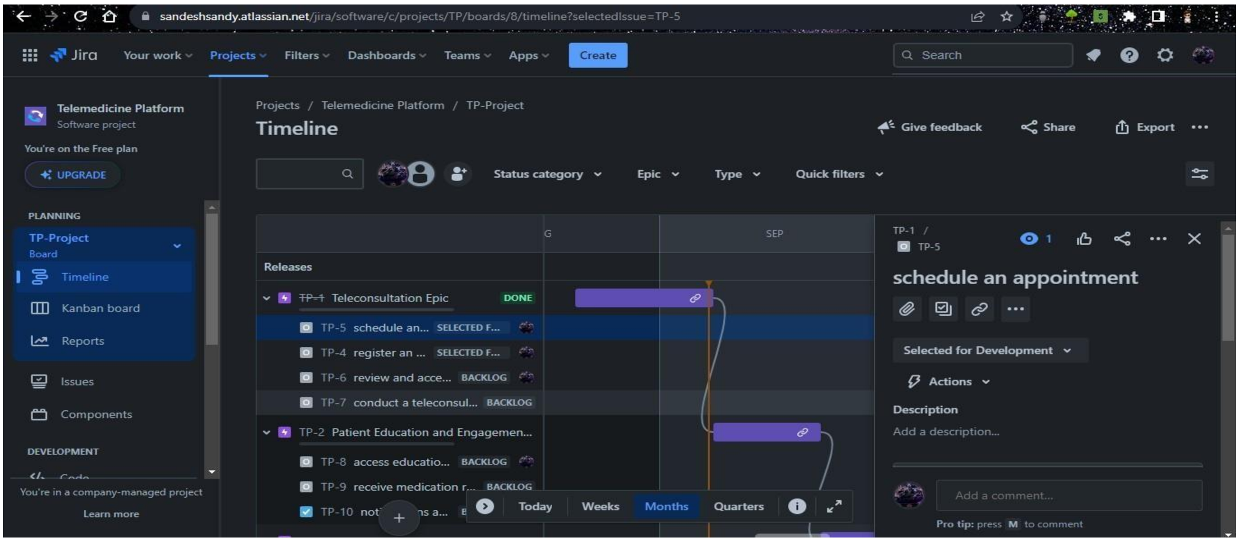
1) Use Jira Product Management tools create Epics, User Stories, Sprint, Backlogs etc for Telemedicine Platform.

→

Telemedicine Platform:

(Using JIRA SOFTWARE)



sandeshsandy.atlassian.net/jira/software/c/projects/TP/boards/8/timeline?selectedIssue=TP-5

Projects / Telemedicine Platform / TP-Project

Timeline

TP-1 ~~Teleconsultation Epic~~ DONE
TP-5 schedule an... SELECTED F...
TP-4 register an ... SELECTED F...
TP-6 review and acce... BACKLOG
TP-7 conduct a teleconsul... BACKLOG
TP-2 Patient Education and Engagemen...
TP-8 access educatio... BACKLOG
TP-9 receive medication r... BACKLOG
TP-10 not...

TP-1 / TP-5

schedule an appointment

Selected for Development

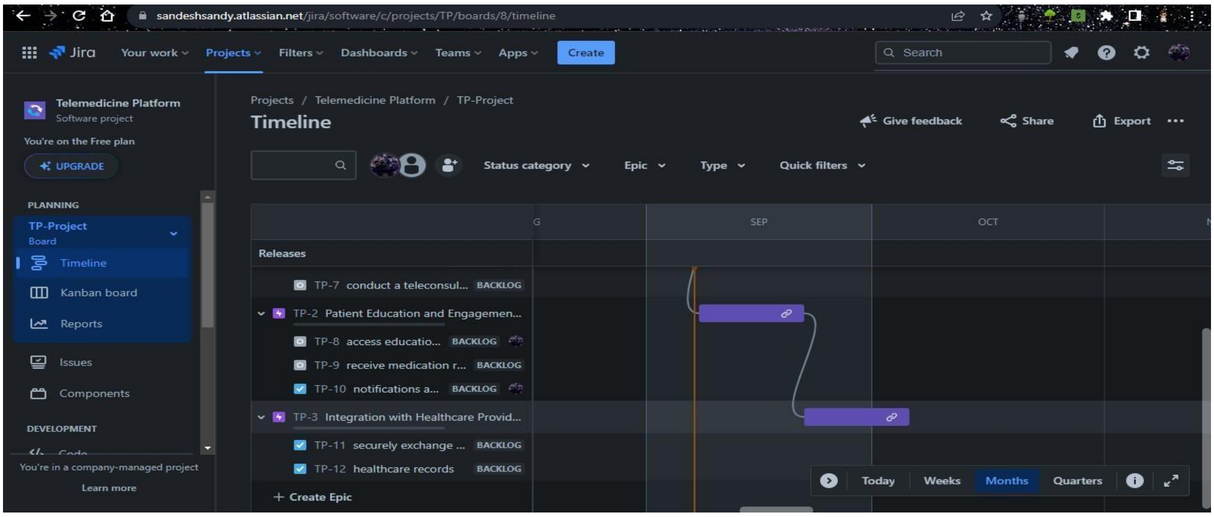
Description

Add a description...

Add a comment...

Pro tip: press M to comment

s



sandeshsandy.atlassian.net/jira/software/c/projects/TP/boards/8/timeline

Projects / Telemedicine Platform / TP-Project

Timeline

TP-7 conduct a teleconsul... BACKLOG
TP-2 Patient Education and Engagemen...
TP-8 access educatio... BACKLOG
TP-9 receive medication r... BACKLOG
TP-10 notifications a... BACKLOG
TP-3 Integration with Healthcare Provid...
TP-11 securely exchange ... BACKLOG
TP-12 healthcare records BACKLOG
+ Create Epic

Creating User Stories & Acceptance criteria:

- **Register an account:**

User Stories: As a patient, I want to register an account on the platform, so I can accesstelemedicine services.

Acceptance Criteria: User registration form with required fields, email verification, and password reset functionality.

- **schedule an appointment:**

User Stories: As a patient, I want to schedule an appointment with a healthcare provider for a teleconsultation.

Acceptance Criteria: User-friendly appointment booking interface, availability calendar, and confirmation email.

- **review and accept teleconsultation requests:**

User Stories: As a healthcare provider, I want to review and accept teleconsultation requests.

Acceptance Criteria: Provider dashboard with pending appointment requests, option to accept or decline requests.

- **access educational materials:**

User Stories: As a patient, I want to access educational materials on various healthtopics.

Acceptance Criteria: Health education content library with categories andsearch functionality.

- **receive medication reminders:**

User Stories: As a patient, I want to receive medication reminders for myprescribed treatments.

Acceptance Criteria: Medication reminder notifications via email or SMS, with adjustable schedules.

- **Notifications and reminders:**

User Stories: As a patient, I want to receive notifications and reminders for upcomingappointments.

Acceptance Criteria: Appointment reminders via email or SMS, with options for rescheduling.

- **securely exchange patient information:**

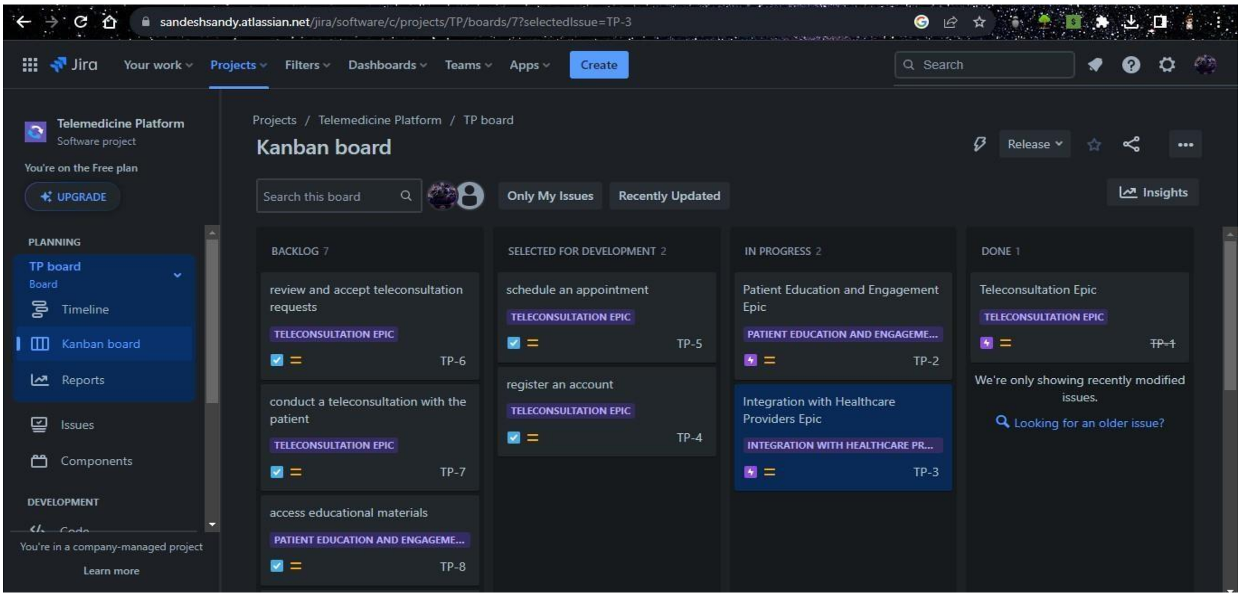
User Stories: As a healthcare provider, I want to securely exchange patient information with the telemedicine platform.

Acceptance Criteria: Encrypted data transfer, access controls, and compliance with data standards.

Backlog Refinement:

Backlog refinement in Jira is a collaborative and ongoing process where the productdevelopment team reviews, discusses, and refines items in the product backlog. Thegoal of backlog refinement is to ensure that the backlog contains well-defined, prioritized, and actionable user stories or items that are ready for development in upcoming sprints.

