**Introduction**

* 1. **Overview**

The main objective of this application is to build an online bed slot booking platform, which allows a user to book a bed in a preferred hospital looking at the number of beds available.

The project Covid Bed Slot Booking System includes registration of patients, storing their details into the system, and also computerized billing for beds in the hospitals. The software has the facility to give a unique id(srfid) for every patient and stores the details of every patient automatically. It includes a facility to know the current status of availability of beds at each listed hospital.

The Covid Bed Slot Booking System can be entered using a username and password. It is only accessible to admin. Only they can add data into the database. The data can be retrieved easily. The interface is very user-friendly. The data is well protected for personal use and makes the data processing very fast.

The Covid Bed Slot Booking System is powerful, flexible, and easy to use and is designed and developed to deliver real conceivable benefits to hospitals and patients.

Bed Slot Booking System is designed for multispecialty hospitals, to cover a wide range of hospital administration and management processes. It is an integrated end-to-end Management System that provides relevant information across the hospital to support effective decision making for patient care, hospital administration and critical financial accounting, in a seamless flow.

Bed Slot Booking System is a software product suite designed to improve the quality and management of hospitals in the areas of clinical process analysis and activity-based costing. It enables you to develop your organization and improve its effectiveness and quality of work. Managing the key processes efficiently is critical to the success of the hospital and helps you manage your processes.

**1.2 Problem Statement**

“To design and develop a system that will manage information about list of various hospitals, availability of beds in hospitals, allotment of beds for patients and display patient details”.

**Back End Design**

**2.1 Conceptual Database Design (ER Diagram)**



Figure 2.1:ER DIAGRAM

The ER diagram for Covid Bed Slot Booking System is as shown above in the Figure 2.1.

The covid bed slot booking database system mainly consists of 6 entities namely User, Admin, Hospital data, Hospital user, Patient, Triggered data.

* **User** and **patients** have **1:1** cardinality ratio, since 1 user can book 1 slot for patient having **unique SDFID**. User has **total participation** in all relation.
* **User** can *check* **Triggered data** of the Hospital which contains all the details about available beds in the hospital and has **M: N** cardinality ratio
* **Hospital user** can *add or update* the **Hospital data** which consists of available beds in the hospital and will get updated regularly.
* Hospital data and Hospital user has **M: N** cardinality ratio.
* **Hospital user** and **Triggered data** have **M: N** cardinality ratio and both have **total participation**.
* **Admin** *adds* **Hospital user** and in-turn hospital user adds the hospital data. Admin has **total participation** in adding hospital user and has **1: N** cardinality ratio.

**2.2 Logical Database Design (ER MAPPING)**

A schema diagram can display only some aspects of a schema like the name of record type, data type and constraints. Other aspects can’t be specified through the schema diagram.

**ADMIN** (Username, Password)

**HOSPITAL USER** (Hoce, Password, Email)

**HOSPITAL DATA** (Hoce, Hname, Normalbed, Vbed, HICUbed, ICUbed)

**TRIGGERED DATA** (Hoce, Normalbed, Vbed, HICUbed, ICUbed, Query, Date)

**USER** (Email, Srfid, Dob)

**PATIENTS** (Hoce, Srfid, Bedtype, SPO2, Pphone, Paddress)

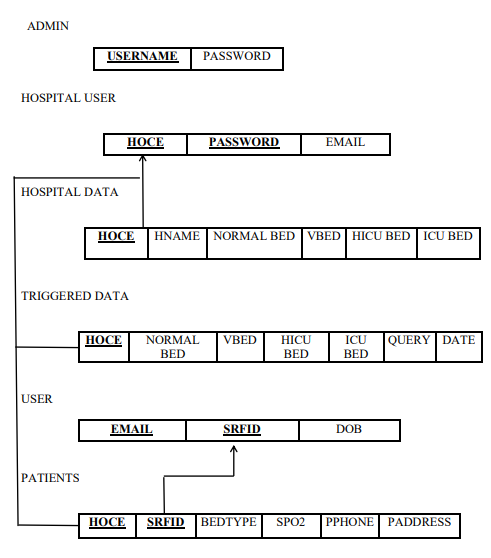


Figure 2.2:ER to Relational Mapping

**2.3 Normalization**

Normalization is the process of reorganizing data in a database so that it meets two basic requirements:

1. There is no redundancy of data (all data is stored in only one place) and
2. Data dependencies are logical (all related data items are stored together).

**First Normal Form (1NF)**

The table is said to be in 1NF if it follows following rules

* It should only have single(atomic) valued attributes/columns.
* Values stored in a column should be the same domain.
* All the columns in a table should have unique names.
* And the order in which data is stored, does not matter.

**Second Normal Form (2NF)**

For a table to be in 2NF, following conditions must be followed

* It should be in the First Normal Form
* It should not have partial functional dependency.

Partial dependency means that a non-prime attribute is functionally dependent on part of a candidate key.

**Third Normal Form (3NF)**

A table is said to be in 3NF when,

* It is in the Second Normal Form (i.e., it should not have partial functional dependency).
* It does not have transitive dependency.

A functional dependency is said to be transitive if it is indirectly formed by two functional dependencies. For example:

X → Z is a transitive dependency if the following functional dependencies hold true:

X → Y

Y → Z

**BOOKING PATIENT**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| PID | Srfid | Bedtype | Hcode | Spo2 | Pname | Pphone | Paddress |

**Functional Dependencies**

PID --> (Srfid, Bedtype, Hcode, Spo2, Pname, Pphone, Paddress)

PID+= (PID, Srfid, Bedtype, Hcode, Spo2, Pname, Pphone, Paddress)

Candidate key= PID

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| PID | Srfid | Bedtype | Hcode | Spo2 | Pname | Pphone | Paddress |
| 1 | KA12345678 | ICUBed | BBH01 | 89 | Rahul | 9833276236 | Bangalore |
| 2 | KA64527123 | HICUBed | BBA02 | 85 | Akshay | 6453729819 | Mysore |

Table 2.1:Booking Patient

**Justification**

* The above-mentioned dependencies we can observe that the BOOKINGPATIENT relation is in 1NF as all the attributes are atomic in nature.
* The relation is in 2NF as all the Non-Prime attributes are fully functionally dependent on the Prime attribute and there is no partial dependency.
* The relation is also in 3NF as there is no Transitive dependency between the Non-Prime attributes and the Prime attribute.

**HOSPITAL DATA**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| HDID | Hcode | Hname | Normalbed | HICUbed | ICUbed | Vbed |

**Functional Dependencies**

HDID --> (Hcode, Hname, Normalbed, HICUbed, ICUbed, Vbed)

HDID+= (HDID, Hcode, Hname, Normalbed, HICUbed, ICUbed, Vbed)

Candidate key= HDID

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| HDID | Hcode | Hname | Normalbed | HICUbed | ICUbed | Vbed |
| 1 | BBH01 | KIMS | 15 | 9 | 4 | 2 |
| 2 | BBA02 | Apollo | 30 | 19 | 5 | 7 |

Table 2.2:Hospital Data

**Justification**

* The above-mentioned dependencies we can observe that the HOSPITALDATA relation is in 1NF as all the attributes are atomic in nature.
* The relation is in 2NF as all the Non-Prime attributes are fully functionally dependent on the Prime attribute and there is no partial dependency.
* The relation is also in 3NF as there is no Transitive dependency between the Non-Prime attributes and the Prime attribute.

**HOSPITAL USER**

|  |  |  |  |
| --- | --- | --- | --- |
| HUID | Hcode | Email | Password |

**Functional Dependencies**

HUID --> (Hcode, Hname, Email, Password)

HUID+= (HUID, Hcode, Hname, Email, Password)

Candidate key= HUID

|  |  |  |  |
| --- | --- | --- | --- |
| HUID | Hcode | Email | Password |
| 2 | BBH01 | [myspace.amd@gmail.com](mailto:myspace.amd@gmail.com) | 2022bbh01 |
| 5 | BBA02 | [abhishekgowda1906@gmail.com](mailto:abhishekgowda1906@gmail.com) | 2022bba02 |

Table 2.3:Hospital User

**Justification**

* The above-mentioned dependencies we can observe that the HOSPITALUSER relation is in 1NF as all the attributes are atomic in nature.
* The relation is in 2NF as all the Non-Prime attributes are fully functionally dependent on the Prime attribute and there is no partial dependency.
* The relation is also in 3NF as there is no Transitive dependency between the Non-Prime attributes and the Prime attribute.

**TRIGGERED DATA**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| TDID | Hcode | Normalbed | HICUbed | ICUbed | Vbed | Queries | Date |

**Functional Dependencies**

TDID --> (Hcode, Normalbed, HICUbed, ICUbed, Vbed, Queries, Date)

TDID+= (TDID, Hcode, Normalbed, HICUbed, ICUbed, Vbed, Queries, Date)

Candidate key= TDID

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| TDID | Hcode | Normalbed | HICUbed | ICUbed | Vbed | Queries | Date |
| 1 | BBH01 | 15 | 10 | 4 | 3 | UPDATES | 2022-01-26 |
| 2 | BBH01 | 15 | 9 | 4 | 2 | UPDATES | 2022-01-26 |

Table 2.4:Triggered Data

**Justification**

* The above-mentioned dependencies we can observe that the TRIGGERDATA relation is in 1NF as all the attributes are atomic in nature.
* The relation is in 2NF as all the Non-Prime attributes are fully functionally dependent on the Prime attribute and there is no partial dependency.
* The relation is also in 3NF as there is no Transitive dependency between the Non-Prime attributes and the Prime attribute.

**USER**

|  |  |  |  |
| --- | --- | --- | --- |
| UID | Srfid | Email | Dob |

**Functional Dependencies**

UID --> (Srfid, Email, Dob)

UID+= (UID, Srfid, Email, Dob)

Candidate key= UID

|  |  |  |  |
| --- | --- | --- | --- |
| UID | Srfid | Email | Dob |
| 3 | KA12345678 | [amd@gmai.com](mailto:amd@gmai.com) | 16-06-2001 |
| 4 | KA64527123 | [akshay@gmail.com](mailto:akshay@gmail.com) | 10-07-2007 |

Table 2.5:User

**Justification**

* The above-mentioned dependencies we can observe that the USER relation is in 1NF as all the attributes are atomic in nature.
* The relation is in 2NF as all the Non-Prime attributes are fully functionally dependent on the Prime attribute and there is no partial dependency.
* The relation is also in 3NF as there is no Transitive dependency between the Non-Prime attributes and the Prime attribute.

**Front End Design**

**3.1 Screen Layout Design for Webpages, forms**

Screen layout is the part of graphic design that deals with the arrangement of visual elements on a page. Screen layout is used to make the web pages look better. It establishes the overall appearance, relative importance and relationships between the graphic elements to achieve a smooth flow of information and eye movement for maximum effectiveness or impact.

Using visually attractive and simple design forms the basis of all the screen layout design and using these layouts in authentication and login makes it easy to use and user-friendly.

Screen layout determines the overall structure of your screen. They define the structure of a harness included in a composite portal. A harness can contain only one screen layout.

A screen layout is comprised of the following:

* <header> - Defines a header for a document or a section
* <nav> - Defines a set of navigation links
* <section> - Defines a section in a document
* <article> - Defines an independent, self-contained content
* <aside> - Defines content aside from the content (like a sidebar)
* <footer> - Defines a footer for a document or a section

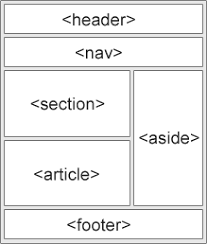


Figure 3.1**:** Screen Layout Design

**3.2 Front End and Back End Connectivity**

**Flask SQLAlchemy with MySQL**

MySQL is developed, distributed, and supported by Oracle Corporation. MySQL is a database system used on the web that runs on a server. MySQL is ideal for both small and large applications. It is very fast, reliable, and easy to use. It supports standard SQL. MySQL can be compiled on a number of platforms.

