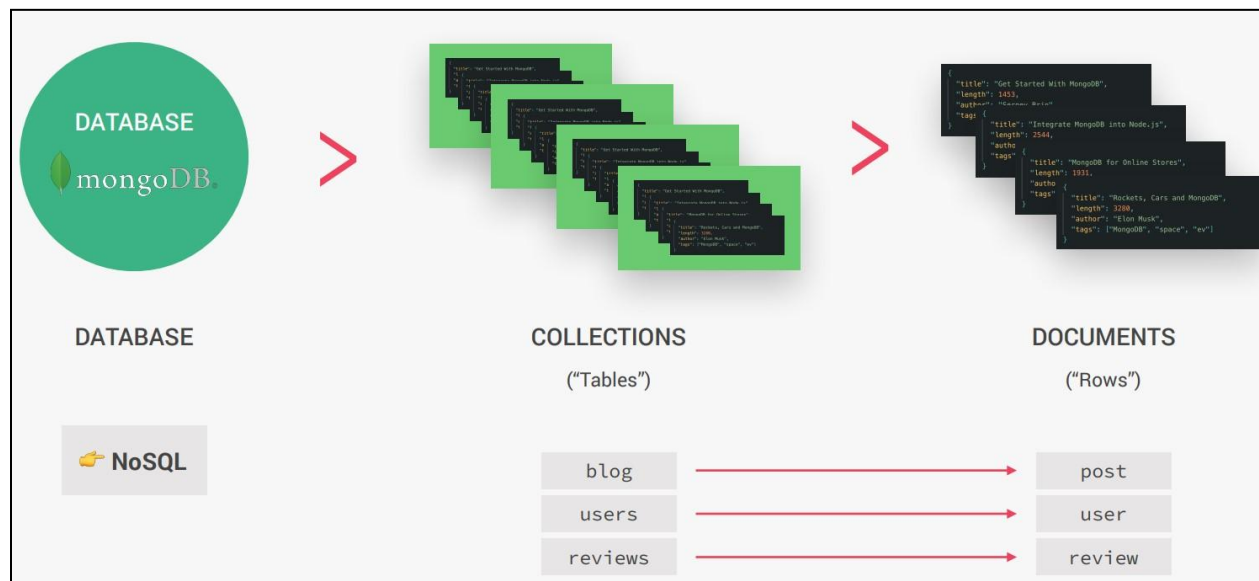


Congratulations!! We are now set to do coding for the e-pharmacist web application.

## Persistence Storage for the application

For database storage, we will be using **mongodb**.

*mongodb: an overview*





## What is mongodb?

### MONGODB

"MongoDB is a document database with the scalability and flexibility that you want with the querying and indexing that you need"

#### KEY MONGODB FEATURES:



- 👉 **Document based:** MongoDB stores data in documents (field-value pair data structures, NoSQL);
- 👉 **Scalable:** Very easy to distribute data across multiple machines as your users and amount of data grows;
- 👉 **Flexible:** No document data schema required, so each document can have different number and type of fields;
- 👉 **Performant:** Embedded data models, indexing, sharding, flexible documents, native duplication, etc.
- 👉 Free and open-source, published under the SSPL License.

## Difference between document structure and relational database:

### DOCUMENT STRUCTURE

👉 **BSON:** Data format MongoDB uses for data storage. Like JSON, **but typed**. So MongoDB documents are typed.

Unique ID →

Fields →

Embedded documents →

```
{
  "_id": ObjectId('9375209372634926'),
  "title": "Rockets, Cars and MongoDB",
  "author": "Elon Musk",
  "length": 3280,
  "published": true,
  "tags": ["MongoDB", "space", "ev"],
  "comments": [
    { "author": "Jonas", "text": "Interesting stuff!" },
    { "author": "Bill", "text": "How did oyu do it?" },
    { "author": "Jeff", "text": "My rockets are better" }
  ]
}
```

Values (typed)

👉 **Embedding/Denormalizing:** Including related data into a single document. This allows for quicker access and easier data models (it's not always the best solution though).

### RELATIONAL DATABASE

Column

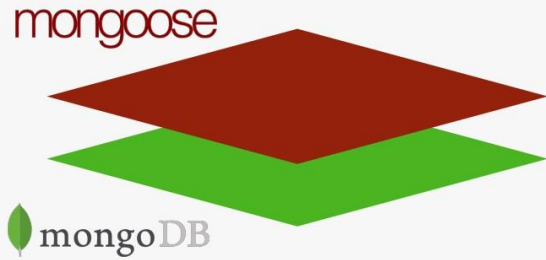
id	title	author	length	published	tags	comments
1	Rockets...	Elon Musk	3280	TRUE	-	-

"JOIN tables"  
Reference by  
comments\_id

id	autor	text
1	Jonas	Interesting stuff!
2	Bill	How do you do it?
3	Jeff	My rockets are better

👉 **Data is always normalized**

## We will be using mongoose. What is mongoose and why use it?



- ✚ Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js, a higher level of abstraction;
- ✚ Mongoose allows for rapid and simple development of mongoDB database interactions;
- ✚ Features: schemas to model data and relationships, easy data validation, simple query API, middleware, etc;
- ✚ **Mongoose schema:** where we model our data, by describing the structure of the data, default values, and validation;
- ✚ **Mongoose model:** a wrapper for the schema, providing an interface to the database for CRUD operations.

SCHEMA
➔
MODEL

35. Next, we will **create a *server.js* file** because it is a good practice to have everything that is related to express in one file and everything that is related to the server in another main file. Hence, our *server.js* file will be our starting file where everything starts and it's there when we listen to our server.



36. Take the server listen code from *app.js* to *server.js*. To use the app, export the app from *app.js* and import using require module in *server.js* as shown:

```

() app.js U X
() app.js > [?] <unknown>
1  const express = require("express")
2
3  const app = express()
4
5  module.exports = app

```



```

() server.js U X
() server.js > ...
1  const app = require('./app')
2  /* Starting the server on port 4001. */
3  const port = 4001
4  app.listen(port, () => {
5    console.log(`App running on port ${port} ..`)
6  })

```

Since, our starting point is *server.js* file, change the code in the *package.json* as shown:

```

() package.json U X
() package.json > {} scripts > start
1  {
2    "name": "epharmacist",
3    "version": "1.0.1",
4    "description": "o ensure that the medicines
5    "main": "app.js",
6    "scripts": {
7      "start": "nodemon server.js"
8    },

```

after the code change, the server still works:

```

o node → /workspace (main X) $ npm start

> epharmacist@1.0.1 start /workspace
> nodemon server.js

[nodemon] 2.0.19
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
App running on port 4001 ..