Here’s the Example of the **BACKLOG** refinement for telemedicine platform:



Kanban board

BACKLOG 7

- review and accept teleconsultation requests — TELECONSULTATION EPIC — TP-6
- conduct a teleconsultation with the patient — TELECONSULTATION EPIC — TP-7
- access educational materials — PATIENT EDUCATION AND ENGAGEME... — TP-8

SELECTED FOR DEVELOPMENT 2

- schedule an appointment — TELECONSULTATION EPIC — TP-5
- register an account — TELECONSULTATION EPIC — TP-4

IN PROGRESS 2

- Patient Education and Engagement Epic — PATIENT EDUCATION AND ENGAGEME... — TP-2
- Integration with Healthcare Providers Epic — INTEGRATION WITH HEALTHCARE PR... — TP-3

DONE 1

- Teleconsultation Epic — TELECONSULTATION EPIC — ~~TP-1~~

We're only showing recently modified issues.

Looking for an older issue?

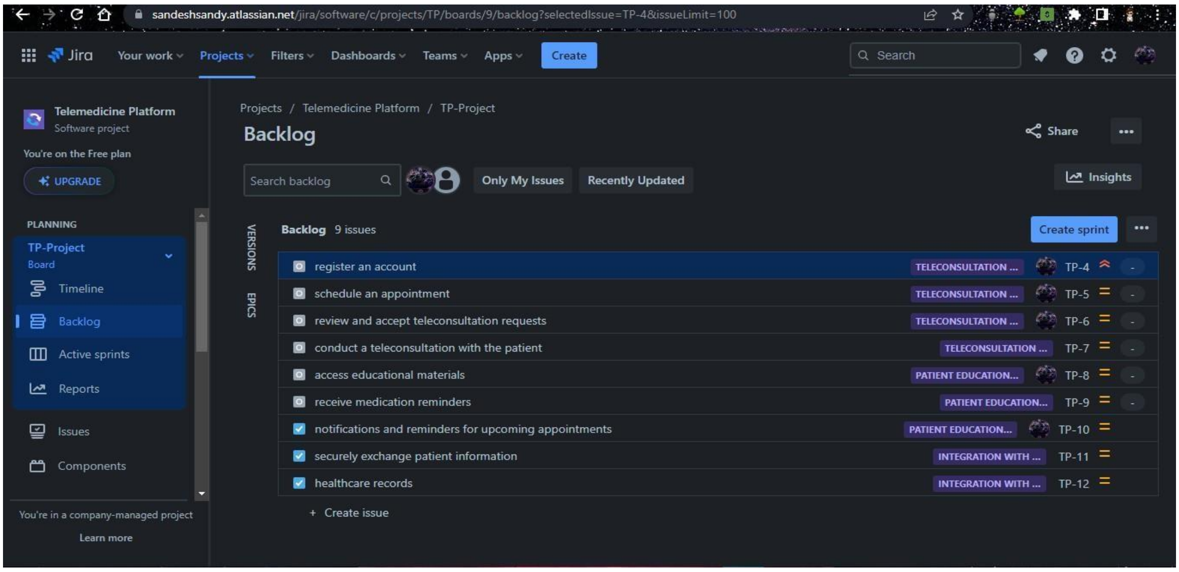
Sprint 1:

Steps to Create a sprint:

Step-1 Go to your project Backlog. Your backlogs appearStep-2: Click on create sprint.

Step-3: Add the related stories and tasks in the created sprint.Step-4: Click on start sprint.

Step-5: Your Sprint Will be created.

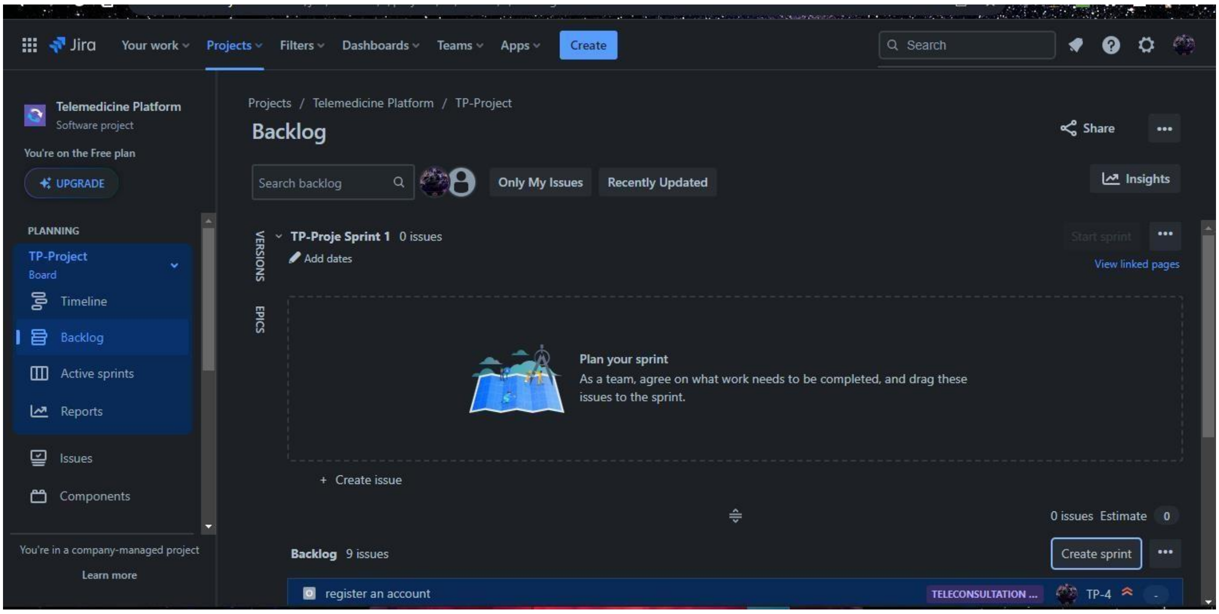


Backlog

Backlog 9 issues

□	register an account	TELECONSULTATION ...	TP-4
□	schedule an appointment	TELECONSULTATION ...	TP-5
□	review and accept teleconsultation requests	TELECONSULTATION ...	TP-6
□	conduct a teleconsultation with the patient	TELECONSULTATION ...	TP-7
□	access educational materials	PATIENT EDUCATION...	TP-8
□	receive medication reminders	PATIENT EDUCATION...	TP-9
☑	notifications and reminders for upcoming appointments	PATIENT EDUCATION...	TP-10
☑	securely exchange patient information	INTEGRATION WITH ...	TP-11
☑	healthcare records	INTEGRATION WITH ...	TP-12

+ Create issue



TP-Proje Sprint 1 0 issues

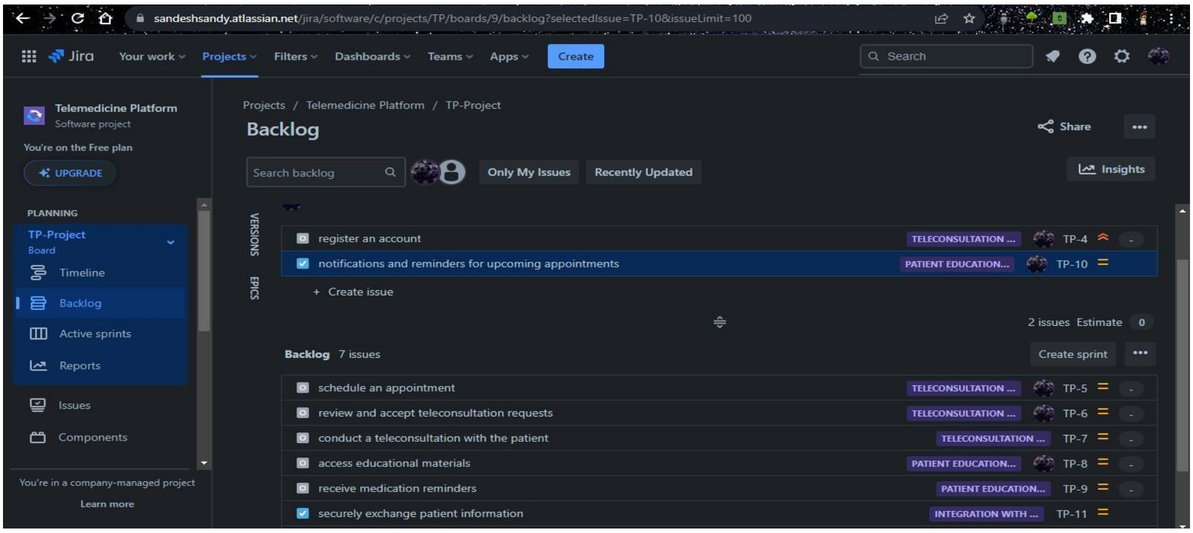
Plan your sprint

As a team, agree on what work needs to be completed, and drag these issues to the sprint.