**What is SQLAlchemy ?**

SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.

It provides a full suite of well-known enterprise-level persistence patterns, designed for efficient and high-performing database access, adapted into a simple and Pythonic domain language.

**What is Flask-SQLAlchemy ?**

Flask-SQLAlchemy is an extension for Flask that adds support for SQLAlchemy to your application. It aims to simplify using SQLAlchemy with Flask by providing useful defaults and extra helpers that make it easier to accomplish common tasks.

**Installation**

You can simply install Flask-SQLAlchemy using pip command.

**pip install Flask-SQLAlchemy**

So now after installation you need to create database, we are using Mysql database. For this you need to download and install Wampp Server. Create a database in the Wamp Server,with flaskcodeloop for the database name, but you can give the name according to your choice.

Right now we don’t have any tables in our database, we will create this using Flask SQLAlchemy.

Below is a **app.py** and we have added our SQLAlchemy and database configuration.

from flask import Flask, render\_template, flash, request, redirect, url\_for

from forms import LoginForm

from flask\_sqlalchemy import SQLAlchemy

**#create the object of Flask**

app = Flask(\_\_name\_\_)

app.config['SECRET\_KEY'] = 'hardsecretkey'

**#SqlAlchemy Database Configuration With Mysql**

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'mysql://root:''@localhost/flaskcodeloop'

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

db = SQLAlchemy(app)

**#our model**

class UserInfo(db.Model):

id = db.Column(db.Integer, primary\_key = True)

username = db.Column(db.String(100), unique = True)

password = db.Column(db.String(100))

def \_\_init\_\_(self, username, password):

self.username = username

self.password = password

**#creating our routes**

@app.route('/')

def index():

return render\_template('index.html')

**#login route**

@app.route('/login' , methods = ['GET', 'POST'])

def Login():

form = LoginForm()

if form.validate\_on\_submit():

if request.form['username'] != 'codeloop' or request.form['password'] != '12345':

flash("Invalid Credentials, Please Try Again")

else:

return redirect(url\_for('index'))

return render\_template('login.html', form = form)

**#run flask app**

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

In the below code this is for SQLAlchemy.

**#SqlAlchemy Database Configuration with Mysql**

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'mysql://root:''@localhost/flaskcodeloop'

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

db = SQLAlchemy(app)

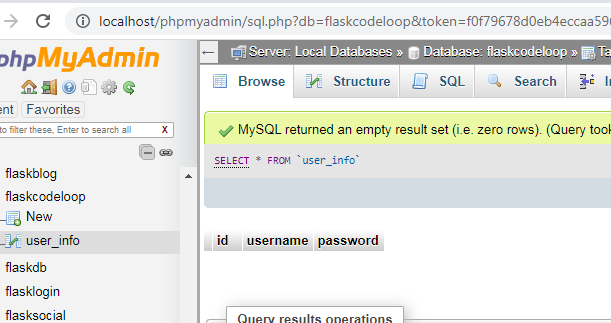
If you check your flaskcodeloop database, you can see that we have our table with the data. 

Fig 3.2.1:Back end database

Now let’s add some data to our database. basically, we are going to add just two UserInfo

to our database table.

>>> from app import db

>>> from app import UserInfo

>>> u = UserInfo('codeloop',1234)

>>> u1 = UserInfo('parwiz', 12345)

>>> db.session.add(u)

>>> db.session.add(u1)

>>> db.session.commit()

Check your database table, you will have two UserInfo data.

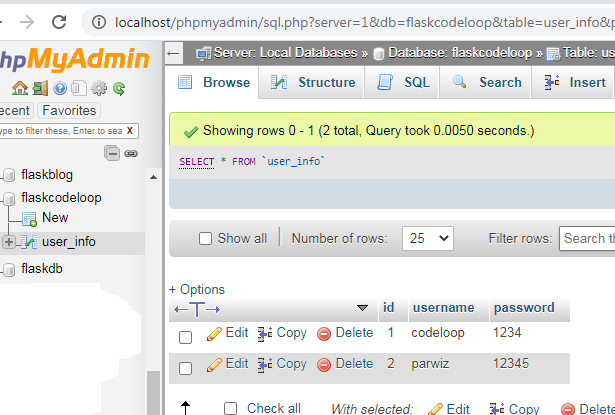


Fig 3.2.2:Back end database table values

**Major Modules**

**4.1 Admin module**

In this module Admin can add list of hospitals and send email to the hospital containing hospital code and password to login. Admin adds various hospital data using his credentials. Only Admin has the right to add hospital lists. Admin module has several sub-modules in it. They are

1. **Login**: Admin uses user\_id and password to login.
2. **Add Hospital user**: Adds hospital user consisting of email, hospital code, password.
3. **Logout**: Admin logs out of the page.

**4.2 Hospital module**

In Hospital module, listed hospitals can add hospital details like various beds (ICU, Normal, Ventilator etc..) available in their hospital. The entered details can be modified (update, delete) and can be viewed using triggered data section. If invalid credentials are entered error message is displayed.   
Hospital module has several sub-modules in it. They are

1. **Login**: Hospital user uses email id and password to login.
2. **Add Hospital Data**: Adds hospital data consisting of hospital bed details.
3. **View Triggered data**: Hospital user can view triggered data like update, insert, delete operations.
4. **Logout**: Hospital user logs out of the page.

**4.3 Patient module**

Patients can login/sign-in using unique SRFID (given for covid +ve patients) and email. Based on available beds in various hospitals they can book the type of bed they want. The details entered by the patients can be seen in patient detail section. Hospital module has several sub-modules in it. They are

1. **Login**: User uses srf\_id and date-of-birth to login.
2. **Booking Bed Slot**: User can book slot among the available beds in the listed hospital.
3. **View Available beds**: View the beds available in the listed hospital.
4. **Patient Details**: User can view the entered patient details.
5. **Logout**: Patient logs out of the page.

**Implementation**

**5.1 Create Statements**

CREATE TABLE `bookingpatient` (

`id` int(11) NOT NULL,

`srfid` varchar(20) NOT NULL,

`bedtype` varchar(50) NOT NULL,

`hcode` varchar(50) NOT NULL,

`spo2` int(11) NOT NULL,

`pname` varchar(50) NOT NULL,

`pphone` varchar(12) NOT NULL,

`paddress` text NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

ALTER TABLE `bookingpatient`

ADD PRIMARY KEY (`id`),

ADD UNIQUE KEY `srfid` (`srfid`);

ALTER TABLE `bookingpatient`

MODIFY `id` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=2;

CREATE TABLE `hospitaldata` (

`id` int(11) NOT NULL,

`hcode` varchar(200) NOT NULL,

`hname` varchar(200) NOT NULL,

`normalbed` int(11) NOT NULL,

`hicubed` int(11) NOT NULL,

`icubed` int(11) NOT NULL,

`vbed` int(11) NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

ALTER TABLE `hospitaldata`

ADD PRIMARY KEY (`id`),

ADD UNIQUE KEY `hcode` (`hcode`);

ALTER TABLE `hospitaldata`

MODIFY `id` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=4;

CREATE TABLE `hospitaluser` (

`id` int(11) NOT NULL,

`hcode` varchar(20) NOT NULL,

`email` varchar(100) NOT NULL,

`password` varchar(1000) NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

ALTER TABLE `hospitaluser`

ADD PRIMARY KEY (`id`);

ALTER TABLE `hospitaluser`

MODIFY `id` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=6;

CREATE TABLE `trig` (

`id` int(11) NOT NULL,

`hcode` varchar(50) NOT NULL,

`normalbed` int(11) NOT NULL,

`hicubed` int(11) NOT NULL,

`icubed` int(11) NOT NULL,

`vbed` int(11) NOT NULL,

`querys` varchar(50) NOT NULL,

`date` date NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

ALTER TABLE `trig`

ADD PRIMARY KEY (`id`);

ALTER TABLE `trig`

MODIFY `id` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=3;

CREATE TABLE `user` (

`id` int(11) NOT NULL,

`srfid` varchar(20) NOT NULL,

`email` varchar(50) NOT NULL,

`dob` varchar(1000) NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

ALTER TABLE `user`

ADD PRIMARY KEY (`id`),

ADD UNIQUE KEY `srfid` (`srfid`);

ALTER TABLE `user`

MODIFY `id` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=4;

**5.2 Trigger statements and Data insertion**

CREATE TRIGGER `Delete` BEFORE DELETE ON `hospitaldata` FOR EACH ROW INSERT INTO trig

VALUES(null,OLD.hcode,OLD.normalbed,OLD.hicubed,OLD.icubed,OLD.vbed,'DELETED',NOW())

CREATE TRIGGER `Insert` AFTER INSERT ON `hospitaldata` FOR EACH ROW INSERT INTO trig

VALUES(null,NEW.hcode,NEW.normalbed,NEW.hicubed,NEW.icubed,NEW.vbed,'INSERTED',NOW())

CREATE TRIGGER `Update` AFTER UPDATE ON `hospitaldata` FOR EACH ROW INSERT INTO trig

VALUES(null,NEW.hcode,NEW.normalbed,NEW.hicubed,NEW.icubed,NEW.vbed,'UPDATED',NOW())

INSERT INTO `bookingpatient` (`id`, `srfid`, `bedtype`, `hcode`, `spo2`, `pname`, `pphone`, `paddress`) VALUES

(1, 'KA12345678', 'ICUBed', 'BBH01', 89, 'Rahul', '9833276236', 'Bangalore');

INSERT INTO `hospitaldata` (`id`, `hcode`, `hname`, `normalbed`, `hicubed`, `icubed`, `vbed`) VALUES

(2, 'BBH01', 'KIMS', 15, 9, 4, 2),

(3, 'BBA02', 'bangalore', 30, 19, 5, 7);

INSERT INTO `hospitaluser` (`id`, `hcode`, `email`, `password`) VALUES

(2, 'BBH01', '[myspace.amd@gmail.com](mailto:myspace.amd@gmail.com)', 'pbkdf2:sha256:260000$M1iXPteF8RJ2qbM3$622f475ce5dd3de53f383487c0415624da68d2a203b38bd0efa7e6c803a0b632'),

(5, 'BBA02', '[abhishekgowda1906@gmail.com](mailto:abhishekgowda1906@gmail.com)', 'pbkdf2:sha256:260000$QuPrZOfiOdQlAaSh$612549985607d0805543f933ab8ed3215993f7165b6a506fce61636127f90fcd');

INSERT INTO `trig` (`id`, `hcode`, `normalbed`, `hicubed`, `icubed`, `vbed`, `querys`, `date`) VALUES

(1, 'BBH01', 15, 10, 4, 3, 'UPDATED', '2022-01-26'),

(2, 'BBH01', 15, 9, 4, 2, 'UPDATED', '2022-01-26');

INSERT INTO `user` (`id`, `srfid`, `email`, `dob`) VALUES

(3, 'KA12345678', '[amd@gmail.com](mailto:amd@gmail.com)', 'pbkdf2:sha256:260000$UeXj8AGlpZ050nCv$c7f0f357f3db0211c9a984db4e305b88de8722f21d354425e62c303a816220c3');