```

## Setting Up the Database



There are 2 ways in which we can host our MongoDB database:

1. **Local MongoDB database:** One of the containers that we have created earlier in the [setup environment](#), enables us to use this database even when the internet is not available.
2. **MongoDB Atlas:** This is a cloud based database that is a free tier. Your application is able to connect it from anywhere since it is residing in the cloud. You can use this for production deployment. The downside of using this in development is that it requires constant internet connectivity.

We will set up our database connectivity for both MongoDB Atlas as well as the Local one.

### Setting Up MongoDB Atlas

37. Setup with Atlas: To host our database, we'll go to MongoDB Atlas, register <https://www.mongodb.com/cloud/atlas/register> ] if we haven't already, create a new project if we want, and create a new cluster. When we've done this, Atlas will display a 5-item "Get Started" checklist, but we'll shelve that for now and go at our own pace. Fill up the details or register using your developer gmail account.



## Sign up


See what Atlas is capable of for free

ⓘ

☐ I agree to the [Terms of Service](#) and [Privacy Policy](#).

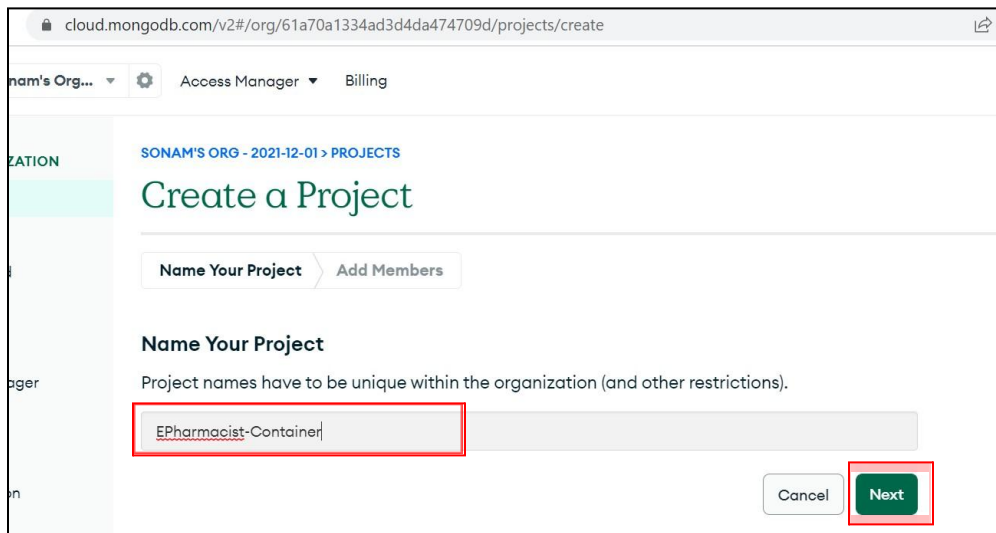
Create your Atlas account

or

 Sign up with Google

Already have an account? [Sign in](#)

**create a project** with the name below in the mongodb atlas.



cloud.mongodb.com/v2#/org/61a70a1334ad3d4da474709d/projects/create

SONAM'S ORG - 2021-12-01 > PROJECTS

## Create a Project

Name Your Project > Add Members

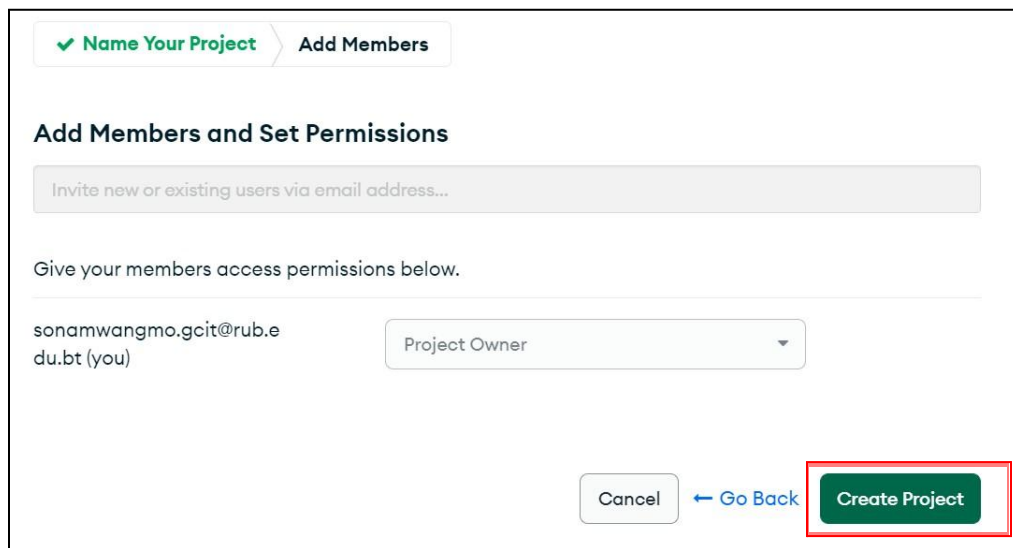
### Name Your Project

Project names have to be unique within the organization (and other restrictions).

EPharmacist-Container

Cancel Next

click Create project.



✓ Name Your Project > Add Members

### Add Members and Set Permissions

Invite new or existing users via email address...

Give your members access permissions below.

sonamwangmo.gcit@rub.e  
du.bt (you)

Project Owner

Cancel Go Back Create Project

Click on “Build a Database” >> Free >> Create


Give username and then click **Autogenerate Secure Password**. *Save the password for future use.*


Then click on ‘Create User’

**Atlas** App Services Charts


credentials are different to your MongoDB Cloud username and password. You can manage existing users via the [Database Access Page](#).

**Username**

**Password** 

**Autogenerate Secure Password**  Copy


Create User


Username	Authentication Type	
Full-stack_Developer	Password	 EDIT

Next, click on Add my current IP address to access the project.

✓ **Where would you like to connect from?**

Enable access for any network(s) that need to read and write data to your cluster.

**My Local Environment**  
Use this to add network IP addresses to the IP Access List. This can be modified at any time.


**Cloud Environment**  
Use this to configure network access between Atlas and your cloud or on-premise environment. Specifically, set up IP Access Lists, Network Peering, and Private Endpoints.

**ADVANCED**

**Add entries to your IP Access List**

Only an IP address you add to your Access List will be able to connect to your project's clusters. You can manage existing IP entries via the [Network Access Page](#).

IP Address	Description	
<input type="text" value="Enter IP Address"/>	<input type="text" value="Enter description"/>	Add Entry
		<b>Add My Current IP Address</b>

IP Access List	Description	
103.197.179.99/32	My IP Address	 REMOVE

click on finish and close.





**Atlas** App Services Charts

existing IP entries via the [Network Access Page](#).