0 issues Estimate 0

Backlog 9 issues

register an account	TELECONSULTATION ...	TP-4	-

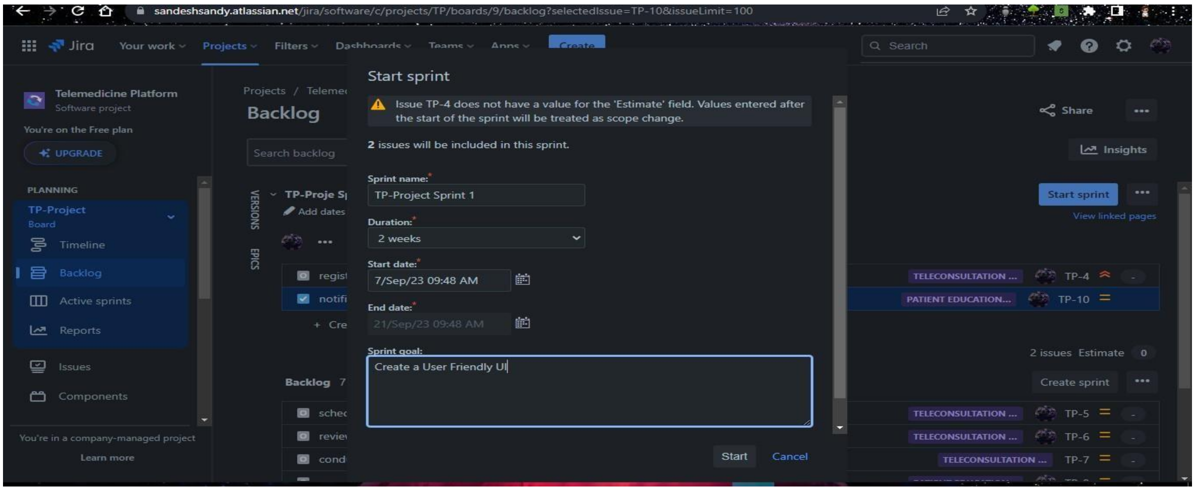


register an account	TELECONSULTATION ...	TP-4	-
☑ notifications and reminders for upcoming appointments	PATIENT EDUCATION...	TP-10	

2 issues Estimate 0

Backlog 7 issues

schedule an appointment	TELECONSULTATION ...	TP-5	-
review and accept teleconsultation requests	TELECONSULTATION ...	TP-6	-
conduct a teleconsultation with the patient	TELECONSULTATION ...	TP-7	-
access educational materials	PATIENT EDUCATION...	TP-8	-
receive medication reminders	PATIENT EDUCATION...	TP-9	-
☑ securely exchange patient information	INTEGRATION WITH ...	TP-11	



Start sprint

Issue TP-4 does not have a value for the 'Estimate' field. Values entered after the start of the sprint will be treated as scope change.

2 issues will be included in this sprint.

Sprint name:* TP-Project Sprint 1

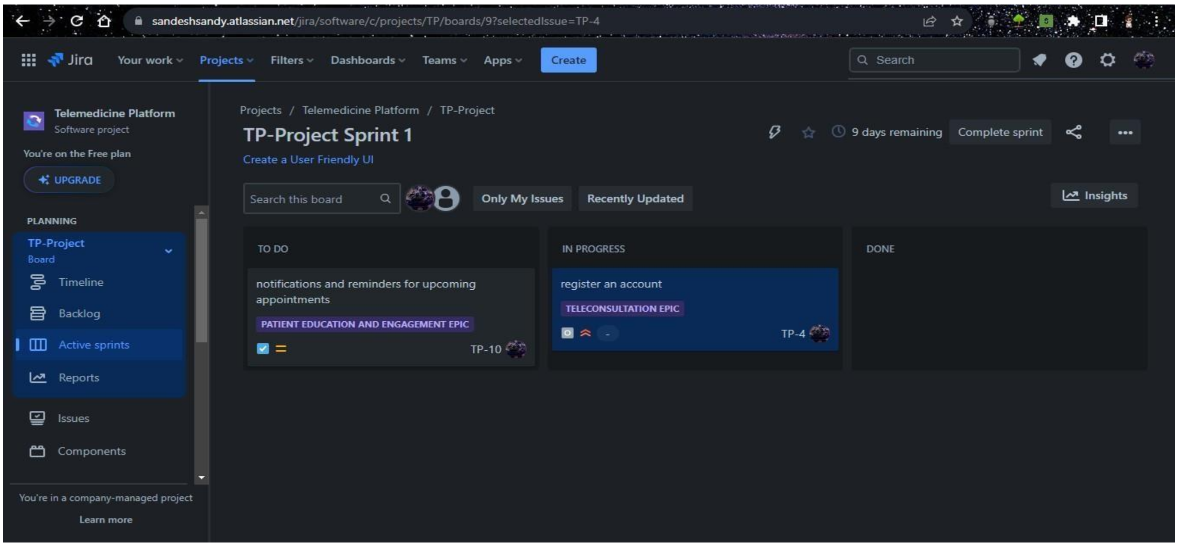
Duration:* 2 weeks

Start date:* 7/Sep/23 09:48 AM

End date:* 21/Sep/23 09:48 AM

Sprint goal: Create a User Friendly UI

Start Cancel



TP-Project Sprint 1

Create a User Friendly UI

TO DO

notifications and reminders for upcoming appointments

PATIENT EDUCATION AND ENGAGEMENT EPIC

TP-10

IN PROGRESS

register an account

TELECONSULTATION EPIC

TP-4

DONE

2) Create a Simple Registration Form with all the Required Input fields, using HTML, CSS. And make sure that only reviewed code is pushed into master branch.

→ Index.html:

```
<!DOCTYPE html>
<!---Coding By CodingLab | www.codinglabweb.com--->
<html lang="en">
 <head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta http-equiv="X-UA-Compatible" content="ie=edge" />
  <title>Registration Form in HTML CSS</title>
  <!---Custom CSS File--->
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <section class="container">
   <header>Registration Form</header>
   <form action="#" class="form">
    <div class="input-box">
     <label>Full Name</label>
     <input type="text" placeholder="Enter full name" required />
    </div>

    <div class="input-box">
     <label>Email Address</label>
     <input type="text" placeholder="Enter email address" required />
    </div>
```

```
<div class="column">
  <div class="input-box">
    <label>Phone Number</label>
    <input type="number" placeholder="Enter phone number" required />
  </div>
  <div class="input-box">
    <label>Birth Date</label>
    <input type="date" placeholder="Enter birth date" required />
  </div>
</div>
<div class="gender-box">
  <h3>Gender</h3>
  <div class="gender-option">
    <div class="gender">
      <input type="radio" id="check-male" name="gender" checked />
      <label for="check-male">male</label>
    </div>
    <div class="gender">
      <input type="radio" id="check-female" name="gender" />
      <label for="check-female">Female</label>
    </div>
    <div class="gender">
      <input type="radio" id="check-other" name="gender" />
      <label for="check-other">prefer not to say</label>
    </div>
  </div>
</div>
<div class="input-box address">
  <label>Address</label>
  <input type="text" placeholder="Enter street address" required />
  <input type="text" placeholder="Enter street address line 2" required />
  <div class="column">
    <div class="select-box">
      <select>
        <option hidden>Country</option>
        <option>America</option>
        <option>Japan</option>
        <option>India</option>
        <option>Nepal</option>
      </select>
    </div>
    <input type="text" placeholder="Enter your city" required />
  </div>
  <div class="column">
    <input type="text" placeholder="Enter your region" required />
    <input type="number" placeholder="Enter postal code" required />
  </div>
```

```
                  </div>
                  <button>Submit</button>
                </form>
              </section>
            </body>
          </html>
```

Style.css:

```
@import
url("https://fonts.googleapis.com/css2?family=Poppins:wght@200;300;400;500;600;700&disp
lay=swap");
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
  font-family: "Poppins", sans-serif;
}
body {
  min-height: 100vh;
  display: flex;
  align-items: center;
  justify-content: center;
  padding: 20px;
  background: rgb(130, 106, 251);
}
.container {
  position: relative;
  max-width: 700px;
  width: 100%;
  background: #fff;
  padding: 25px;
  border-radius: 8px;
  box-shadow: 0 0 15px rgba(0, 0, 0, 0.1);
}
.container header {
  font-size: 1.5rem;
  color: #333;
  font-weight: 500;
  text-align: center;
}
.container .form {
  margin-top: 30px;
}
.form .input-box {
  width: 100%;
  margin-top: 20px;
}
.input-box label {
  color: #333;
}
.form :where(.input-box input, .select-box) {
  position: relative;
  height: 50px;
  width: 100%;
  outline: none;
  font-size: 1rem;
  color: #707070;
  margin-top: 8px;
  border: 1px solid #ddd;
  border-radius: 6px;
  padding: 0 15px;
}
.input-box input:focus {
  box-shadow: 0 1px 0 rgba(0, 0, 0, 0.1);
}
.form .column {
  display: flex;
  column-gap: 15px;
}
.form .gender-box {
  margin-top: 20px;
}
.gender-box h3 {
  color: #333;
  font-size: 1rem;
  font-weight: 400;
  margin-bottom: 8px;
}
.form :where(.gender-option, .gender) {
  display: flex;
  align-items: center;
  column-gap: 50px;
  flex-wrap: wrap;
}
.form .gender {
  column-gap: 5px;
}
.gender input {
  accent-color: rgb(130, 106, 251);
}
.form :where(.gender input, .gender label) {
  cursor: pointer;
}
.gender label {
  color: #707070;
}
.address :where(input, .select-box) {
  margin-top: 15px;
}
```

```
.select-box select {
  height: 100%;
  width: 100%;
  outline: none;
  border: none;
  color: #707070;
  font-size: 1rem;
}
.form button {
  height: 55px;
  width: 100%;
  color: #fff;
  font-size: 1rem;
  font-weight: 400;
  margin-top: 30px;
  border: none;
  cursor: pointer;
  transition: all 0.2s ease;
  background: rgb(130, 106, 251);
}
.form button:hover {
  background: rgb(88, 56, 250);
}
/*Responsive*/
@media screen and (max-width: 500px) {
  .form .column {
    flex-wrap: wrap;
  }
  .form :where(.gender-option, .gender) {
    row-gap: 15px;
  }
}
```

3) Create a Login / Registration Form with Credentials using HTML, CSS, JS. And make sure that only reviewed code is pushed into master branch.