**5.3 Python Code**

**5.3.1 Packages**

from flask import Flask, json, redirect, render\_template, flash, request

from flask.globals import request, session

from flask.helpers import url\_for

from flask\_sqlalchemy import SQLAlchemy

from flask\_login import UserMixin

from werkzeug.security import generate\_password\_hash, check\_password\_hash

from flask\_login import login\_required, logout\_user, login\_user, login\_manager, LoginManager, current\_user

from flask\_mail import Mail

import json

**5.3.2 Database connection**

local\_server = True

app = Flask(\_\_name\_\_)

app.secret\_key="darshan"

**#config file**

with open('config.json', 'r') as c:

params = json.load(c)["params"]

**#configure the mail**

app.config.update(

MAIL\_SERVER='smtp.gmail.com',

MAIL\_PORT='465',

MAIL\_USE\_SSL=True,

MAIL\_USERNAME=params['gmail-user'],

MAIL\_PASSWORD=params['gmail-password']

)

mail = Mail(app)

**# this is for getting the unique user access**

login\_manager = LoginManager(app)

login\_manager.login\_view = 'login'

**# app.confing**

['SQLALCHEMY\_DATABASE\_URI'] = 'mysql://username:password@localhost/databasename'

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'mysql://root:@localhost/covid'

db = SQLAlchemy(app)

@login\_manager.user\_loader

def load\_user(user\_id):

return User.query.get(int(user\_id)) or Hospitaluser.query.get(int(user\_id))

**5.3.3 Database creation**

**#test db**

class Test(db.Model):

id = db.Column(db.Integer,primary\_key=True)

name = db.Column(db.String(50))

**#user login db**

class User(UserMixin,db.Model):

id = db.Column(db.Integer,primary\_key=True)

srfid = db.Column(db.String(20),unique=True)

email = db.Column(db.String(50))

dob = db.Column(db.String(1000))

**#hospital user db**

class Hospitaluser(UserMixin, db.Model):

id = db.Column(db.Integer, primary\_key=True)

hcode = db.Column(db.String(20))

email = db.Column(db.String(50))

password = db.Column(db.String(1000))

**#hospital user db**

class Hospitaldata(db.Model):

id = db.Column(db.Integer, primary\_key=True)

hcode=db.Column(db.String(20),unique=True)

hname=db.Column(db.String(100))

normalbed=db.Column(db.Integer)

hicubed=db.Column(db.Integer)

icubed=db.Column(db.Integer)

vbed=db.Column(db.Integer)

**#booking patient db**

class Bookingpatient(db.Model):

id=db.Column(db.Integer,primary\_key=True)

srfid=db.Column(db.String(20),unique=True)

bedtype=db.Column(db.String(100))

hcode=db.Column(db.String(20))

spo2=db.Column(db.Integer)

pname=db.Column(db.String(100))

pphone=db.Column(db.String(100))

paddress=db.Column(db.String(100))

**#Trigger data**

class Trig(db.Model):

id = db.Column(db.Integer, primary\_key=True)

hcode=db.Column(db.String(20))

normalbed=db.Column(db.Integer)

hicubed=db.Column(db.Integer)

icubed=db.Column(db.Integer)

vbed=db.Column(db.Integer)

querys=db.Column(db.String(50))

date=db.Column(db.String(50))

**5.3.4 Different Routes**

@app.route("/")

def home():

return render\_template("index.html")

@app.route("/signup",methods=['POST','GET'])

def signup():

if request.method == "POST":

srfid = request.form.get('srf')

email = request.form.get('email')

dob = request.form.get('dob')

#print(srfid,email,dob)

encpassword = generate\_password\_hash(dob)

user=User.query.filter\_by(srfid=srfid).first()

emailUser=User.query.filter\_by(email=email).first()

if user or emailUser:

flash("Email or srif is already taken","warning")

return render\_template("userSignup.html")

new\_user = db.engine.execute(f"INSERT INTO `user` (`srfid`,`email`,`dob`) VALUES ('{srfid}','{email}','{encpassword}') ")

flash("SignUp Success Please Login", "success")

return render\_template("userlogin.html")

return render\_template("userSignup.html")

**#login feedback**

@app.route("/login", methods=['POST', 'GET'])

def login():

if request.method == "POST":

srfid = request.form.get('srf')

dob = request.form.get('dob')

user = User.query.filter\_by(srfid=srfid).first()

if user and check\_password\_hash(user.dob,dob):

login\_user(user)

flash("Login succesful","info")

return render\_template("index.html")

else:

flash("Invalid credentials","danger")

return render\_template("userlogin.html")

return render\_template("userlogin.html")

**#hospital login**

@app.route('/hospitallogin', methods=['POST', 'GET'])

def hospitallogin():

if request.method == "POST":

email = request.form.get('email')

password = request.form.get('password')

user = Hospitaluser.query.filter\_by(email=email).first()

if user and check\_password\_hash(user.password, password):

login\_user(user)

flash("Login Success", "info")

return render\_template("index.html")

else:

flash("Invalid Credentials", "danger")

return render\_template("hospitallogin.html")

return render\_template("hospitallogin.html")

**#admin login**

@app.route('/admin', methods=['POST', 'GET'])

def admin():

if request.method == "POST":

username = request.form.get('username')

password = request.form.get('password')

if(username == params['user'] and password == params['password']):

session['user'] = username

flash("login success", "info")

return render\_template("addHosUser.html")

else:

flash("Invalid Credentials", "danger")

return render\_template("admin.html")

**#add hospital user**

@app.route('/addHospitalUser', methods=['POST', 'GET'])

def hospitalUser():

if('user' in session and session['user'] == params['user']):

if request.method == "POST":

hcode = request.form.get('hcode')

email = request.form.get('email')

password = request.form.get('password')

encpassword = generate\_password\_hash(password)

hcode = hcode.upper()

emailUser = Hospitaluser.query.filter\_by(email=email).first()

if emailUser:

flash("Email or srif is already taken", "warning")

db.engine.execute(

f"INSERT INTO `hospitaluser` (`hcode`,`email`,`password`) VALUES ('{hcode}','{email}','{encpassword}') ")

mail.send\_message('COVID CARE CENTER', sender=params['gmail-user'], recipients=[email], body=f"Welcome thanks for choosing us\nYour Login Credentials Are:\n Email Address: {email}\nPassword: {password}\n\nHospital Code {hcode}\n\n Do not share your password\n\n\nThank You...")

flash("Data Sent and Inserted Successfully", "warning")

return render\_template("addHosUser.html")

else:

flash("Login and try Again", "warning")

return render\_template("addHosUser.html")

**#hospital data info**

@app.route("/addhospitalinfo", methods=['POST', 'GET'])

def addhospitalinfo():

email = current\_user.email

posts = Hospitaluser.query.filter\_by(email=email).first()

code = posts.hcode

postsdata = Hospitaldata.query.filter\_by(hcode=code).first()

if request.method == "POST":

hcode = request.form.get('hcode')

hname = request.form.get('hname')

nbed = request.form.get('normalbed')

hbed = request.form.get('hicubeds')

ibed = request.form.get('icubeds')

vbed = request.form.get('ventbeds')

hcode = hcode.upper()

huser = Hospitaluser.query.filter\_by(hcode=hcode).first()

hduser = Hospitaldata.query.filter\_by(hcode=hcode).first()

if hduser:

flash("Data is already Present you can update it..", "primary")

return render\_template("hospitaldata.html")

if huser:

db.engine.execute(

f"INSERT INTO `hospitaldata` (`hcode`,`hname`,`normalbed`,`hicubed`,`icubed`,`vbed`) VALUES ('{hcode}','{hname}','{nbed}','{hbed}','{ibed}','{vbed}')")

flash("Data Is Added", "primary")

else:

flash("Hospital Code not Exist", "warning")

return render\_template("hospitaldata.html", postsdata=postsdata)

**#hospital data edit**

@app.route("/hedit/<string:id>", methods=['POST', 'GET'])

@login\_required

def hedit(id):

posts = Hospitaldata.query.filter\_by(id=id).first()

if request.method == "POST":

hcode = request.form.get('hcode')

hname = request.form.get('hname')

nbed = request.form.get('normalbed')

hbed = request.form.get('hicubeds')

ibed = request.form.get('icubeds')

vbed = request.form.get('ventbeds')

hcode = hcode.upper()

db.engine.execute(

f"UPDATE `hospitaldata` SET `hcode` ='{hcode}',`hname`='{hname}',`normalbed`='{nbed}',`hicubed`='{hbed}',`icubed`='{ibed}',`vbed`='{vbed}' WHERE `hospitaldata`.`id`={id}")

flash("Slot Updated", "info")

return redirect("/addhospitalinfo")

return render\_template("hedit.html", posts=posts)

**#hospital data delete**

@app.route("/hdelete/<string:id>", methods=['POST', 'GET'])

@login\_required

def hdelete(id):

db.engine.execute(

f"DELETE FROM `hospitaldata` WHERE `hospitaldata`.`id`={id}")

flash("Date Deleted", "danger")

return redirect("/addhospitalinfo")

**#logout admin**

@app.route("/logoutadmin")

def logoutadmin():

session.pop('user')

flash("You are logout admin", "primary")

return redirect('/admin')

**#logout user**

@app.route('/logout')

@login\_required

def logout():

logout\_user()

flash("Logout SuccessFul", "warning")

return redirect(url\_for('login'))

**#patient bed booking**

@app.route("/slotbooking", methods=['POST', 'GET'])

@login\_required

def slotbooking():

query = db.engine.execute(f"SELECT \* FROM `hospitaldata` ")

if request.method=="POST":

srfid=request.form.get('srfid')

bedtype=request.form.get('bedtype')

hcode=request.form.get('hcode')

spo2=request.form.get('spo2')

pname=request.form.get('pname')

pphone=request.form.get('pphone')

paddress=request.form.get('paddress')

check2=Hospitaldata.query.filter\_by(hcode=hcode).first()

if not check2:

flash("Hospital Code not exist","warning")

code=hcode

dbb=db.engine.execute(f"SELECT \* FROM `hospitaldata` WHERE `hospitaldata`.`hcode`='{code}' ")

bedtype=bedtype

if bedtype=="NormalBed":

for d in dbb:

seat=d.normalbed

print(seat)

ar=Hospitaldata.query.filter\_by(hcode=code).first()

ar.normalbed=seat-1

db.session.commit()

elif bedtype=="HICUBed":

for d in dbb:

seat=d.hicubed

print(seat)

ar=Hospitaldata.query.filter\_by(hcode=code).first()

ar.hicubed=seat-1

db.session.commit()

elif bedtype=="ICUBed":

for d in dbb:

seat=d.icubed

print(seat)

ar=Hospitaldata.query.filter\_by(hcode=code).first()

ar.icubed=seat-1

db.session.commit()

elif bedtype=="VENTILATORBed":

for d in dbb:

seat=d.vbed

ar=Hospitaldata.query.filter\_by(hcode=code).first()

ar.vbed=seat-1

db.session.commit()

else:

pass

check=Hospitaldata.query.filter\_by(hcode=hcode).first()

if(seat>0 and check): res=Bookingpatient(srfid=srfid,bedtype=bedtype,hcode=hcode,spo2=spo2,pname=pname,pphone=pphone,paddress=paddress)

db.session.add(res)

db.session.commit()

flash("Slot is Booked , kindly Visit Hospital for Further Procedure","success")

else:

flash("Something Went Wrong","danger")

return render\_template("booking.html",query=query)

**#patient details**

@app.route("/pdetails",methods=['GET'])

@login\_required

def pdetails():

code=current\_user.srfid

print(code)

data=Bookingpatient.query.filter\_by(srfid=code).first()

return render\_template("details.html",data=data)

**#triggered data**

@app.route("/triggers")

def triggers():

query=Trig.query.all()

return render\_template("triggers.html",query=query)