IP Address	Description		
<input type="text" value="Enter IP Address"/>	<input type="text" value="Enter description"/>	<input type="button" value="Add Entry"/>	<input type="button" value="Add My Current IP Address"/>

IP Access List	Description	
103.197.179.99/32	My IP Address	<input type="button" value="REMOVE"/>

You will receive the popup message given.

×

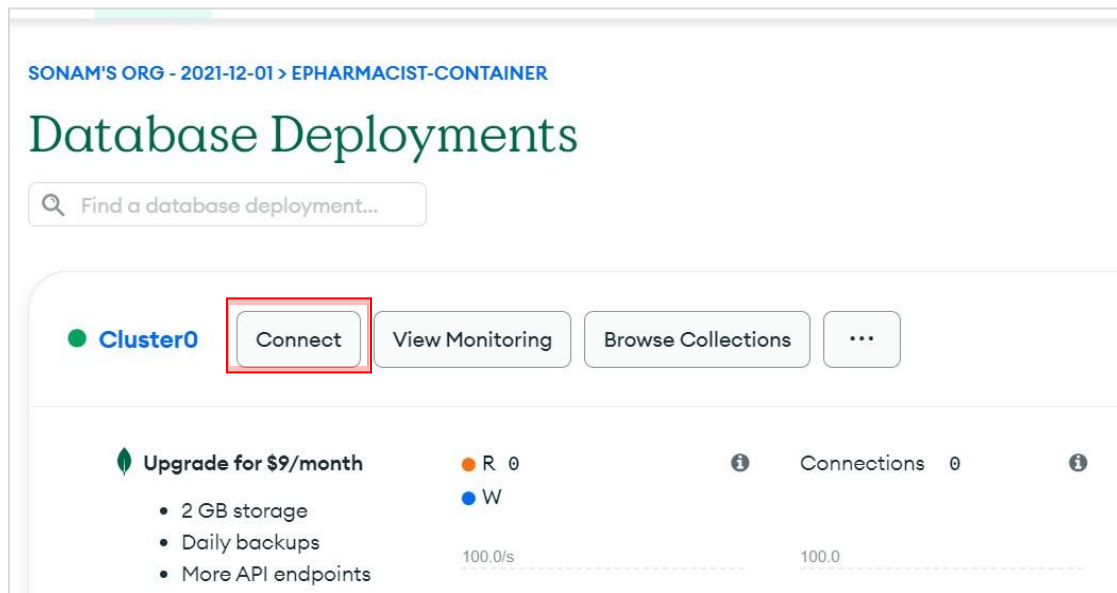
## Congratulations on setting up access rules!

You will now be able to connect to your deployments. You can continue to add and update access rules in [Database Access](#) and [Network Access](#).

☒ Hide Quickstart guide in the navigation. You can visit [Project Settings](#) to access it in the future.

click on **Connect** section to connect the database with the Express App.





### Getting the Connection String for MongoDB Atlas

38. Goto “Choose a connection method” tab and select “Connect Your Application.”


×


## Connect to Cluster0


✓ Setup connection security > Choose a connection method > Connect


Choose a connection method [View documentation](#)

Get your pre-formatted connection string by selecting your tool below.

**Connect with the MongoDB Shell**  
Interact with your cluster using MongoDB's interactive Javascript interface >

**Connect your application**  
Connect your application to your cluster using MongoDB's native drivers >

**Connect using MongoDB Compass**  
Explore, modify, and visualize your data with MongoDB's GUI >

**Connect using VS Code**  
Connect to a MongoDB host in Visual Studio Code >

Go Back

Close

39. Making sure we have Node.js selected as our driver, we'll paste the connection string into our config.env file.



### Connect to Cluster0

✓ Setup connection security
✓ Choose a connection method
Connect

- Select your driver and version**

DRIVER

Node.js

VERSION

4.1 or later
- Add your connection string into your application code**

☐ Include full driver code example

```
mongodbsrv://Full-stack_Developer:<password>@cluster0.tkjo0j9.mongodb.net/?
retryWrites=true&w=majority
```

Replace **<password>** with the password for the **Full-stack\_Developer** user. Ensure any option params are [URL encoded](#).

Having trouble connecting? [View our troubleshooting documentation](#)

Copy the connection string and paste it into the config.env file that you have created.

40. First we need to create and update our **config.env** file to include the **mongoDB** database URI address that we will be creating. Insert the following into the **config.env** file.

Database\_local will be used for using the local mongodb, and you can give any database name [in our case we have given dat

```
config.env U X
config.env
1 DATABASE = mongodbsrv://Full-stack_Developer:PASSWORD@cluster0.tkjo0j9.mongodb.net/epharmacist?retryWrites=true&w=majority
2 DATABASE_LOCAL = mongodb://localhost:27017/epharmacist
3 DATABASE_PASSWORD = tfmsjMJsMlnIZHha
4
5
```

abase name as 'epharmacist'] .

Replace <password> to PASSWORD or you can directly give the autogenerated password there.

Then add any database name “ / database name?”. In our case we have added 'epharmacist' as database name for our mongodb cloud.



Note that password is the *password* of our database and *epharmacist* is the *database name*.

DATABASE\_PASSWORD is the password that you have auto generated while setting up the MongoDB atlas database cloud.

### Setting Up Dependencies for MongoDB

41. Next, we'll install Mongoose from npm (`npm i mongoose`). Mongoose, by the way, is simply a layer of abstraction on top of MongoDB, the same way Express is a layer of abstraction on top of Node.

```
node →/workspace (main X) $ npm i mongoose
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2:
  wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})

+ mongoose@6.6.0
added 28 packages from 69 contributors and audited 118 packages in 18.866s
```

42. Tool like dotenv loads environment variables from a .env file that you create and adds them to the process.env object that is made available to the application. Hence, let install dotenv tool [`npm install dotenv --save`]

```
node →/workspace (main X) $ npm install dotenv --save
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted
  {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})

+ dotenv@16.0.2
added 1 package and audited 119 packages in 3.45s
```

43. We'll require Mongoose at the top of our JS file that starts the server and then invoke Mongoose's connect() method. We need to pass in our connection string, but before we do so, we'll make sure it includes our password:

```
() server.js U X
() server.js > ...
1  const mongoose = require('mongoose')
2  const dotenv = require('dotenv')
3  dotenv.config({ path: './config.env' })
4  const app = require('./app')
5
6  const DB = process.env.DATABASE.replace(
7    'PASSWORD',
8    process.env.DATABASE_PASSWORD,
9  )
10 // console.log(process.env.DATABASE_PASSWORD)
11 mongoose.connect(DB).then((con) => {
12   console.log(con.connections)
13   console.log('DB connection succesful')
14 }).catch(error => console.log(error));
15
16 /* Starting the server on port 4001. */
17 const port = 4001
18 app.listen(port, () => {
19   console.log(`App running on port ${port} ..`)
20 })
```

## Creating an User Model

44. Now let's start to complete the second subtask of the PBI 'Create User Models with roles'.

### Creating a New Feature Branch

45. First and foremost, we need to create a new feature branch in our git repository before we start to do our development work. This is to ensure that whatever we do will not be affecting the source codes in the master branch.



```
node → /workspace/(master X) $ git branch
      config-branch
* master
node → /workspace/(master X) $ git branch sprint-1-branch
node → /workspace/(master X) $ git checkout sprint-1-branch
M      .gitignore
Switched to branch 'sprint-1-branch'
node → /workspace/(sprint-1-branch X) $
```

Once we have finished sprint 1, we will merge this branch back to the master main again.