→

Index.html:

```
<!DOCTYPE html>

<html>

   <head>

      <meta charset="utf-8">

      <title>Login + Firebase Database</title>

       <!-- Cool Google Fonts -->

      <link rel="preconnect" href="https://fonts.gstatic.com">

      <link href="https://fonts.googleapis.com/css2?family=Montserrat&display=swap"
rel="stylesheet">

      <link
href="https://fonts.googleapis.com/css2?family=Montserrat:wght@900&display=swap"
rel="stylesheet">

      <link
href="https://fonts.googleapis.com/css2?family=Montserrat:wght@500&display=swap"
rel="stylesheet">

      <link href="https://fonts.googleapis.com/css2?family=Bowlby+One+SC&display=swap"
rel="stylesheet">

      <!-- Our stylesheet -->

      <link rel="stylesheet" type="text/css" href="style.css">

   </head>

   <body>

      <div id="content_container">

         <div id="form_container">

            <div id="form_header_container">

               <h2 id="form_header"> Login / Registration Form </h2>

            </div>


            <div id="form_content_container">

               <div id="form_content_inner_container">
            <input type="text" id="full_name" placeholder="Full name">
            <input type="email" id="email" placeholder="Email">
            <input type="password" id="password" placeholder="Enter Password">
            <div id="button_container">
              <button onclick="login()">Login</button>
              <button onclick="register()">Register</button>
            </div>

          </div>
        </div>
      </div>
    </div>
  </body>
  <script src="https://www.gstatic.com/firebasejs/8.6.8/firebase-app.js"></script>

  <script src="https://www.gstatic.com/firebasejs/8.6.8/firebase-auth.js"></script>
  <script src="https://www.gstatic.com/firebasejs/8.6.8/firebase-database.js"></script>
  <script src="app.js"></script>
</html>
```

Style.css:

```
html, body {
  width: 100%;
  height: 100%;
  background: -webkit-linear-gradient(25deg, #FFBE0B, #FB5607, #FF006E, #8338EC,
#3A86FF);
  display: flex;
  justify-content: center;
  align-items: center;
}

* {
  margin: 0;
```

```
    padding: 0;
    box-sizing: border-box;
    border: none;
    outline: none;
}
#content_container {
    width: 30%;
    height: 50%;
}

#form_container {
    width: 100%;
    height: 100%;
    background-color: #370617;
    box-shadow: 0 0 50px -20px #000;
    border-radius: 2%;
    overflow: hidden;
}

#form_header_container {
    width: 100%;
    height: 5%;
    display: flex;
    justify-content: center;
    align-items: center;
    float: left;
    padding: 20px;
    padding-bottom: 30px;
    padding-top: 30px;
    border-bottom: 1px solid transparent;
    border-image: -webkit-linear-gradient(25deg, #FFBE0B, #FB5607, #FF006E, #8338EC,
#3A86FF) 20;
```

```
   background: #000;
}

#form_header {
   display: inline-block;
   font-size: 15px;
   font-family: "Bowlby One SC";
   font-weight: 900;
   text-transform: uppercase;
   letter-spacing: 1px;
   background: -webkit-linear-gradient(25deg, #FFBE0B, #FB5607, #FF006E, #8338EC,
#3A86FF);
     -webkit-background-clip: text;
     -webkit-text-fill-color: transparent;
}

#form_content_container {
   width: 100%;
   height: 90%;
   float: left;
   display: flex;
   justify-content: center;
   align-items: center;
   padding-top: 30px;
}

#form_content_inner_container {
   width: 75%;
   height: 100%;
   float: left;
}
```

```
input {
    width: 100%;
    height: 40px;
    padding-left: 10px;
    margin-bottom: 20px;
    background: #000;
    font-family: Montserrat;
    font-weight: 500;
    color: #fff;
    font-size: 12px;
    border-bottom: 2px solid transparent;
    border-top-left-radius: 2%;
    border-top-right-radius: 2%;
    border-image: -webkit-linear-gradient(25deg, #FFBE0B, #FB5607, #FF006E, #8338EC, #3A86FF) 1;
}

#button_container {
    width: 100%;
    height: 10%;

    background-image: linear-gradient(80deg, #FFBE0B, #FB5607 50%, #FF006E 50%, #8338EC);
    color: #fff;

    float: left;

    margin-top: 5px;
}

#button_container button {
    width: 50%;
    height: 100%;
    float: left;

    background: transparent;
    color: inherit;

    font-family: Montserrat;
    letter-spacing: 1px;
    font-weight: 900;
    font-size: 12px;

    cursor: pointer;

    display: flex;
    justify-content: center;
    align-items: center;
}
```

App.js:

```
let firebaseConfig = {
  apiKey: "",
  authDomain: "",
  projectId: "",
  storageBucket: "",
  messagingSenderId: "",
  appId: ""
};
firebase.initializeApp(firebaseConfig);
const auth = firebase.auth()
const database = firebase.database()

function register () {
    let email = document.getElementById('email').value;
    let password = document.getElementById('password').value;
    let full_name = document.getElementById('full_name').value;

  if (validate_email(email) == false || validate_password(password) == false) {
    alert('Email or Password is Outta Line!!')
    return
  }
  if (validate_field(full_name) == false) {
    alert('One or More Extra Fields is Outta Line!!')
    return
  }

  auth.createUserWithEmailAndPassword(email, password)
  .then(function() {
    let user = auth.currentUser

    let database_ref = database.ref()
    const currentDate = new Date().toDateString();
    // Create User data
    let user_data = {
      email : email,
      full_name : full_name,
      last_login : currentDate
    }

    database_ref.child('users/' + user.uid).set(user_data)

    alert('User Created!!')
  })
  .catch(function(error) {
    let error_code = error.code
    let error_message = error.message
    alert(error_code, error_message)
   })
  }

  function login () {
    let email = document.getElementById('email').value
    let password = document.getElementById('password').value

   if (validate_email(email) == false || validate_password(password) == false) {
    alert('Email or Password is Outta Line!!')
    return
   }

   auth.signInWithEmailAndPassword(email, password)
   .then(function() {
    let user = auth.currentUser

    let database_ref = database.ref()

    let user_data = {
     last_login : Date.now()
    }
    database_ref.child('users/' + user.uid).update(user_data)
    alert('User Logged In!!')

   })
   .catch(function(error) {
    let error_code = error.code
    let error_message = error.message
    alert(error_code, error_message)
   })
  }
```

```
}

function validate_email(email) {
  let expression = /^[^@]+@\w+(\.\w+)+\w$/
  if (expression.test(email) == true) {
    return true
  } else {
    return false
  }
}

function validate_password(password) {
  if (password < 6) {
    return false
  } else {
    return true
  }
}

function validate_field(field) {
  if (field == null) {
    return false
  }
  if (field.length <= 0) {
    return false
  } else {
    return true
  }
}
```

4) Create a Weather App which shows the Weather report based on city name mentioned Using HTML, CSS, JS. And make sure only reviewed code is commited to master branch

Index.html:

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>weather App</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <main>
    <h1>Weather App</h1>
    <div class="row">
     <form action="">
      <input
       type="search"
       id="search"
       placeholder="Search Your City Name"
      />
     </form>
    </div>
    <div class="row" id="weather">


    </div>
   </main>
   <script src="app.js"></script>
</body>
</html>
```

Style.css:

```
* {
  padding: 0px;
  margin: 0px;
  box-sizing: border-box;
  font-family: Verdana, Geneva, Tahoma, sans-serif;
}
body{
  overflow-x:hidden;


}
main {
  width: 100%;
  height: 100vh;
  background: linear-gradient(to right, #4298ae, #2e2bc5);
  display: flex;
  justify-content: center;
  align-items: center;
  flex-direction: column;
}
.row {
  width: 1000px;
  display: flex;
  justify-content: center;
  align-items: center;
  margin: 10px;
  color: white;
  font-size: 20px;
  font-family: "Segoe UI", Tahoma, Geneva, Verdana, sans-serif;
}

.row > div > img {
  width: 100px;
  padding: 0px 15px;
}

#search {
  font-size: 18px;
  padding: 15px;
  border-radius: 25px;
  border: none;
  outline: none;
  box-shadow: 0px 0px 5px gray;
  font-weight: bolder;
}
```

app.js:

```
// Add your API key here
const API_KEY = ``;

const form = document.querySelector("form");
const search = document.querySelector("#search");
const weather = document.querySelector("#weather");

const getWeather = async (city) => {
  weather.innerHTML = "<h1 style='color:yellow'>Loading...</h1>"
  const url = `https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${API_KEY}&units=metric`;
  const response = await fetch(url);
  const data = await response.json();
  return showWeather(data);
};

const showWeather = (data) => {
  if(data.cod == "404"){
    weather.innerHTML = "<h1>City Not Found</h1>";
    return;
  }
  else{
```

```
  weather.innerHTML = `
     <div>
     <img
        src="https://openweathermap.org/img/wn/${data.weather[0].icon}@2x.png"
        alt="Cloud-img"
      />
     </div>
     <div>
      <h2>${data.main.temp} °C</h2>
      <h4>Clear</h4>
     </div>
     `;
  }
};


form.addEventListener("submit", function (event) {
  getWeather(search.value);
  event.preventDefault();
});
```

5) Create a Fully functional Calculator which performs all the Operations Using HTML, CSS, JS. And make sure to only commit the reviewed code to master branch.

→

Index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Calculator</title>
   <link rel="stylesheet" href="style.css">
</head>
```

```
<body>
  <div class="container">
    <div class="calculator">
     <form id="form-control">
      <div class="display">
       <input type="text" name="display" style="color:black">
      </div>
      <div>
       <input type="button" value="AC" class="operator">
       <input type="button" value="DE" class="operator">
       <input type="button" value="." class="operator">
       <input type="button" value="/" class="operator">
      </div>

      <div>
       <input type="button" value="7" class="operator">
       <input type="button" value="8" class="operator">
       <input type="button" value="9" class="operator">
       <input type="button" value="*" class="operator">
      </div>

        <div>
       <input type="button" value="4" class="operator">
       <input type="button" value="5" class="operator">
       <input type="button" value="6" class="operator">
       <input type="button" value="-" class="operator">
      </div>

      <div>
       <input type="button" value="1" class="operator">
       <input type="button" value="2" class="operator">
       <input type="button" value="3" class="operator">
       <input type="button" value="+" class="operator">
      </div>


       <div>
       <input type="button" value="00" class="operator">
       <input type="button" value="0" class="operator">
       <input type="button" value="=" class="equal operator" onclick="display.value = eval(display.value)">
      </div>
     </form>
    </div>
   </div>
   <script src="app.js"></script>
</body>
</html>
```