**#testing whether db is connected or not**

@app.route("/test")

def test():

try:

a = Test.query.all()

print(a)

return 'My database is connected'

except Exception as e:

print(e)

return f'database not connected {e}'

app.run(debug=True)

**5.4 HTML FILES**

**5.4.1 Base.html (Home page)**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta content="width=device-width, initial-scale=1.0" name="viewport">

<title>{% block title %}{% endblock title %}</title>

<meta content="" name="description">

<meta content="" name="keywords">

<!-- Favicons -->

<link href="static/assets/img/favicon.png" rel="icon">

<link href="static/assets/img/apple-touch-icon.png" rel="apple-touch-icon">

<!-- Google Fonts -->

<link

href="https://fonts.googleapis.com/css?family=Open+Sans:300,300i,400,400i,600,600i,700,700i|Roboto:300,300i,400,400i,500,500i,600,600i,700,700i|Poppins:300,300i,400,400i,500,500i,600,600i,700,700i"

rel="stylesheet">

<!-- Vendor CSS Files -->

<link href="static/assets/vendor/fontawesome-free/css/all.min.css" rel="stylesheet">

<link href="static/assets/vendor/animate.css/animate.min.css" rel="stylesheet">

<link href="static/assets/vendor/aos/aos.css" rel="stylesheet">

<link href="static/assets/vendor/bootstrap/css/bootstrap.min.css" rel="stylesheet">

<link href="static/assets/vendor/bootstrap-icons/bootstrap-icons.css" rel="stylesheet">

<link href="static/assets/vendor/boxicons/css/boxicons.min.css" rel="stylesheet">

<link href="static/assets/vendor/glightbox/css/glightbox.min.css" rel="stylesheet">

<link href="static/assets/vendor/swiper/swiper-bundle.min.css" rel="stylesheet">

<!-- Template Main CSS File -->

<link href="static/assets/css/style.css" rel="stylesheet">

</head>

<body>

<!-- ======= Top Bar ======= -->

<div id="topbar" class="d-flex align-items-center fixed-top">

<div class="container d-flex align-items-center justify-content-center justify-content-md-between">

<div class="align-items-center d-none d-md-flex">

<i class="bi bi-clock"></i> 24\*7 Available

</div>

<div class="d-flex align-items-center">

<i class="bi bi-phone"></i> Call us now +91-9558955488

</div>

</div>

</div>

<!-- ======= Header ======= -->

<header id="header" class="fixed-top">

<div class="container d-flex align-items-center">

<a href="index.html" class="logo me-auto"><img src="static/assets/img/logo.png" alt=""></a>

<!-- Uncomment below if you prefer to use an image logo -->

<!-- <h1 class="logo me-auto"><a href="index.html">Medicio</a></h1> -->

<nav id="navbar" class="navbar order-last order-lg-0">

<ul>

<li><a class="nav-link scrollto " href="/">Home</a></li>

<li><a class="nav-link scrollto" href="/pdetails">Patient Details</a></li>

<li><a class="nav-link scrollto" href="/slotbooking">Available Beds</a></li>

<!-- <li><a class="nav-link scrollto" href="/triggers">Operations</a></li> -->

{% if current\_user.is\_authenticated and current\_user.hcode%}

<li><a class="nav-link scrollto" href="/triggers">Operations</a></li>

<li class="dropdown"><a href="/login"><span>Welcome {{current\_user.email}}</span> <i class="bi bi-chevron-down"></i></a>

<ul>

<li><a href="/addhospitalinfo">Add Hospital Data</a></li>

<li><a href="/logout">Logout</a></li>

</ul>

</li>

{% elif current\_user.is\_authenticated and current\_user.srfid %}

<li class="dropdown"><a href="/login"><span>Welcome {{current\_user.email}}</span> <i

class="bi bi-chevron-down"></i></a>

<ul>

<li><a href="/slotbooking">Book Slot</a></li>

<li><a href="/logout">Logout</a></li>

</ul>

</li>

{% else %}

<li class="dropdown"><a href="/login"><span>Sign In</span> <i class="bi bi-chevron-down"></i></a>

<ul>

<li><a href="/login">User Login</a></li>

<li><a href="/hospitallogin">Hospital Login</a></li>

<li><a href="/admin">Admin Login</a></li>

</ul>

</li>

{% endif %}

<li><a class="nav-link scrollto" href="#contact">Contact</a></li>

</ul>

<i class="bi bi-list mobile-nav-toggle"></i>

</nav><!-- .navbar -->

<a href="/slotbooking" class="appointment-btn scrollto"><span class="d-none d-md-inline">Book Bed</span>

Slot Now</a>

</div>

</header><!-- End Header -->

<!-- ======= Hero Section ======= -->

<section id="hero">

<div id="heroCarousel" data-bs-interval="5000" class="carousel slide carousel-fade" data-bs-ride="carousel">

<ol class="carousel-indicators" id="hero-carousel-indicators"></ol>

<div class="carousel-inner" role="listbox">

<!-- Slide 1 -->

<div class="carousel-item active" style="background-image: url(static/assets/img/slide/slide-1.jpg)">

<div class="container">

<h2>Welcome to <span>Covid center</span></h2>

<p>Bangalore bed booking System</p>

<a href="#about" class="btn-get-started scrollto">Read More</a>

</div>

</div>

<!-- Slide 2 -->

<div class="carousel-item" style="background-image: url(static/assets/img/slide/slide-2.jpg)">

<div class="container">

<h2>India</h2>

<p>Bangalore bed booking System.</p>

<a href="#about" class="btn-get-started scrollto">Read More</a>

</div>

</div>

</div>

<a class="carousel-control-prev" href="#heroCarousel" role="button" data-bs-slide="prev">

<span class="carousel-control-prev-icon bi bi-chevron-left" aria-hidden="true"></span>

</a>

<a class="carousel-control-next" href="#heroCarousel" role="button" data-bs-slide="next">

<span class="carousel-control-next-icon bi bi-chevron-right" aria-hidden="true"></span>

</a>

</div>

</section><!-- End Hero -->

<main id="main">

{% block body %}

{% endblock body %}

<footer id="footer">

<div class="container">

<div class="copyright">

&copy; Copyright <strong><span>AM Darshan</span></strong>. All Rights Reserved

</div>

<div class="credits">

Designed by <a href="https://github.com/AM-Darshan">AM Darshan</a>

</div>

</div>

</footer><!-- End Footer -->

<div id="preloader"></div>

<a href="#" class="back-to-top d-flex align-items-center justify-content-center"><i

class="bi bi-arrow-up-short"></i></a>

<!-- Vendor JS Files -->

<script src="static/assets/vendor/purecounter/purecounter.js"></script>

<script src="static/assets/vendor/aos/aos.js"></script>

<script src="static/assets/vendor/bootstrap/js/bootstrap.bundle.min.js"></script>