46. 'Models' are what communicate directly to our database. So in our **model** folder, let's create a **userModels.js** file.

User model is representation of user in database. It defines the schema for a user and provides methods for interacting with the database.

The library we use in Node.js for creating user models is Mongoose. Libraries provides an easy way to define schemas and interact with databases.

### Defining Schema

47. *Schemas* outline what our documents will look like. Note how we use native Javascript types: We can be a little more specific by making our values into options objects: For the required key, we can simply put a boolean or an array containing true and an error message, like [true, 'Please provide valid email'].



```
models > userModels.js U X
models > userModels.js > [0] <unknown>
1  const mongoose = require('mongoose')
2  const validator = require('validator')
3
4  const userSchema = new mongoose.Schema({
5    name: {
6      type: String,
7      required: [true, 'Please tell us your name!'],
8    },
9    email: {
10     type: String,
11     required: [true, 'Please provide your email'],
12     unique: true,
13     lowercase: true,
14     validate: [validator.isEmail, 'Please provide a valid email'],
15   },
16   photo: {
17     type: String,
18     default: 'default.jpg',
19   },
20   role: {
21     type: String,
22     enum: ['user', 'sme', 'pharmacist', 'admin'],
23     default: 'user',
24   },
25   password: {
26     type: String,
27     required: [true, 'Please provide a password!'],
28     minlength: 8,
29     //password wont be included when we get the users
30     select: false,
31   },
32   active: {
33     type: Boolean,
34     default: true,
35     select: false,
36   },
37 })
```

### Creating Documents with Mongoose



48. With our schema in place, we create a *model*. A *model* is analogous to a class in OOP.

```

userModels.js U X
models > userModels.js > userSchema > password
1  const mongoose = require('mongoose')
2  const validator = require('validator')
3
4  const userSchema = new mongoose.Schema({
5    name: {
6      type: String,
7      required: [true, 'Please tell us your name!'],
8    },
9    email: {
10     type: String,
11     required: [true, 'Please provide your email'],
12     unique: true,
13     lowercase: true,
14     validate: [validator.isEmail, 'Please provide a valid email'],
15   },
16   photo: {
17     type: String,
18     default: 'default.jpg',
19   },
20   role: {
21     type: String,
22     enum: ['user', 'sme', 'pharmacist', 'admin'],
23     default: 'user',
24   },
25   password: {
26     type: String,
27     required: [true, 'Please provide a password!'],
28     minlength: 8,
29     //password wont be included when we get the users
30     select: false,
31   },
32   active: {
33     type: Boolean,
34     default: true,
35     select: false,
36   },
37 })
38
39 const User = mongoose.model('User', userSchema)
40 module.exports = User
41

```





Remember to use npm to install the validator library and also import the UserModel so that it can create the model in the MongoDB.

```
npm i validator
```

#### ✓ TERMINAL

```
    topology: [Topology],
    [Symbol(kCapture)]: false,
    [Symbol(options)]: [Object: null prototype]
  },
  '$initialConnection': Promise { [Circular *3] },
  db: Db { s: [Object] },
  host: 'ac-dkp45c-shard-00-00.tkjooj9.mongodb.net',
  port: 27017,
  name: 'epharmacist'
}
]
DB connection succesful
```

49. Congratulations, you have just completed another subtask under the first PBI!  
Mark as complete in the scrum board as shown.

Subtasks

✓ Create an entity relationship diagram of the web app.

✓ Create User Models with roles

⊙ Design UI for patient/care-giver Sign-Up and Login

⊙ Perform front-end form Validation

⊙ Taking Data from forms and inserting them into respective tables

⋮ ⊙ Loading the stored data back into admin panels for verification

⊙ Encrypt the credentials of all the users

50. Finally, push your work in the git repository.

```

● node → /workspace (main X) $ git add .
● node → /workspace (main) $ git commit -m "updates"
[main 3d29e83] updates
 8 files changed, 1009 insertions(+), 2 deletions(-)
 create mode 100644 app.js
 create mode 100644 config.env
 create mode 100644 models/userModels.js
 create mode 100644 package-lock.json
 create mode 100644 package.json
 create mode 100644 server.js
● node → /workspace (main) $ git push
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 8 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (11/11), 13.93 KiB | 460.00 KiB/s, done.
Total 11 (delta 2), reused 0 (delta 0)
To gitlab.com:agile19/csf202_container.git
 e2b2480..3d29e83  main -> main
○ node → /workspace (main) $

```

## Design UI for patient/care-giver Sign-Up and Login

51. Now let's start to complete the third subtask of the PBI, 'Design UI for patient/care-giver Sign-Up and Login'.

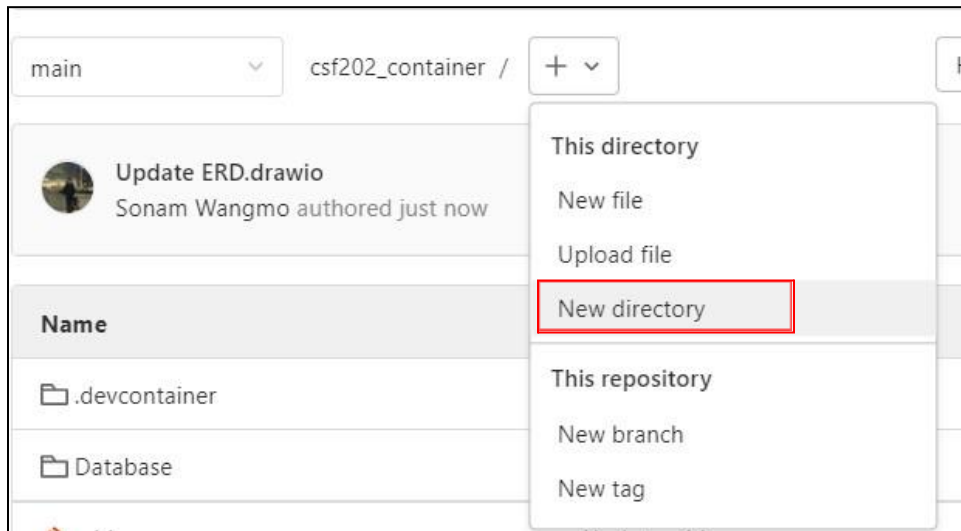
Subtasks

- ✓ Create an entity relationship diagram of the web app.
- ✓ Create User Models with roles
- ✓ Design UI for patient/care-giver Sign-Up and Login
- ✓ Perform front-end form Validation
- ✓ Taking Data from forms and inserting them into respective tables
- ✓ Loading the stored data back into admin panels for verification
- ✓ Encrypt the credentials of all the users