Style.css:

```
*{
  margin:0;
  padding:0;
  font-family: 'poppins',sans-serif;
  box-sizing:border-box;
 }

 .container{
  width:100%;
  height:100vh;
  background-color:#e3f9ff;
  display:flex;
  justify-content:center;
  align-items:center;
 }
```

```
.calculator{
  background-color:#3a4452;
  padding:20px;
  border-radius:10px;
}

.calculator form input{
  border:0;
  outline:0;
  width:60px;
  height:60px;
  border-radius:10px;
    box-shadow: -8px -8px 15px rgba(255, 255, 255, 0.1),5px 5px 15px rgba(0, 0, 0, 0.2);
      background: transparent;
      font-size: 20px;
      color: #fff;
      margin: 10px;
  cursor:pointer;
}

form .display{
  display:flex;
  justify-content:center;
  margin:20px 0;
  border-radius:8px;
  background-color: #e3f9ff;
}

form .display input{
    text-align: right;
    flex: 1;
    font-size: 45px;
```

```
 /*     box-shadow: none; */
 }


 form input.equal{
    width: 145px;
 }



 form input.operator{
    color: #33ffd8 ;
 }
```

App.js:

```
const formControl = document.getElementById("form-control");

document.addEventListener("DOMContentLoaded", function () {
   const display = document.querySelector('input[name="display"]');
   const buttons = document.querySelectorAll('.operator');

   buttons.forEach(button => {
     button.addEventListener('click', function () {
       const value = this.value;

       if (value === '=') {
         display.value = eval(display.value);
       } else if (value === 'AC') {
         display.value = '';
       } else if (value === 'DE') {
         display.value = display.value.toString().slice(0, -1);
       } else {
         display.value += value;
       }
     });
```

```
    });
  });
```

6) Create a Word Count App which keeps track of Total words, characters, Alphabets, Numbers Using HTML, CSS, JS. And make sure to commit only reviewed code to master branch

→

Index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Words Counter</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <div class="container">
    <h1 class="heading">Words Counter</h1>
      <textarea name="inputBox" id="inputBox" cols="30" rows="10"
      oninput="wordCounter()" placeholder="Enter your text.."></textarea>
      <div class="outputBox">
        <div>
          <label for="words">Total Words</label>
          <input type="button" value="0" id="words">
        </div>
        <div>
          <label for="chars">Total Characters</label>
          <input type="button" value="0" id="chars">
        </div>
        <div>
          <label for="alphabets">Total Alphabets</label>
```

```
                    <input type="button" value="0" id="alphabets">
                </div>
                <div>
                    <label for="numbers">Total Numbers</label>
                    <input type="button" value="0" id="numbers">
                </div>
            </div>
        </div>
    </div>

    <script src="app.js"></script>
</body>
</html>
```

Style.css:

```
*{
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}
.container{
    display: flex;
    flex-direction: column;
    align-items: center;
    padding-top: 20px;
    width: 100vw;
    height: 100vh;
    background-color: rgb(95, 94, 88);
    color:white;
}
.container textarea{
    width: 70%;
    height: 40%;
    outline: none;
    border-radius: 10px;
    font-size: 20px;
    font-family: Cambria;
    max-width: 780px;
    resize:vertical;
    margin-top: 20px;
    padding: 4px;
    padding-top: 10px;
    transition: box-shadow .15s ease-in-out;
}
.container textarea:focus{
    box-shadow: 0px 0px 2px 4px rgb(150, 148, 138);
}
.outputBox{
    display: flex;
    flex-direction: column;
    margin-top: 10px;
}
.outputBox div{
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding: 4px;
}
.outputBox input{
    width:90px;
    height: 40px;
    margin-left: 10px;
    box-shadow: 1px 1px 1px 1px rgb(91, 86, 86);
    border:none;
    border-radius: 4px;
```

```
      background-color:rgb(220, 222, 225);
      color: rgb(144, 50, 6);
      font-weight: bold;
      font-size: larger;
    }
    .outputBox label{
      font-weight: bold;
      font-size: larger;
      /* color:rgb(33, 26, 26); */
    }
```

App.js:

```
function getWordCount(str) {
  let splited=str.trim().split(/\s+/);
  console.log(splited);
  if(splited==''){
    return 0;
  }
  return splited.length;
}
function counter(str){
  let alphaCount=0;
  let numberCount=0;
  let charCount=0;
  for(let i=0;i<str.length;i++){
    if(str[i]!=' ' && str[i]!='\n'){
      charCount++;
    }
    if((str[i]>='a'&&str[i]<='z') || (str[i]>='A' && str[i]<='Z')){
      alphaCount++;
    }
    else if(str[i]>='0' && str[i]<='9'){
      numberCount++;
    }
  }
  document.getElementById("chars").value = charCount;
  document.getElementById("numbers").value = numberCount;
  document.getElementById("alphabets").value = alphaCount;
}
function wordCounter()
{
  let inputTxt = document.getElementById("inputBox");
  let words = getWordCount(inputTxt.value);
  document.getElementById("words").value = words;
  counter(inputTxt.value);
}
```

7) Create a Single React Application Using react -router-dom. And make sure to commit reviewed code to master branch.

→

App.jsx:

```
import { Route, Routes } from "react-router-dom";
import "./App.css";
import { Navbar } from "./components/Navbar";
import { About, Contact, Home, Services } from "./components/pages";

function App() {
 return (
  <div className="App">
   <Navbar />
   <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/about" element={<About />} />
    <Route path="/services" element={<Services />} />
    <Route path="/contact" element={<Contact />} />
```

```
      </Routes>
    </div>
  );
}

export default App;
```

main.js

```
import { BrowserRouter } from "react-router-dom";
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App.jsx";
import "./index.css";

ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>
);
```

Navbar.js:

```
import React, { useState } from "react";

import "./Navbar.css";
import { Link, NavLink } from "react-router-dom";

export const Navbar = () => {
  const [menuOpen, setMenuOpen] = useState(false);
  return (
    <nav>
      <Link to="/" className="title">
        Website
      </Link>
      <div className="menu" onClick={() => setMenuOpen(!menuOpen)}>
        <span></span>
        <span></span>
        <span></span>
      </div>
      <ul className={menuOpen ? "open" : ""}>
        <li>
          <NavLink to="/about">About</NavLink>
        </li>
        <li>
          <NavLink to="/services">Services</NavLink>
        </li>
        <li>
          <NavLink to="/contact">Contact</NavLink>
        </li>
      </ul>
    </nav>
  );
};
```

Navbar.css:

```
.active {
  background-color: #1d4ed8;
}

nav {
  display: flex;
  justify-content: space-between;
  align-items: center;
  background-color: #0f172a;
  color: white;
  position: sticky;
  top: 0;
}

nav .title {
  font-size: 1.5rem;
  margin: 1rem;
  font-weight: bold;
  text-decoration: none;
  color: white;
}

nav ul {
  display: flex;
}

nav ul li {
  list-style: none;
}

nav ul li a {
  display: block;
  text-decoration: none;
  color: white;
  padding: 0.5rem;
  margin: 0 0.5rem;
  border-radius: 0.5rem;
}
```

```
nav ul li a:not(.active):hover {
  background-color: #172554;
}

nav .menu {
  display: none;
  position: absolute;
  top: 0.75rem;
  right: 0.5rem;
  flex-direction: column;
  justify-content: space-between;
  width: 2.25rem;
  height: 2rem;
}

nav .menu span {
  height: 0.4rem;
  width: 100%;
  background-color: #fff;
  border-radius: 0.2rem;
}

@media (max-width: 480px) {
  nav .menu {
    display: flex;
  }

  nav {
    flex-direction: column;
    align-items: flex-start;
  }
  nav ul {
    display: none;
    flex-direction: column;
    width: 100%;
    margin-bottom: 0.25rem;
  }

  nav ul.open {
    display: flex;
  }

  nav ul li {
    width: 100%;
    text-align: center;
  }

  nav ul li a {
    margin: 0.2rem 0.5rem;
  }
}
```

Home.js:

```
import React from "react";

export const Home = () => {
  return <h1>Home</h1>;
};
```

About.js:

```
import React from "react";
```

```
export const About = () => {
  return <h1>About</h1>;
};
```

Service.js:

```
import React from "react";


export const Services = () => {
  return (
    <>
      <h1>Services</h1>
    </>
  );
};
```

8) Create a Spring boot Application and perform all the Annotations (Autowiring, Qualifier and Bean Scope). And make sure to commit reviewed code to master branch.

→

Autowiring in Spring:

- Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection.
- Autowiring can't be used to inject primitive and string values. It works with reference only.

Autowiring Modes

1. No: It is the default autowiring mode. It means no autowiring bydefault.

2. byName: The byName mode injects the object dependency according to name of the bean. In such case, property name and bean name must be same. It internally calls setter method.

3. byType: The byType mode injects the object dependency according to type. So property name and bean name can be different. It internally calls setter method.

Com.example.demo:

```
package com.example.demo;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.support.ClassPathXmlApplicationContext;

@SpringBootApplication public class NewApplication { public static void main(String[] args) {
System.out.println("hello");

ClassPathXmlApplicationContext context=new ClassPathXmlApplicationContext("stud.xml"); student s=
(student) context.getBean("stud");
```

```
 System.out.println(s); } }
```

(@Autowired)

Student.java:

```
package com.example.demo;
import org.springframework.beans.factory.annotation.Autowired;
public class student {
private String rollno;
private String name;
@Autowired
private Address address;
public student() { }
public student(String rollno, String name, Address address) {
super();
this.rollno = rollno;
this.name = name;
this.address = address;
 System.out.println("this is get and set level");
}
public String getRollno() { return rollno; }
public void setRollno(String rollno) { this.rollno = rollno; }
public String getName() { return name; }
public void setName(String name) { this.name = name; }
public Address getAddress() { return address; }
public void setAddress(Address address) { this.address = address; }
@Override public String toString() {
 return "student{" + "rollno='" + rollno + '\'' + ", name='" + name + '\'' + ", address=" + address + '}'; }
 }
```

Address.java:

```
public class Address {
private String place;
private String city;
public Address() { }
public Address(String place, String city) {
this.place = place;
this.city = city;
 }
public String getPlace() { return place; }
public void setPlace(String place) { this.place = place; }
public String getCity() { return city; }
public void setCity(String city) { this.city = city; }
@Override public String toString() {
 return "Address{" + "place='" + place + '\'' + ", city='" + city + '\'' + '}'; }
 }
```

(@Qualifier)

Student.java :

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
public class student { private String rollno;
private String name;
@Autowired
@Qualifier("temp")
private Address address;
public student() { }
public student(String rollno, String name, Address address) {
 super();
 this.rollno = rollno;
this.name = name;
this.address = address;
System.out.println("this is get and set level"); }
public String getRollno() { return rollno; }
public void setRollno(String rollno) { this.rollno = rollno; }
public String getName() { return name; }
public void setName(String name) { this.name = name; }
public Address getAddress() { return address }
public void setAddress(Address address) { this.address = address; }
@Override

public String toString() {

return "student{" + "rollno='" + rollno + '\'' + ", name='" + name + '\'' + ", address=" + address + '}'; }

 }
```

9) Perform CURD Operations on MongoDB.

basic methods of interacting with a MongoDB server are called CRUD operations. CRUDstands for Create, Read, Update, and Delete. CRUD operations describe the conventions of a user-interface that let users view, search, and modify parts of the database.