<script src="static/assets/vendor/glightbox/js/glightbox.min.js"></script>

<script src="static/assets/vendor/swiper/swiper-bundle.min.js"></script>

<script src="static/assets/vendor/php-email-form/validate.js"></script>

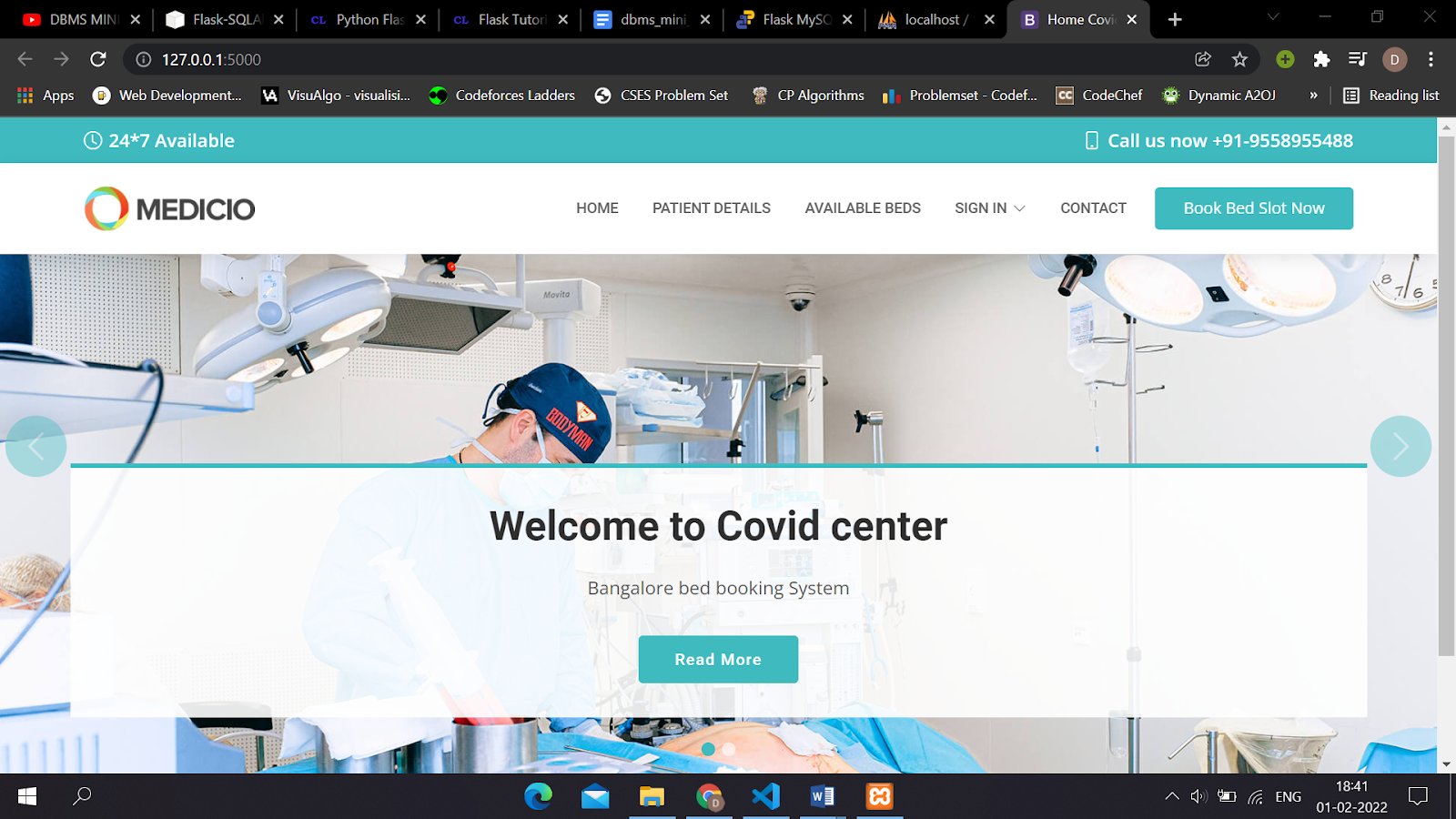
<!-- Template Main JS File -->

<script src="static/assets/js/main.js"></script>

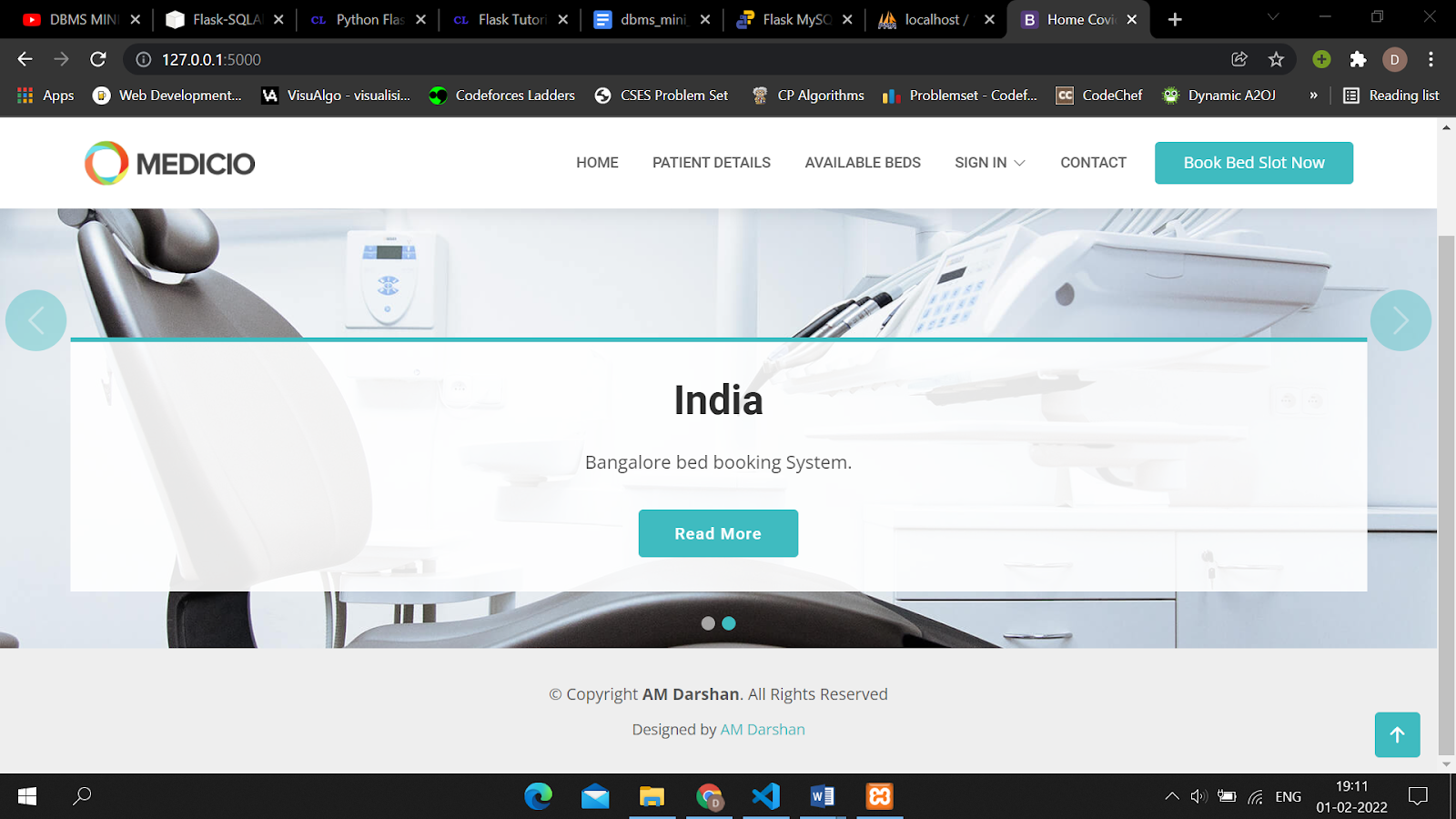
</body>

</html>

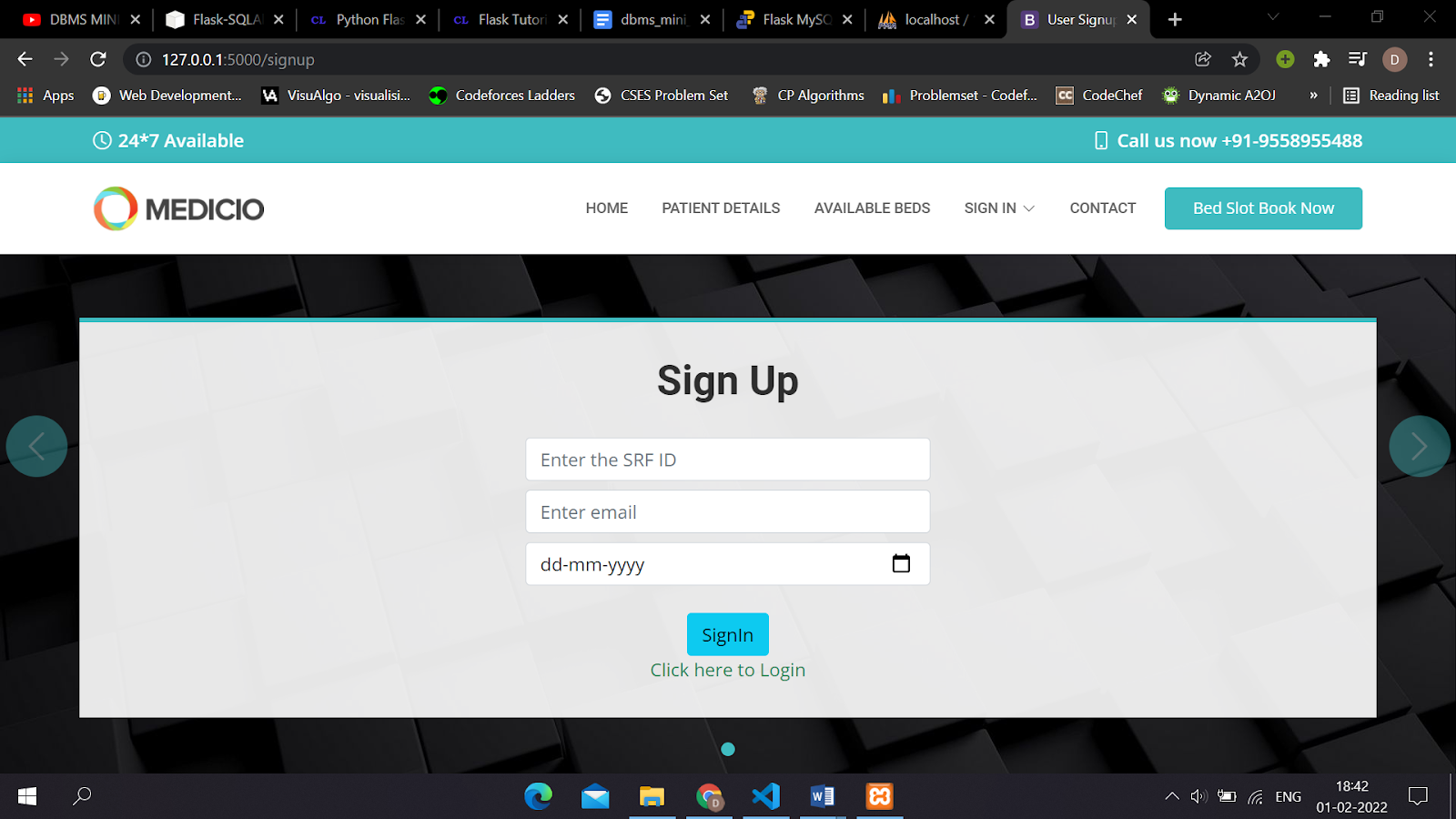
**Snapshots**

****

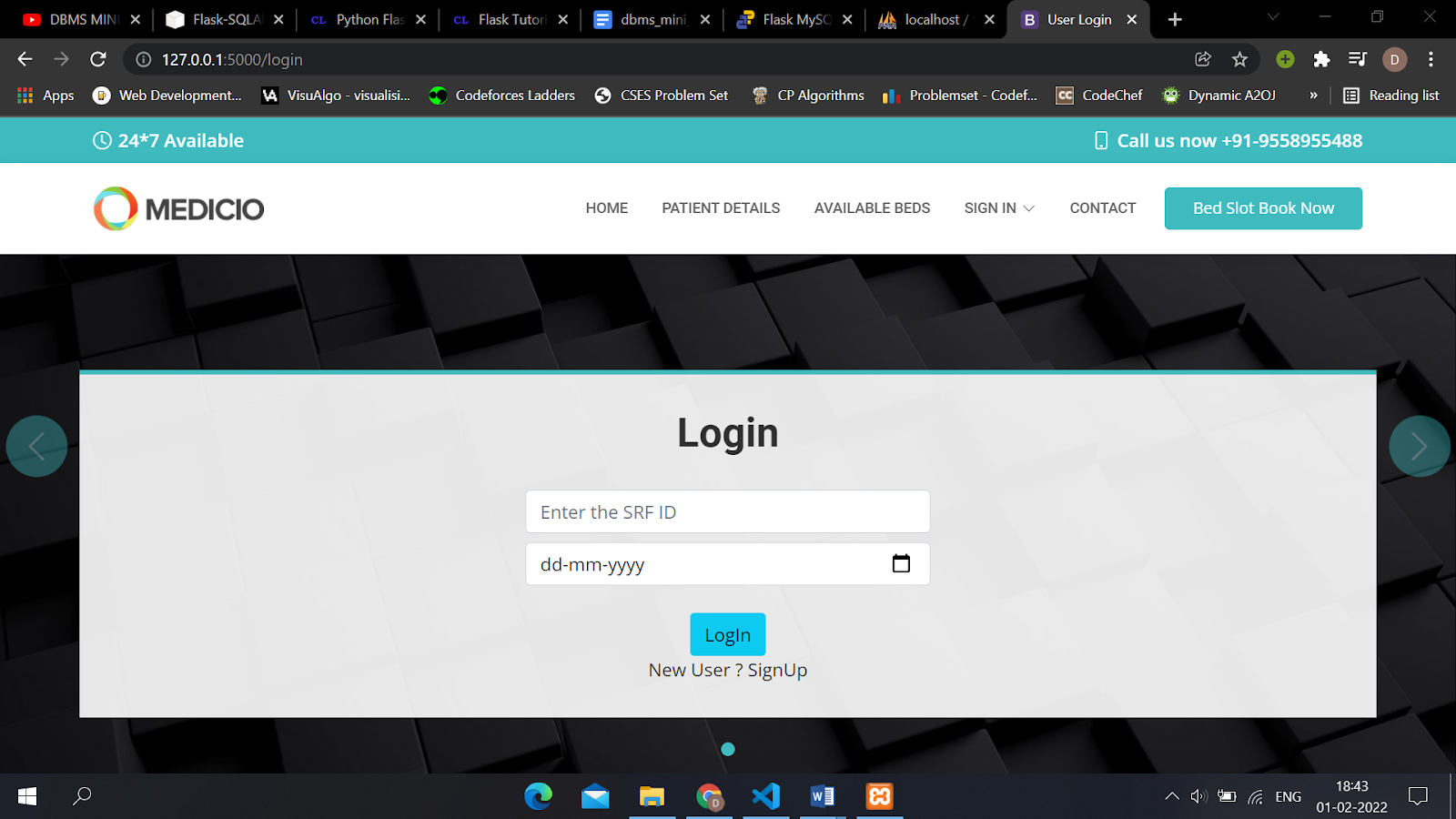
6.1 Home Page

****

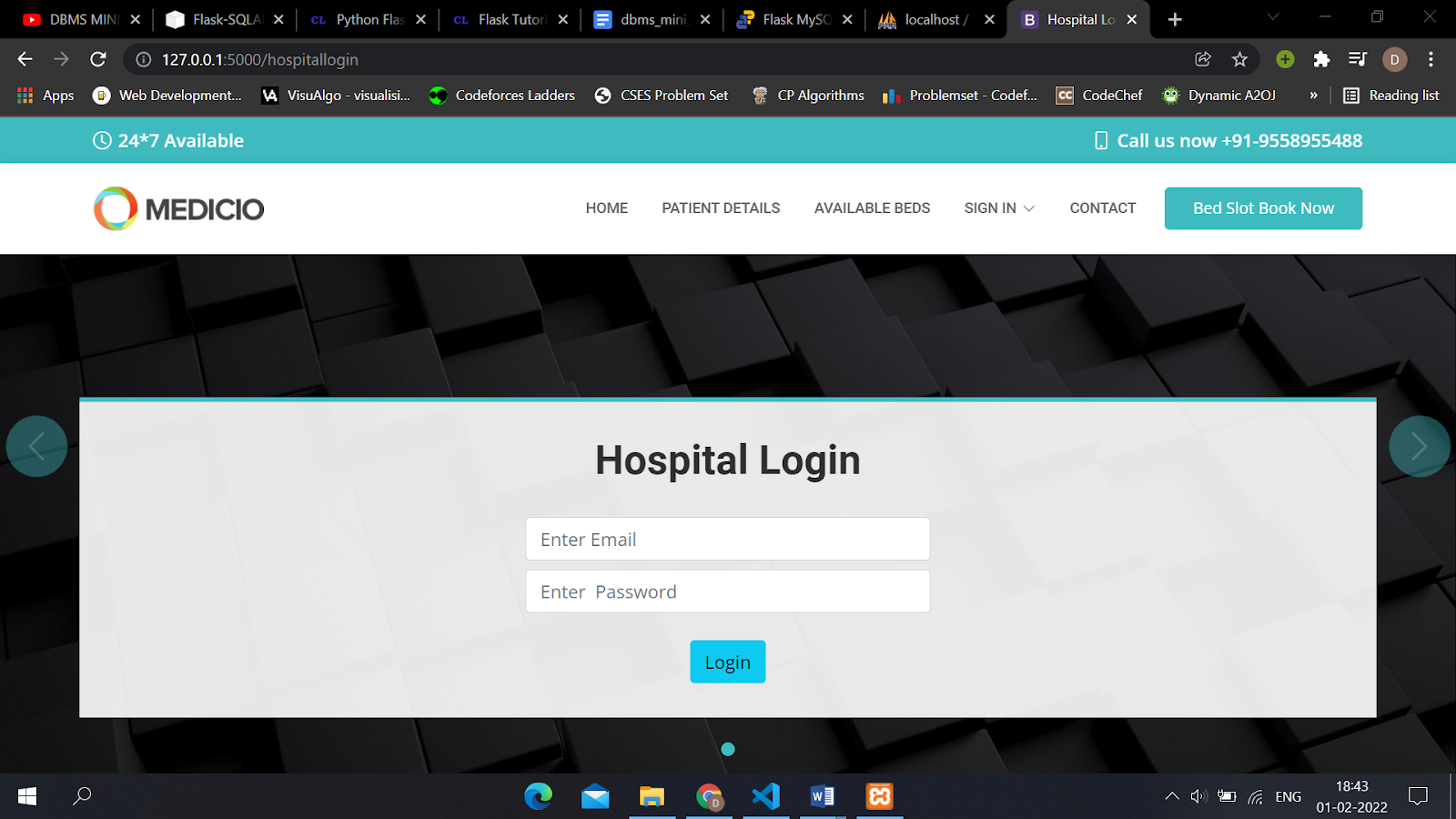