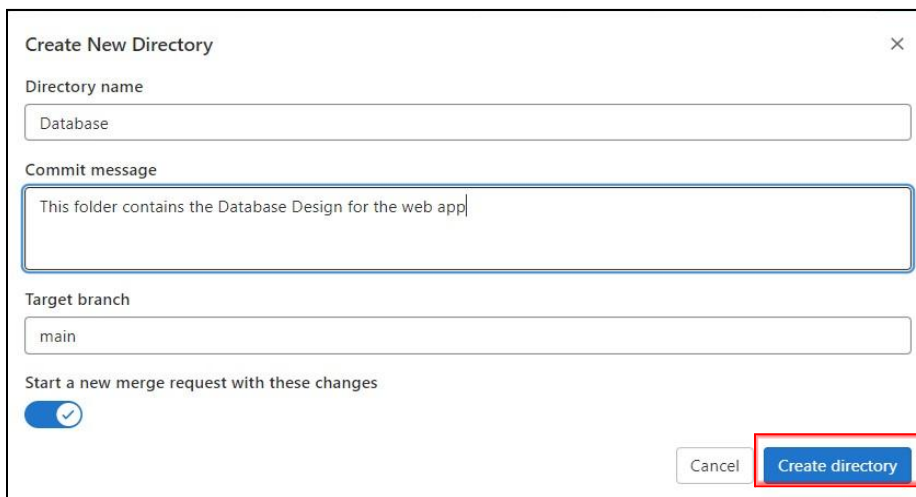
+ Add subtask

52. For setting up the wireframe, we will use the same procedure as Sl No. 7 onwards. At the root folder create a folder called **wireframes** and follow the subsequent steps to create the wireframe.