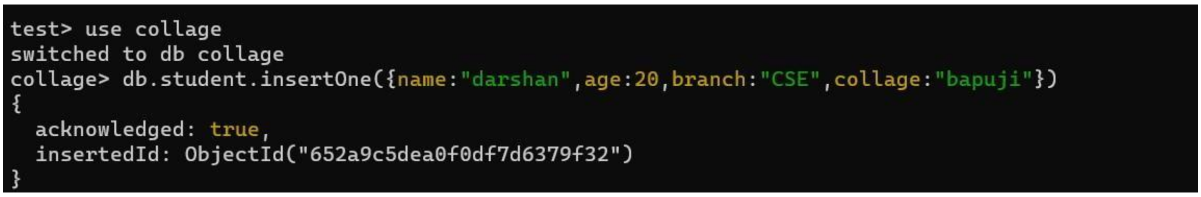
The individual CRUD operations:

- The Create operation is used to insert new documents in the MongoDB database.
- The Read operation is used to query a document in the database.
- The Update operation is used to modify existing documents in the database.
- The Delete operation is used to remove documents in the database.

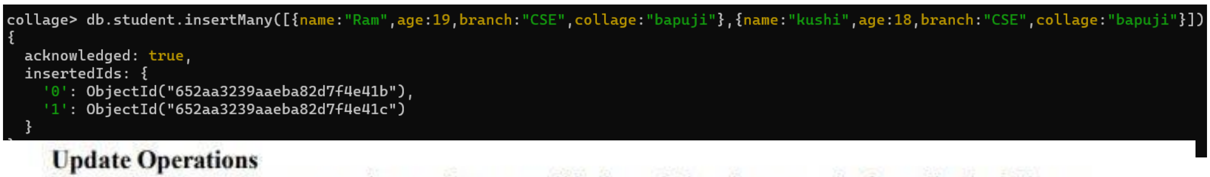
Create Operations

The create or insert operations are used to insert or add new documents in the collection. If a collection does not exist, then it will create a new collection in the database. We can perform,create operations using the following methods provided by the MongoDB:

Method	Description
db.collection.insertOne()	It is used to insert a single document in the collection.
db.collection.insertMany()	It is used to insert multiple documents in the collection.



```
test> use collage
switched to db collage
collage> db.student.insertOne({name:"darshan",age:20,branch:"CSE",collage:"bapuji"})
{
  acknowledged: true,
  insertedId: ObjectId("652a9c5dea0f0df7d6379f32")
}
```

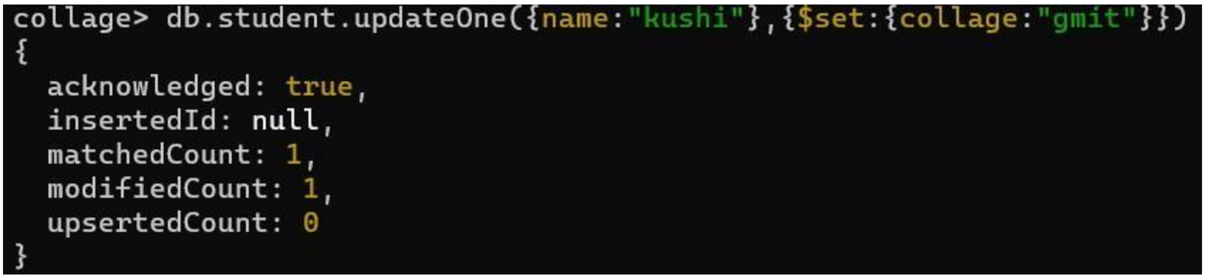


```
collage> db.student.insertMany([{name:"Ram",age:19,branch:"CSE",collage:"bapuji"},{name:"kushi",age:18,branch:"CSE",collage:"bapuji"}])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("652aa3239aaeba82d7f4e41b"),
    '1': ObjectId("652aa3239aaeba82d7f4e41c")
  }
```

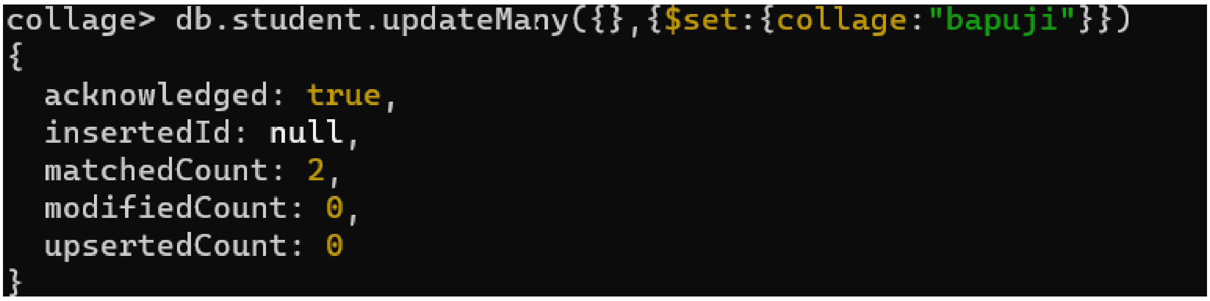
Update Operations

The update operations are used to update or modify the existing document in the collection.We can perform update operations using the following methods provided by the MongoDB:

Method	Description
db.collection.updateOne()	It is used to update a single document in the collection that satisfy the given criteria.
db.collection.updateMany()	It is used to update multiple documents in the collection that satisfy the given criteria.



```
collage> db.student.updateOne({name:"kushi"},{$set:{collage:"gmit"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

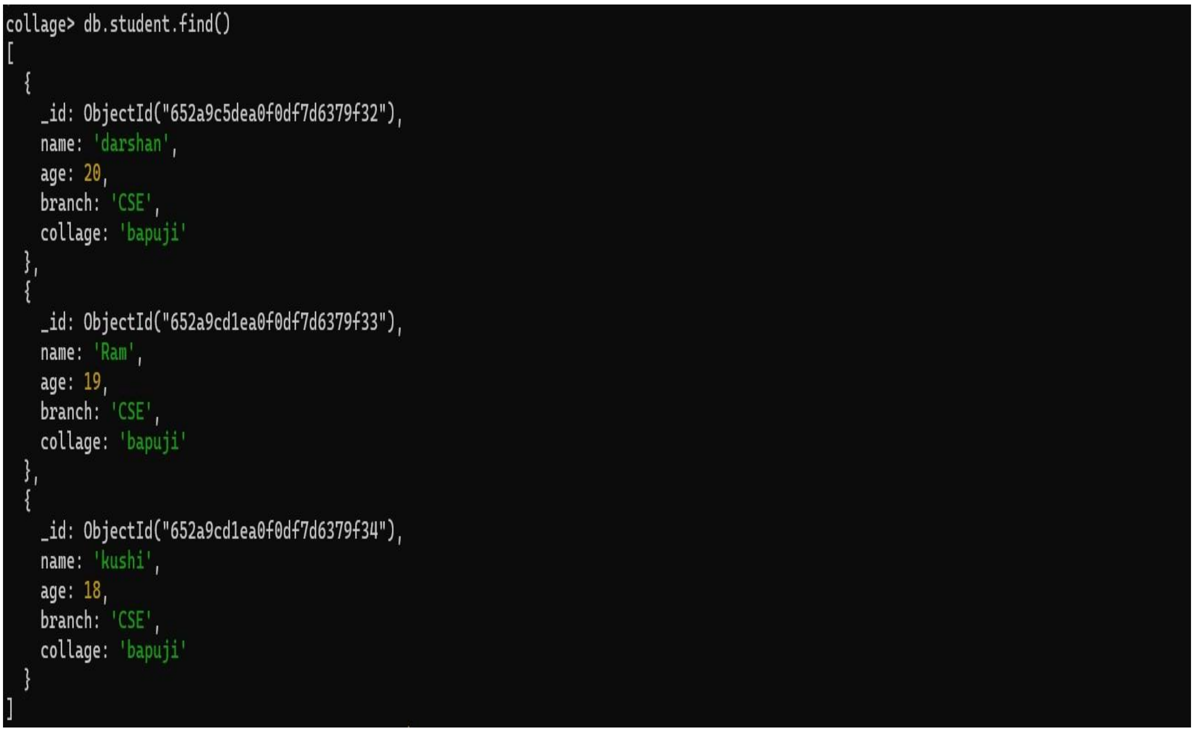