6.2 Home Page Slider

****

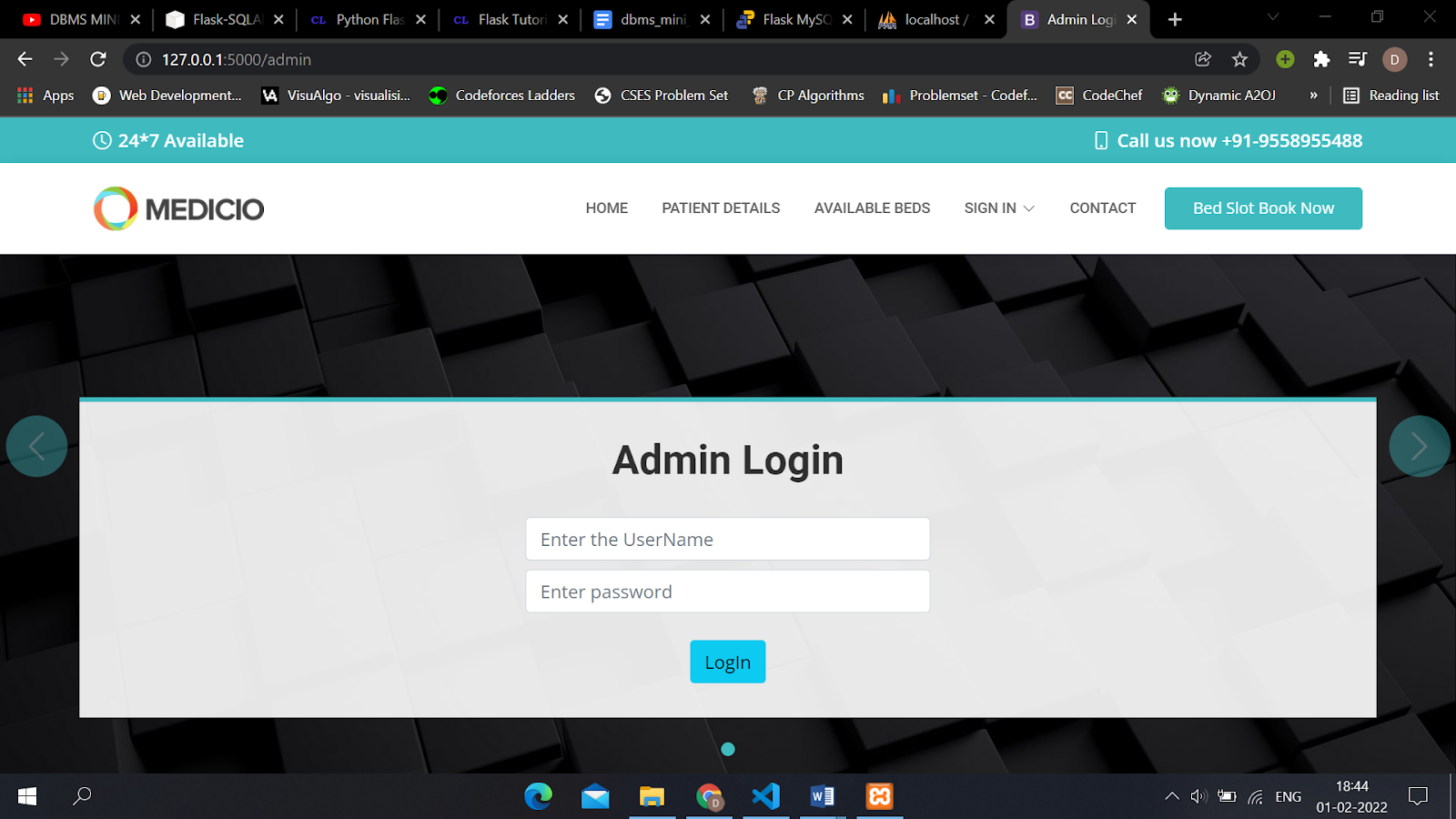
6.3 User Sign Page

****

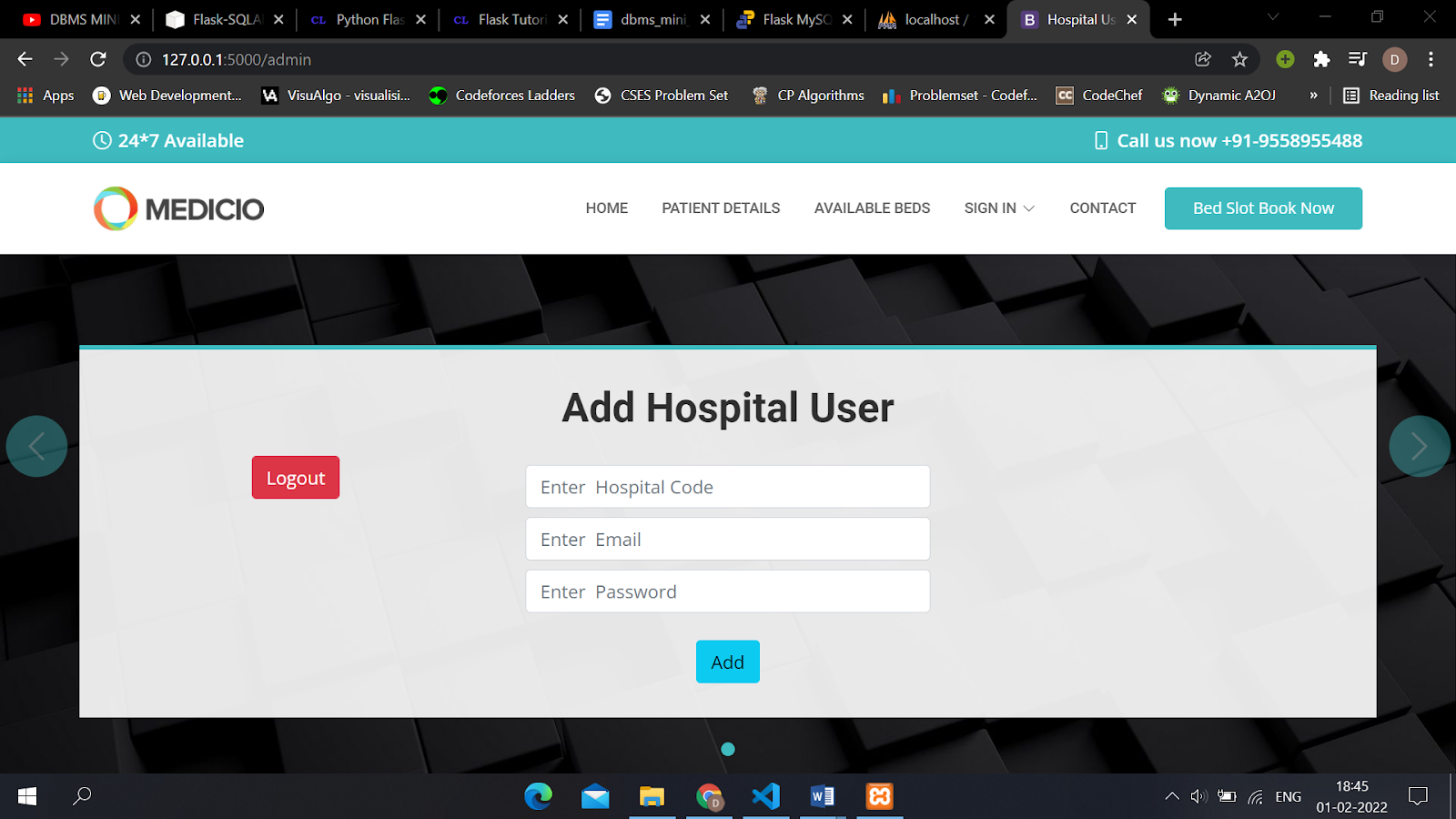
6.4 User login page

****

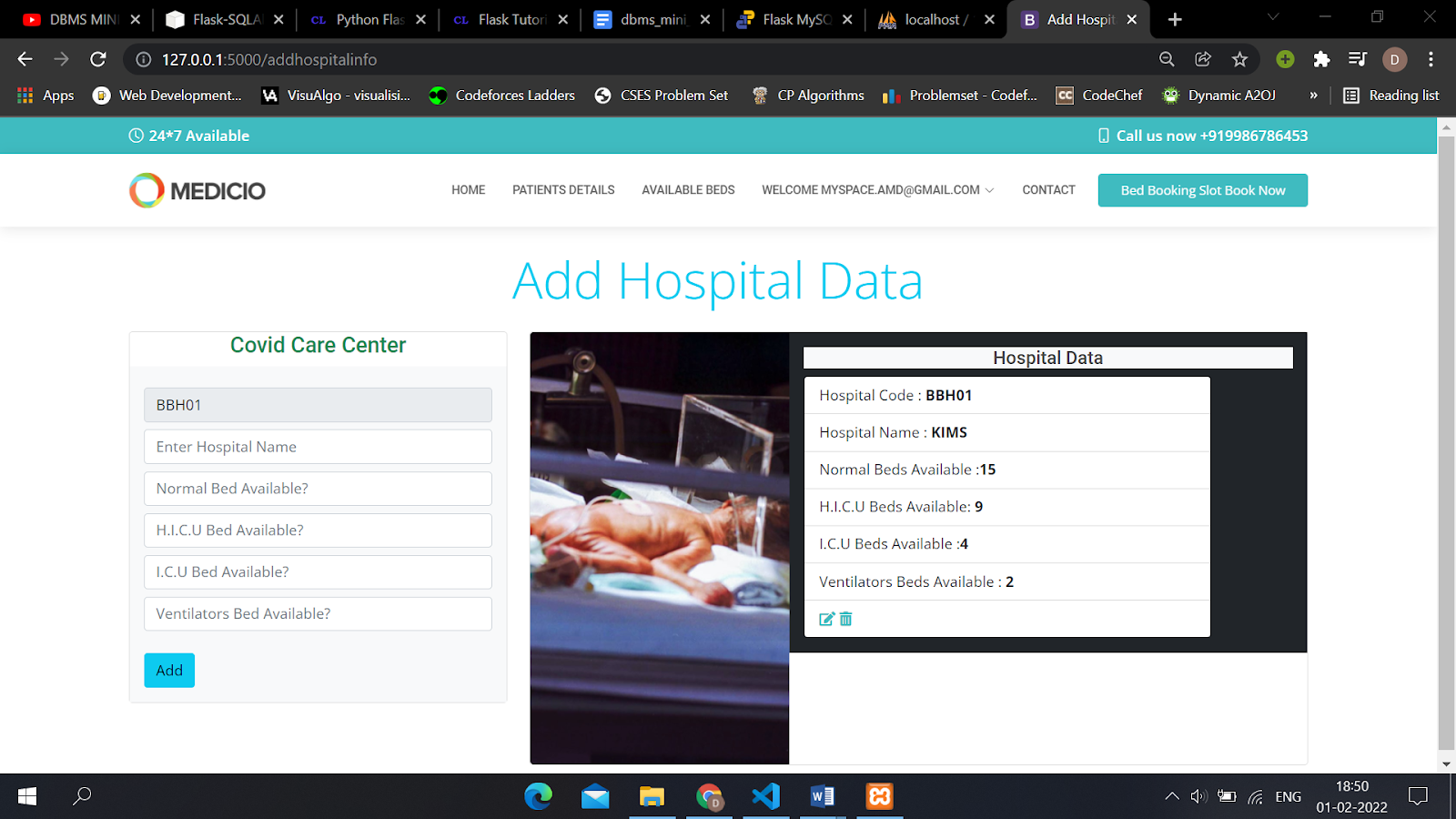
6.5 Hospital Login page

****

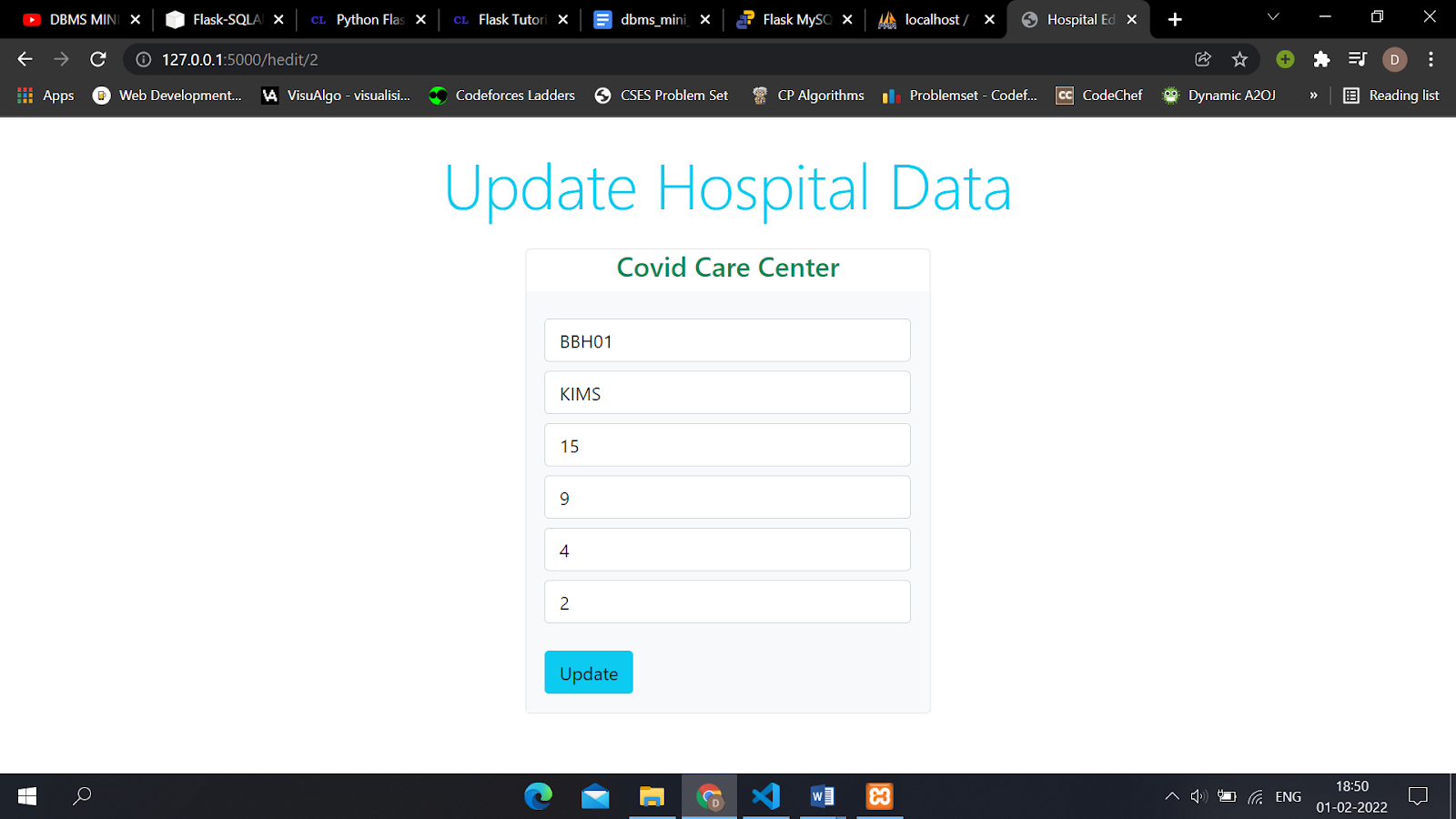
6.6 Admin Login Page

****

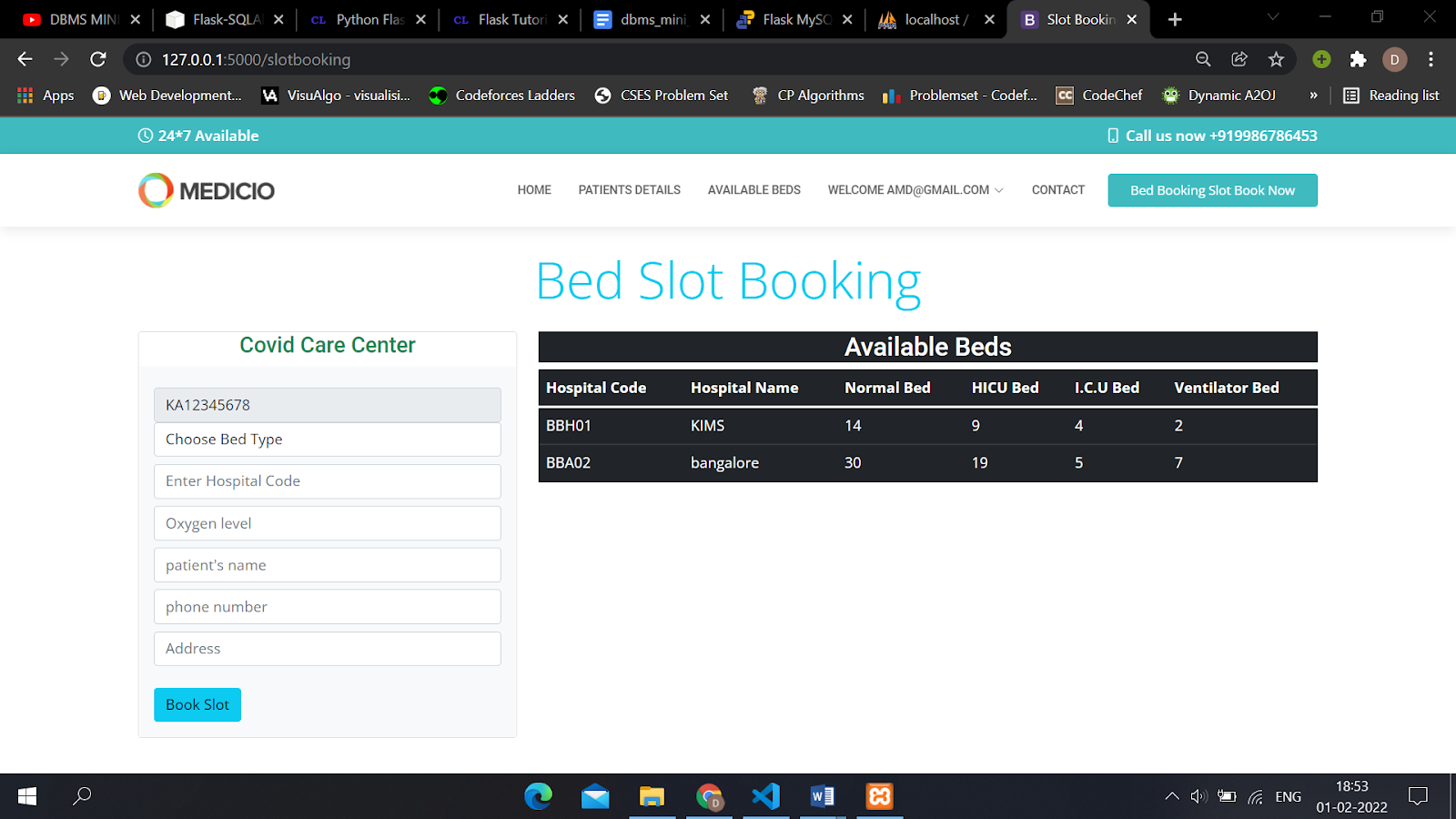
6.7 Add Hospital user page

****

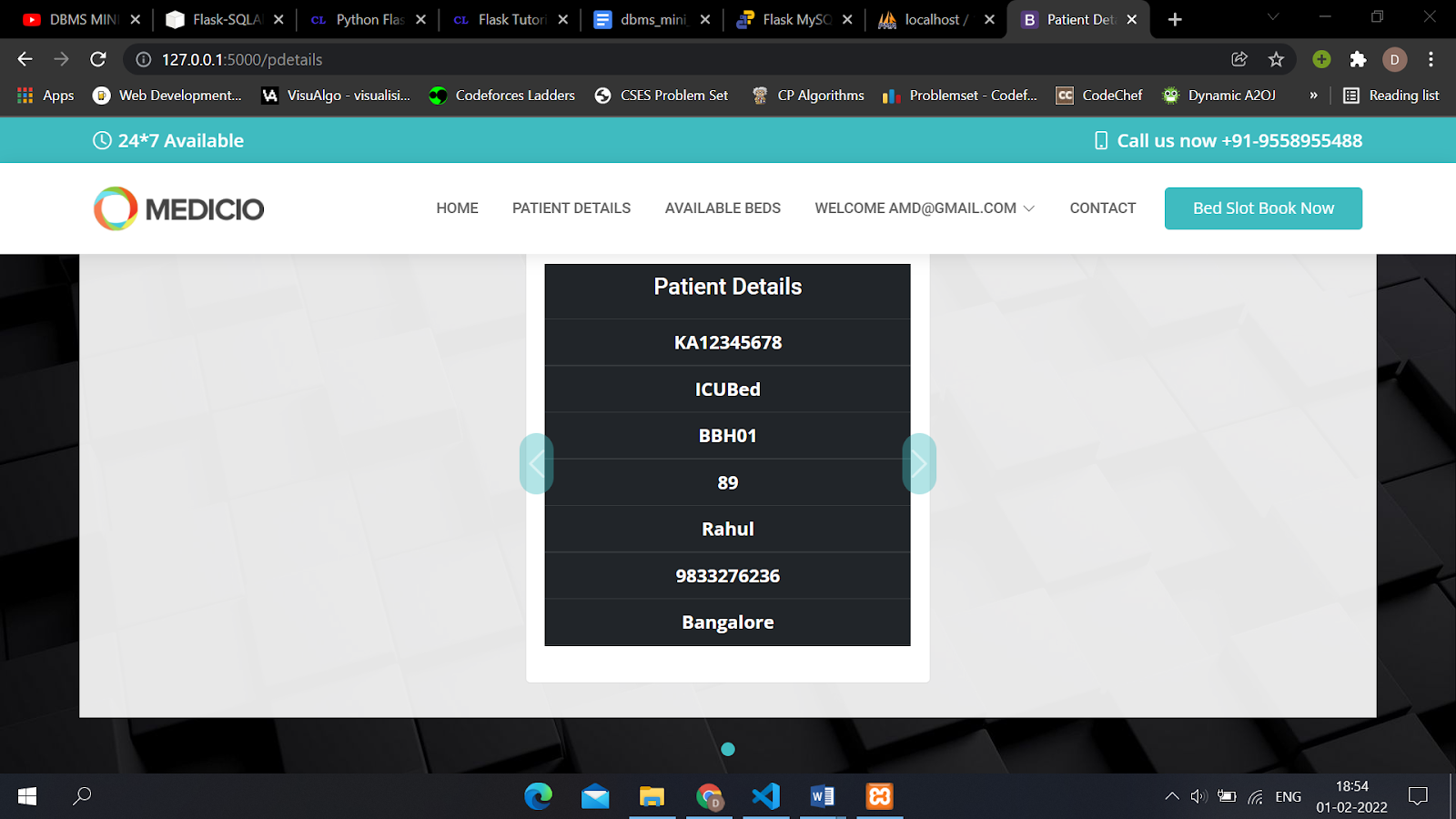
6.8 Add Hospital Data page

****

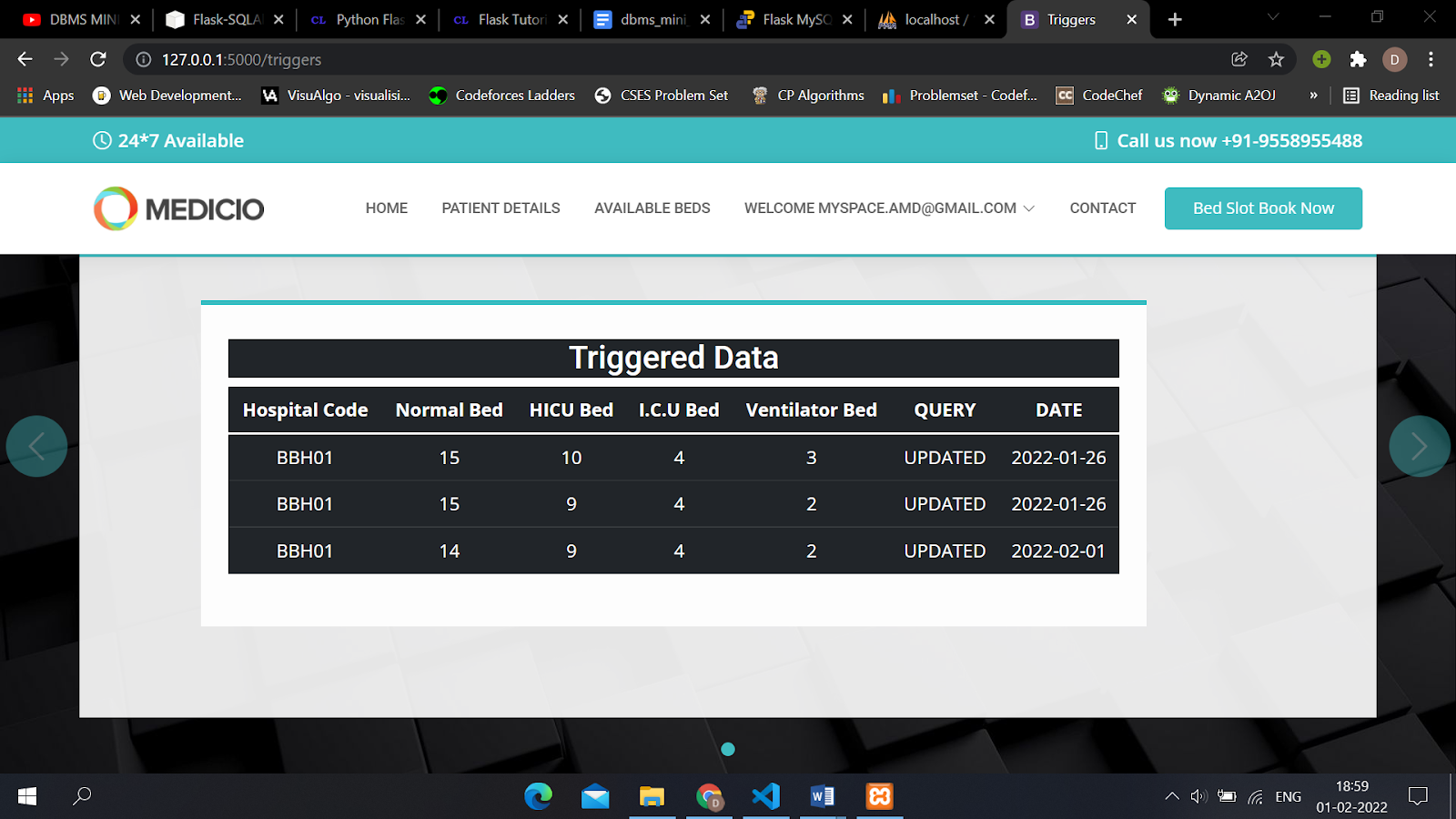
6.9 Hospital Data Update page