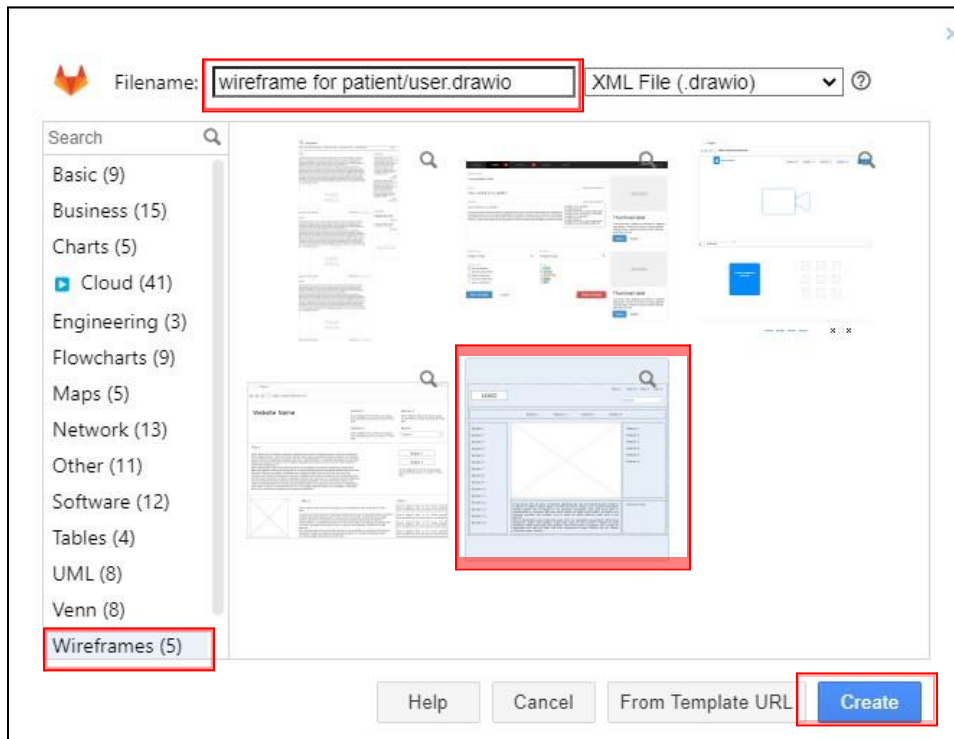
*step1:*



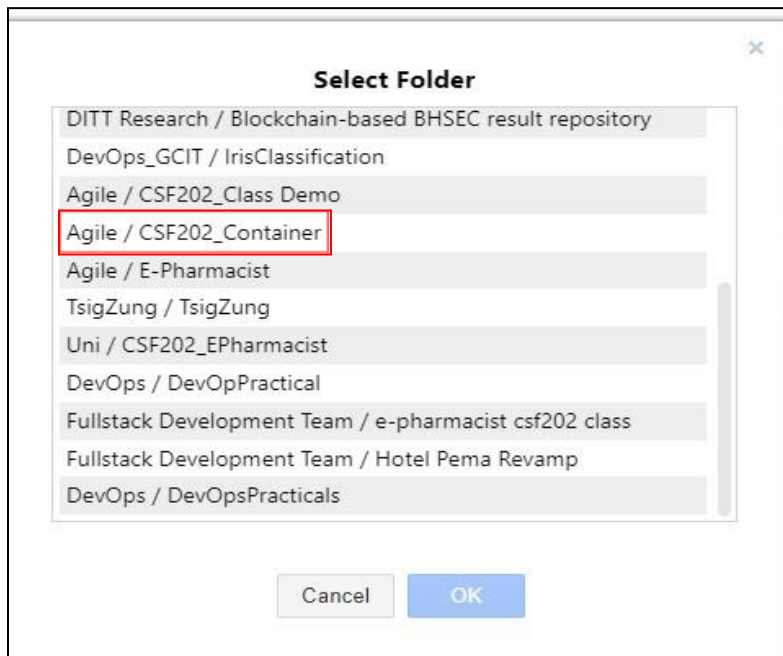
*step2:*

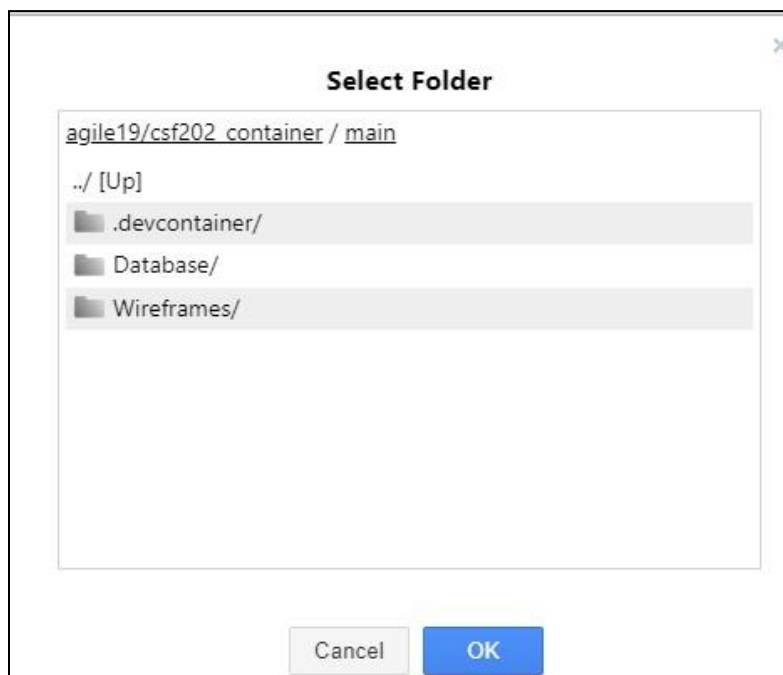
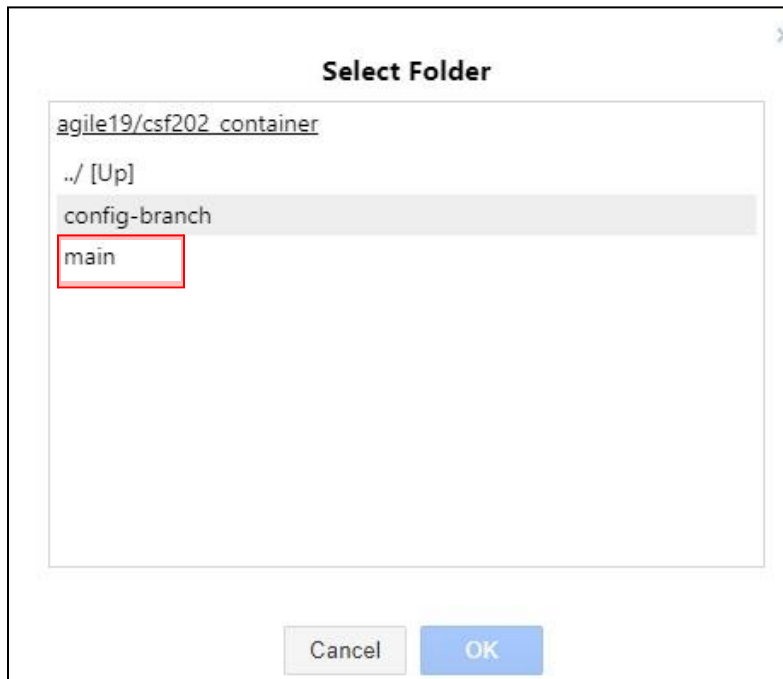


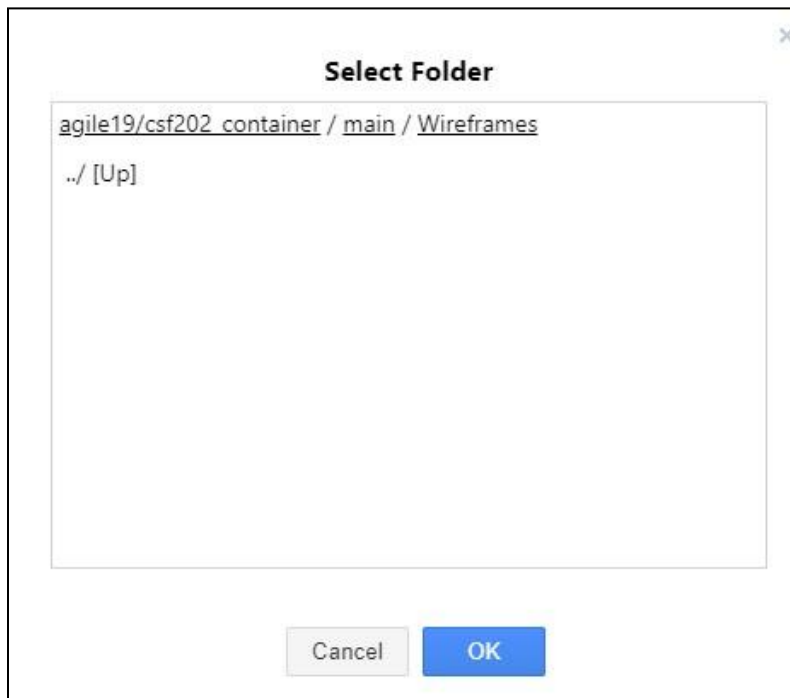
*step3:* Launch a new window in your browser and navigate to: <https://app.diagrams.net/>. Since we already logged in before, directly jump to create a new file. The given page will be displayed.



*step4: Select the repo, branch and folder of your project*







*step5: Commit the changes with the default message.*



53. Based on the template created, modify the wireframe to the one shown below.

ALL NEWS

LOGO

LOGIN

SIGN UP

NEWS

DETAILS

NEWS

DETAILS

NEWS

DETAILS

LOGO

Agile Software Management System

© E-Pharmacist, Autumn Semester 2022!

ALL NEWS

LOGO

LOG INTO YOUR ACCOUNT

Email address

Password

LOGIN

LOGO

Agile Software Management System

© E-Pharmacist, Autumn Semester 2022!

ALL NEWS

LOGO

CREATE A NEW ACCOUNT

Name

Email address

Password

Confirm Password

Role

☐ User
 ☐ Subject Matter Expert
 ☐ Pharmacist

SIGNUP

LOGO

Agile Software Management System

© E-Pharmacist, Autumn Semester 2022!

Once you have finished, click on Save and commit using the default message



## Git: Pulling Changes

Here, we have simulated such changes by creating the wireframe file for the project from an external party. Note that our local repository has not been updated yet. Let's pull the changes from the origin repo (GitLab).