```
collage> db.student.updateMany({},{$set:{collage:"bapuji"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 2,
  modifiedCount: 0,
  upsertedCount: 0
}
```

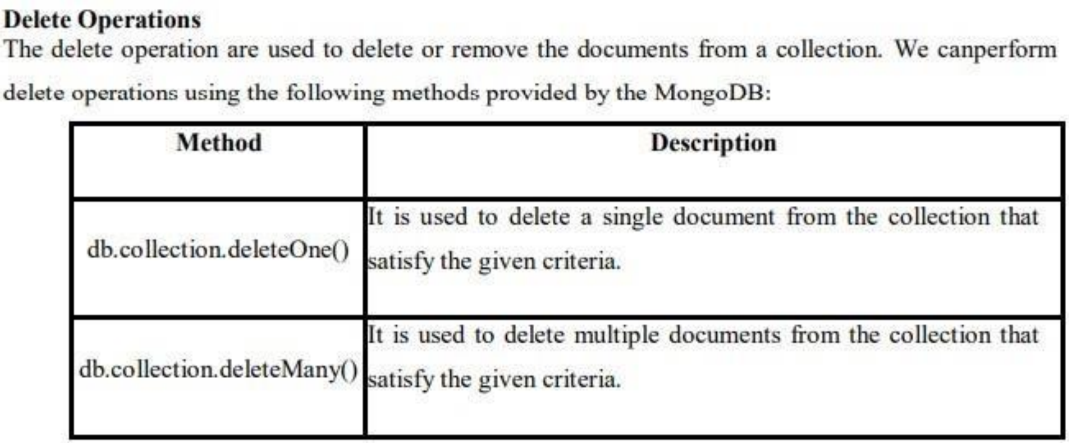
Read Operations

The Read operations are used to retrieve documents from the collection, or in other words, read operations are used to query a collection for a document. We can perform read operationusing the following method provided by the MongoDB:

Method	Description
db.collection.find()	It is used to retrieve documents from the collection.



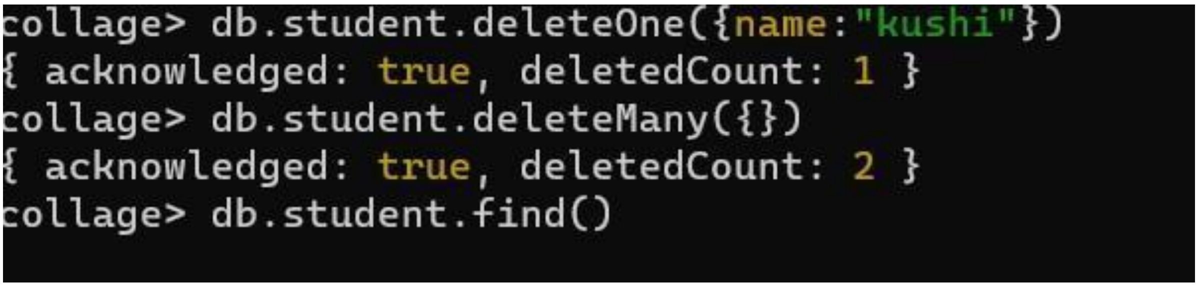
```
collage> db.student.find()
[
  {
    _id: ObjectId("652a9c5dea0f0df7d6379f32"),
    name: 'darshan',
    age: 20,
    branch: 'CSE',
    collage: 'bapuji'
  },
  {
    _id: ObjectId("652a9cd1ea0f0df7d6379f33"),
    name: 'Ram',
    age: 19,
    branch: 'CSE',
    collage: 'bapuji'
  },
  {
    _id: ObjectId("652a9cd1ea0f0df7d6379f34"),
    name: 'kushi',
    age: 18,
    branch: 'CSE',
    collage: 'bapuji'
  }
]
```



Delete Operations

The delete operation are used to delete or remove the documents from a collection. We canperform delete operations using the following methods provided by the MongoDB:

Method	Description
db.collection.deleteOne()	It is used to delete a single document from the collection that satisfy the given criteria.
db.collection.deleteMany()	It is used to delete multiple documents from the collection that satisfy the given criteria.



```
collage> db.student.deleteOne({name:"kushi"})
{ acknowledged: true, deletedCount: 1 }
collage> db.student.deleteMany({})
{ acknowledged: true, deletedCount: 2 }
collage> db.student.find()
```

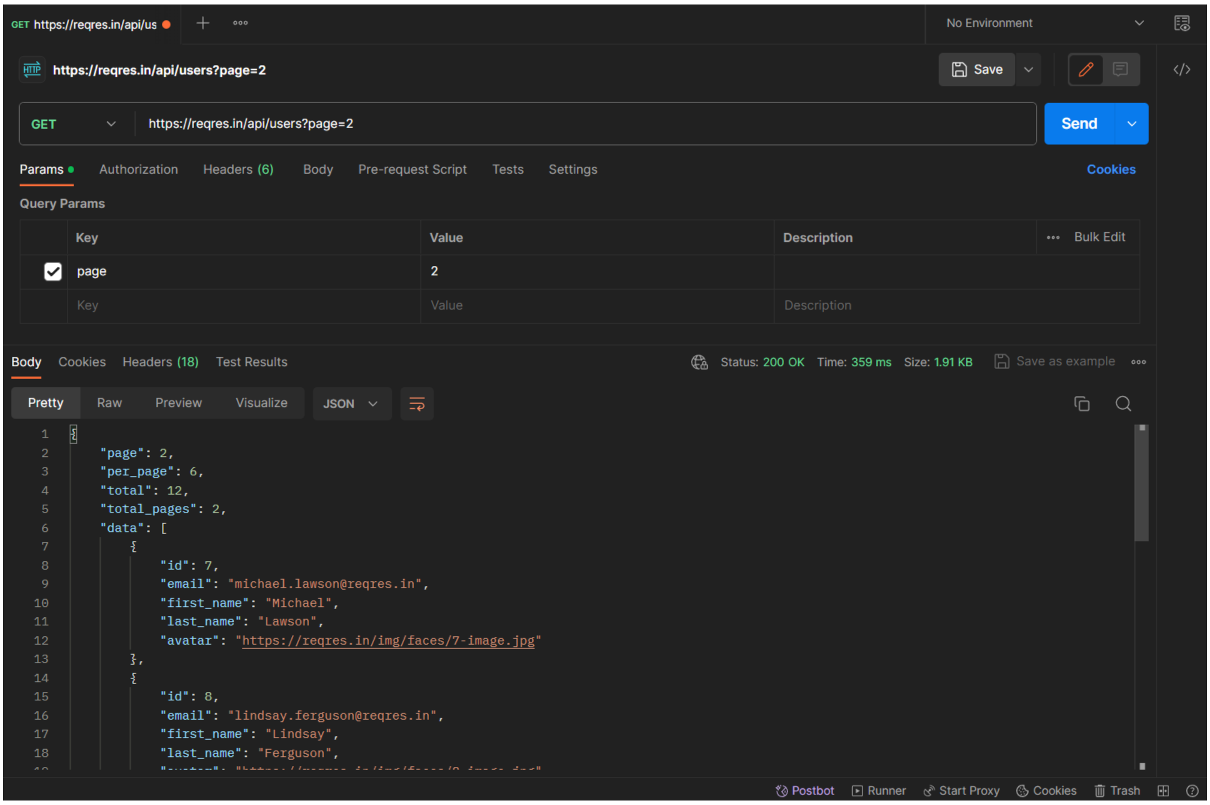
10) Perform HTTPS Methods [GET, POST, PUT, DELETE] to Understand the JSON structure for API request and response data using Software Testing tools like Postman.

→

HTTPS Methods:

GET Method:

A GET request in HTTP is like asking for a specific webpage or information from a server. It's used to retrieve data, works through URLs, and is visible in the browser's address bar. GET requests are simple, cacheable, and considered safe for retrieving data, but not ideal for sending sensitive information as the data is visible in the URL.



https://reqres.in/api/users?page=2

GET https://reqres.in/api/users?page=2

Query Params

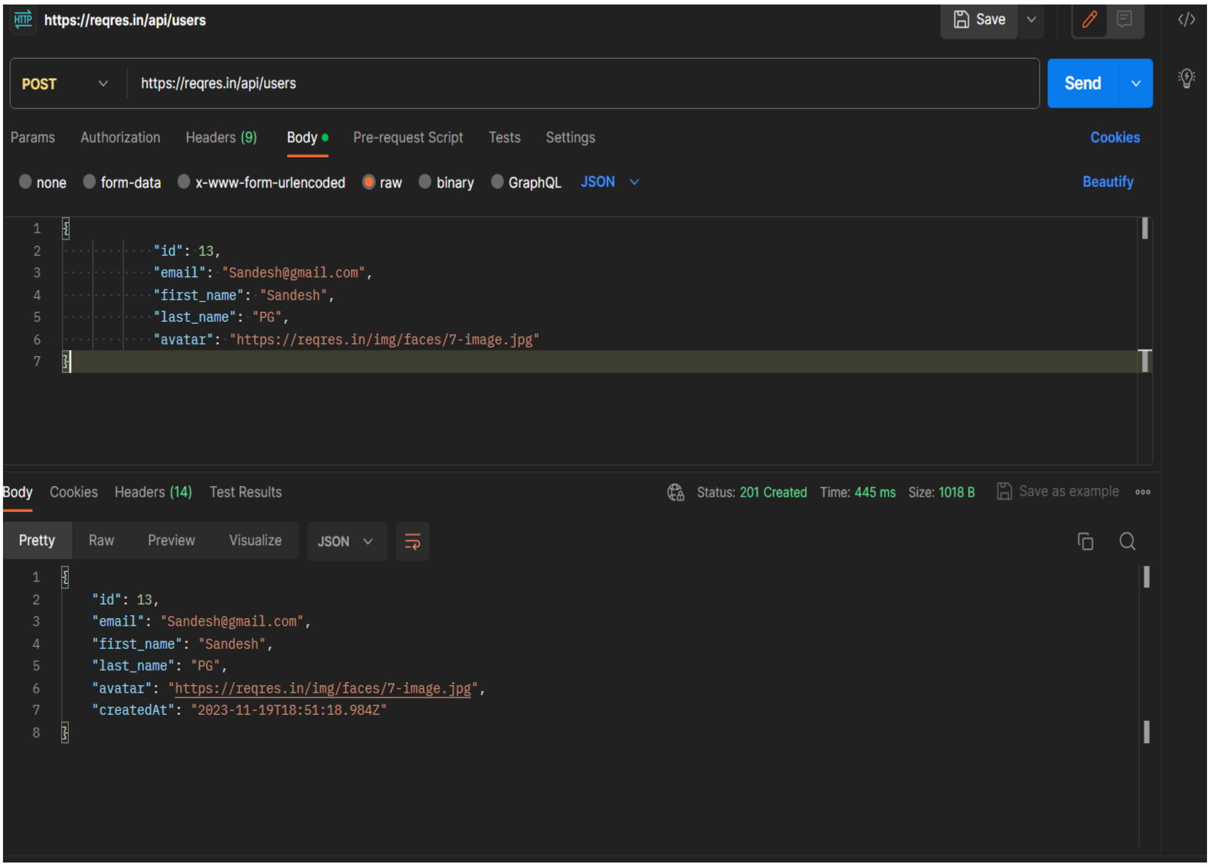
	Key	Value	Description
☑	page	2	
	Key	Value	Description

Status: 200 OK Time: 359 ms Size: 1.91 KB