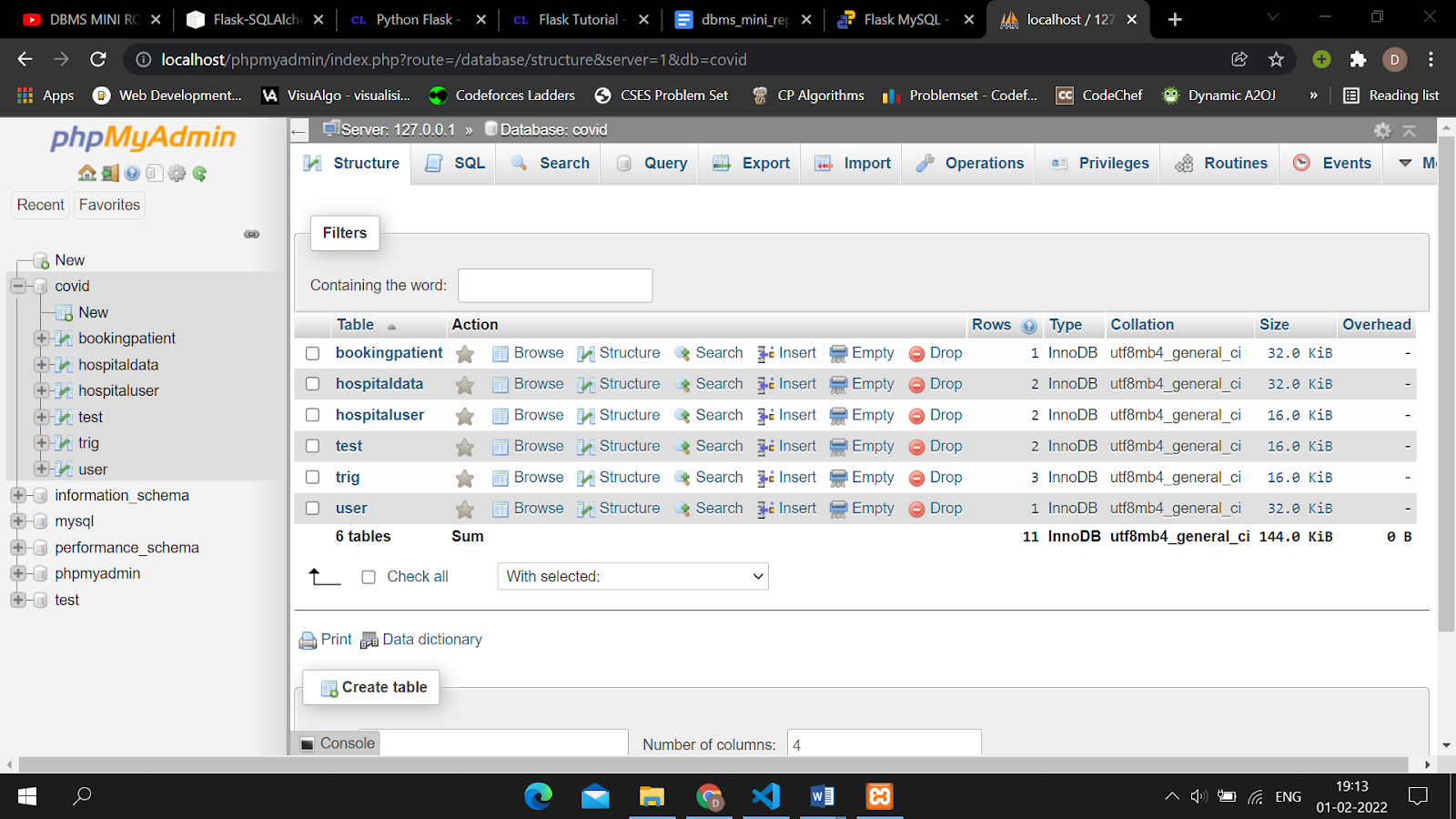
6.10 Bed Slot Booking page



6.11 Patient Detail page



6.12 Triggered Data page



6.13 Xampp server Database page



6.14 Email

**Applications**

* Patients can schedule services in comfort of their home.
* Quick and easy booking of beds for patients.
* Online scheduling system operate 24/7.
* Online Booking reduces the appointment gaps and hence immediate concern can be shown for the patient.
* User-friendly and error free experience for the user booking the bed slot

**Conclusion**

The project work titled “Covid Bed Slot Booking System” has been designed using Python wherein many user-friendly form controls have been added in order to make it a user interactive application. The system is developed in such a way that the user with common knowledge of computers can handle it easily. The system forms a general platform for building a most advanced bed slot booking systems.

So, the Covid Bed Slot Booking System is mainly used to list the various hospitals, to store the hospital details such as types of beds available, slot bookings, triggered data and patient details.