55. In your VSCode terminal, use the command:

```
● node → /workspace (main) $ git pull
Updating 3d29e83..514cdd2
Fast-forward
 Wireframes/wireframe for patient/user.drawio | 170 +++++
 1 file changed, 169 insertions(+), 1 deletion(-)
○ node → /workspace (main) $
```

## ACTIVITY

56. Create the wireframes for your application and place it in the repo.

## Appendix A: Set Up for Debugging

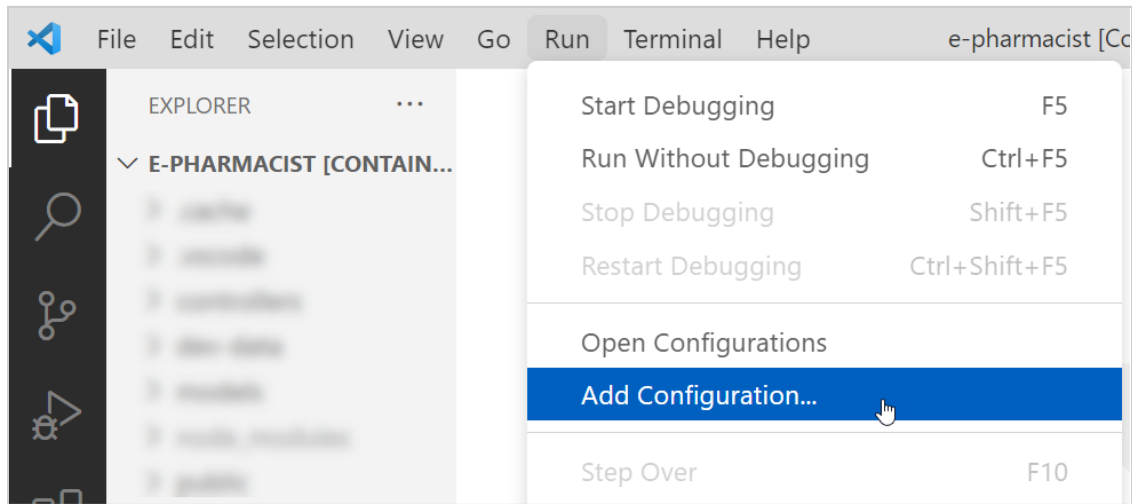
57. It is important that you set up the proper debugging environment when you are doing development work. This will help you to find out the problems that you are facing during coding as well as allow others to help us in the troubleshooting process. Fortunately, the VSCode allows you to perform debugging tasks easily.

For our project, we are using 2 different environments, one is the client side (browser) and the other is the server side (nodeJS + Express). As such we need 2 debugging environments for our purpose.

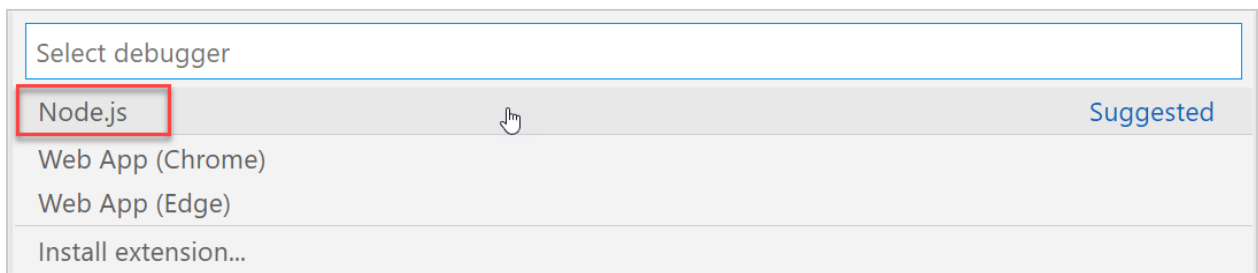


## Setup Debugging For nodeJS

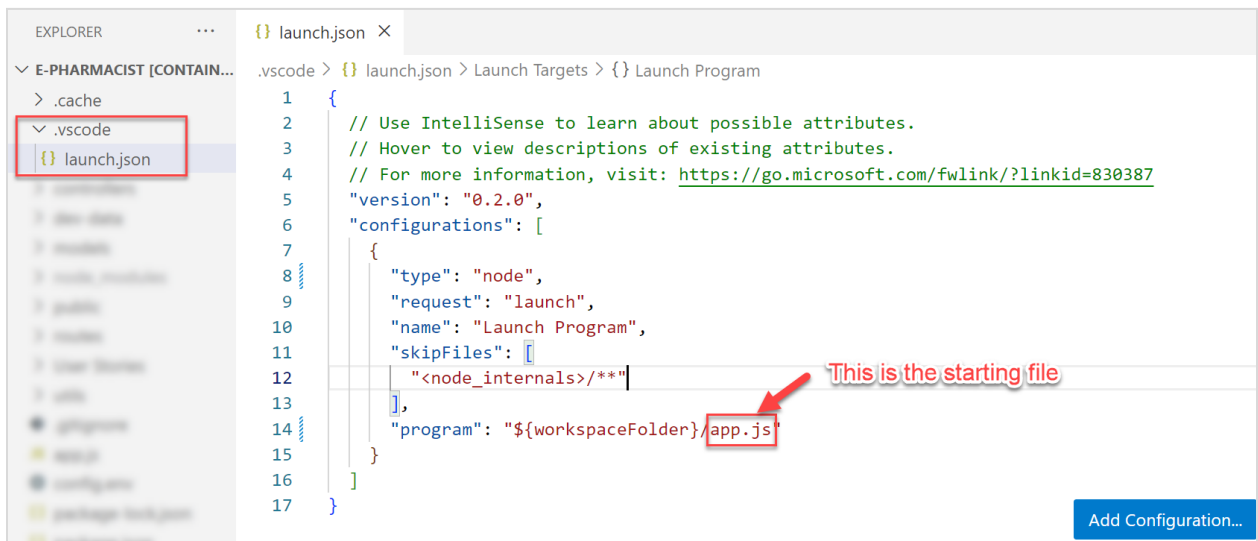
58. Assuming that you have already set up your project and are ready to run your server application. Goto **Run > Add Configuration**.



59. Choose Node.js as the platform that you want to conduct debugging.

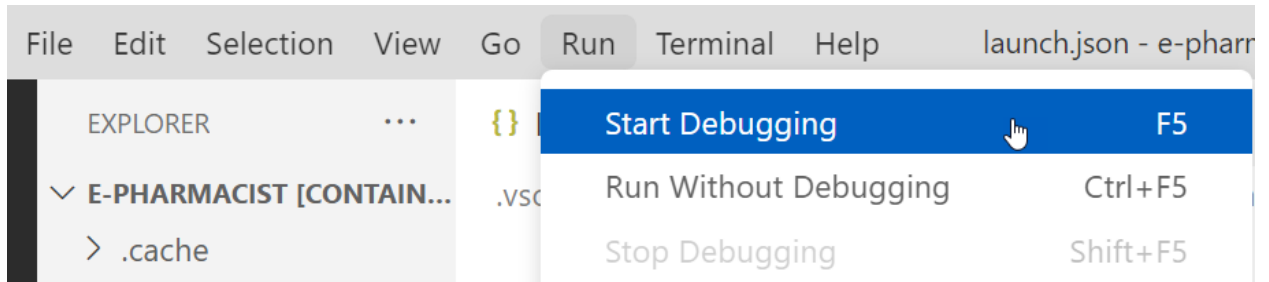


60. This will create a **.vscode** folder containing the configuration file called **launch.json** in the project folder. The starting file for our project is **app.js**.