```
{
    "page": 2,
    "per_page": 6,
    "total": 12,
    "total_pages": 2,
    "data": [
        {
            "id": 7,
            "email": "michael.lawson@reqres.in",
            "first_name": "Michael",
            "last_name": "Lawson",
            "avatar": "https://reqres.in/img/faces/7-image.jpg"
        },
        {
            "id": 8,
            "email": "lindsay.ferguson@reqres.in",
            "first_name": "Lindsay",
            "last_name": "Ferguson",
            "avatar": "https://reqres.in/img/faces/8-image.jpg"
```

POST Method:

The POST method in HTTP is used to send data to a server to create or update a resource. Unlike GET, it doesn't show data in the URL; instead, it sends it in the request body. POST requests are often used for forms, login credentials, or any data that should not be visible in the URL. They're not cached and are considered non-idempotent, meaning repeating the same request may result in different outcomes.



POST https://reqres.in/api/users

Body: ○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON

```
{
            "id": 13,
            "email": "Sandesh@gmail.com",
            "first_name": "Sandesh",
            "last_name": "PG",
            "avatar": "https://reqres.in/img/faces/7-image.jpg"
}
```

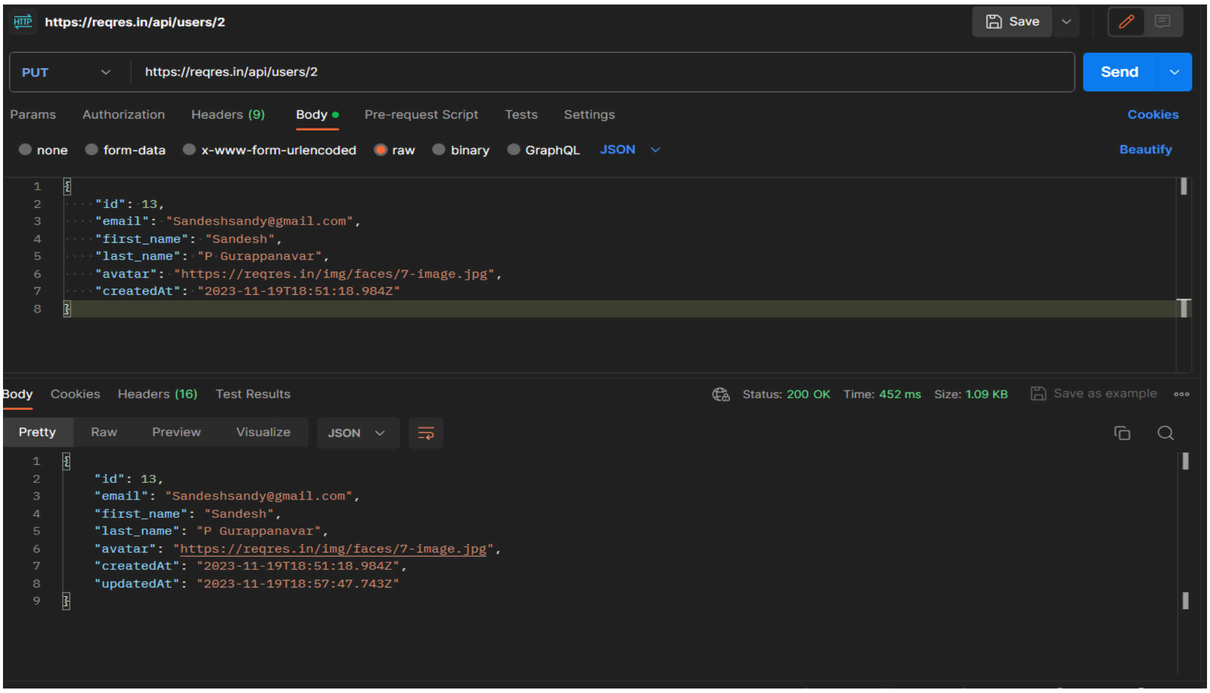
Body — Status: 201 Created Time: 445 ms Size: 1018 B

JSON

```
{
    "id": 13,
    "email": "Sandesh@gmail.com",
    "first_name": "Sandesh",
    "last_name": "PG",
    "avatar": "https://reqres.in/img/faces/7-image.jpg",
    "createdAt": "2023-11-19T18:51:18.984Z"
}
```

PUT Method:

PUT in HTTP is a method to update or create a resource on the server. It sends data to a specific URL to replace or create the resource indicated by that URL. It's used for updating existing resources or creating new ones and is considered idempotent, meaning repeating the same request won't change the outcome if the resource remains the same.



https://reqres.in/api/users/2

PUT https://reqres.in/api/users/2

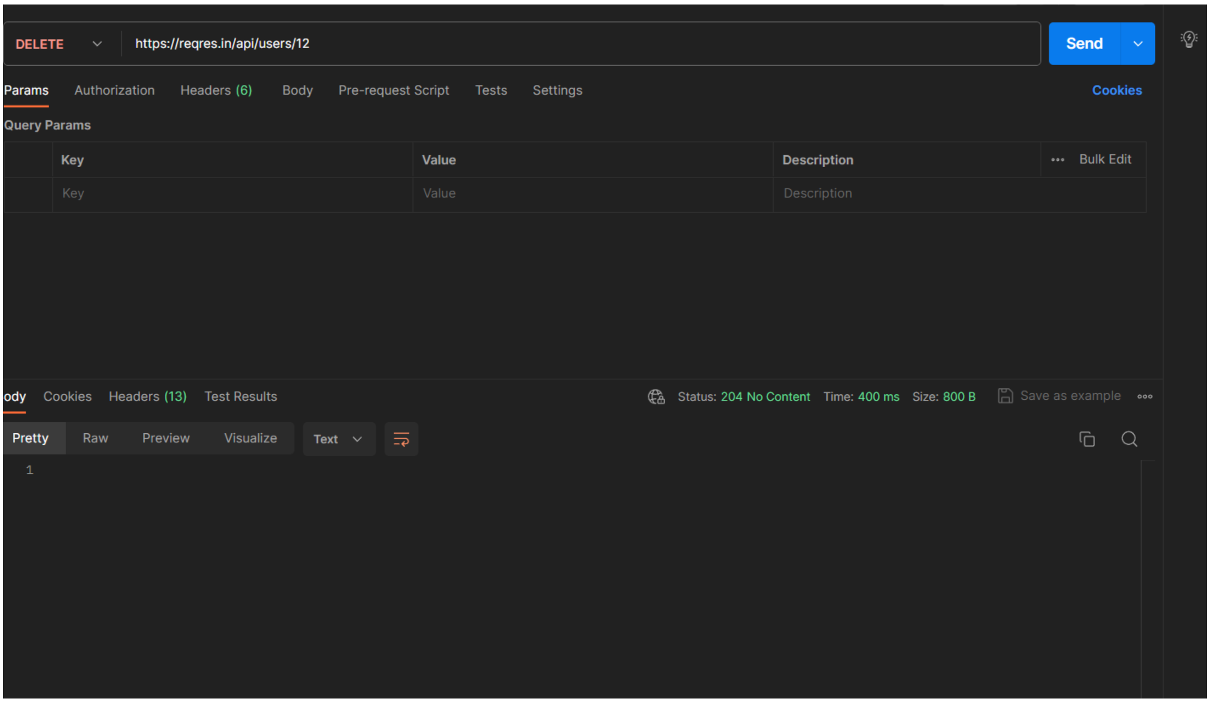
```
{
    "id": 13,
    "email": "Sandeshsandy@gmail.com",
    "first_name": "Sandesh",
    "last_name": "P Gurappanavar",
    "avatar": "https://reqres.in/img/faces/7-image.jpg",
    "createdAt": "2023-11-19T18:51:18.984Z"
}
```

Status: 200 OK Time: 452 ms Size: 1.09 KB

```
{
    "id": 13,
    "email": "Sandeshsandy@gmail.com",
    "first_name": "Sandesh",
    "last_name": "P Gurappanavar",
    "avatar": "https://reqres.in/img/faces/7-image.jpg",
    "createdAt": "2023-11-19T18:51:18.984Z",
    "updatedAt": "2023-11-19T18:57:47.743Z"
}
```

DELETE Method:

DELETE in HTTP is a method used to remove a resource from a server. It's like asking the server to get rid of a specific item or data stored at a particular URL. DELETE requests are used to delete resources and are considered idempotent, meaning repeated requests won't change the outcome once the resource is deleted.



DELETE https://reqres.in/api/users/12

Query Params

Key	Value	Description
Key	Value	Description

Status: 204 No Content Time: 400 ms Size: 800 B