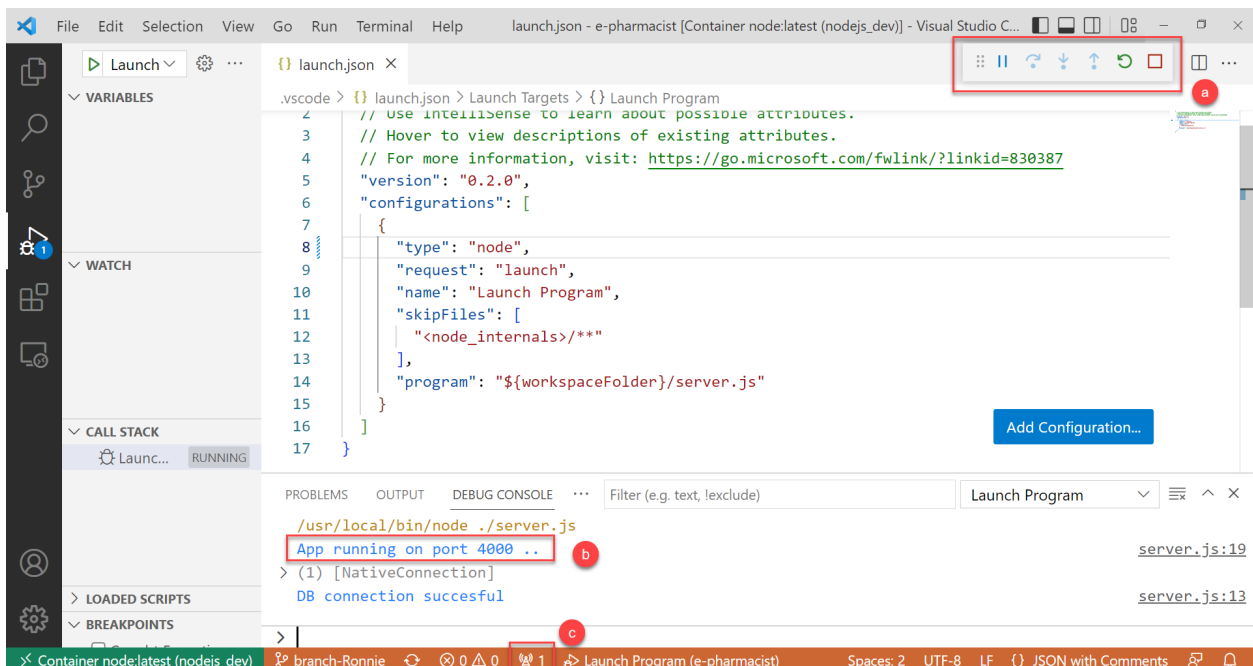
## Starting the Debug Session

61. Now you are ready to launch your application in debugging mode. Go to **Run** ➤ **Start Debugging (F5)**.

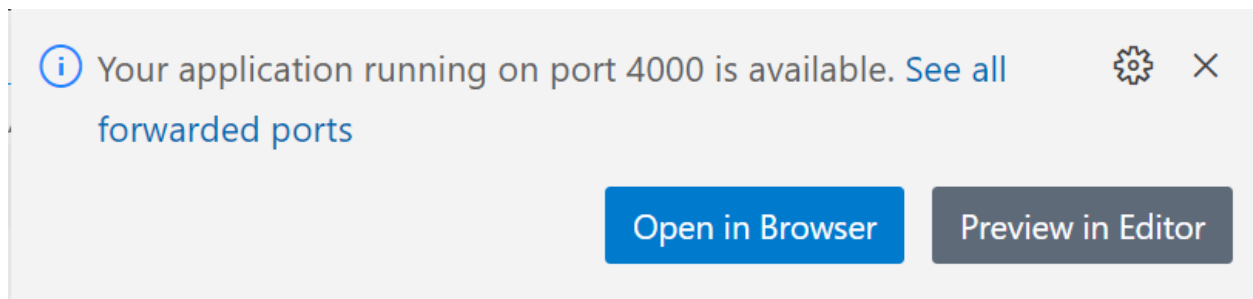


62. You should see 3 things in your VS Code:

- The Debugging Toolbar should appear indicating that the application is currently running in debug mode.
- In the Debug console tab, you should see the console log indicating that the application is running on port 4000. Note that the port number depends on what you set in your application.
- The port forwarding tab indicates that there is 1 port forwarded.



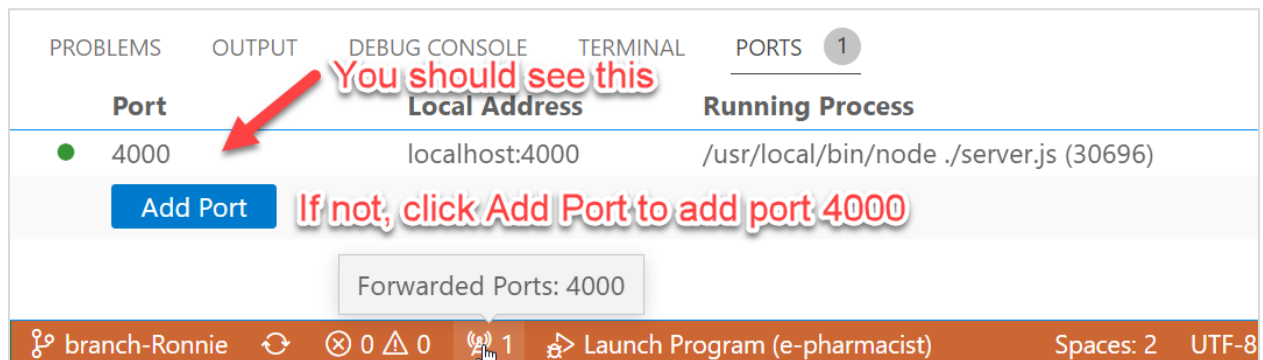
Sometimes, you can see the following message:



Click on Open in Browser to launch your application. You should see your application running.

### Port Forwarding

If you do not get the above message, click on the **port forwarding** icon at the bottom of the VS Code as shown below. You should see the port that is forwarded. If you do not see the port forwarding, chances are you will not be able to navigate to the page that is running. Click on **Add Port** to add port 4000. Now go to your browser and you should see your application running.



~ End of Practical ~