BRAC UNIVERSITY

Inspiring Excellence

CSE471: System Analysis and Design

Project Report

Project Title:

TutorConnect: Personalized online tutoring and course delivery system

Group No: 03, CSE471 Lab Section: 09, Summer 2025	
ID	**Name**
22101451	S. M. Saidur Rahman
22101651	Shayonto Rayhan
22101439	MD.Niamatullah
22101438	Md. Ann-Am Akbar Saad

Submission Date: 10/09/2025

Table of Contents

1. System Request

Business need:

TutorConnect was developed to provide a personalized and scalable online tutoring and course delivery system where students, tutors, and administrators can interact through a single platform. Traditional online tutoring platforms often fail to offer strong personalization, multilingual academic support, and automated evaluation mechanisms. These limitations make the learning experience less efficient and engaging for students, while also creating additional workload for tutors and administrators. TutorConnect addresses these challenges by integrating role-based access control, interactive learning modules, in-app communication channels and an intelligent AI assistant. The platform allows students to access high-quality learning resources, book live tutoring sessions, and receive bilingual academic support, while tutors benefit from automated feedback analytics, dynamic course management, and AI-assisted grading. Administrators, on the other hand, can efficiently manage the system, enforce security, and maintain smooth operation of the platform.

Business requirements:

The system is designed to fulfill several key requirements. It provides secure authentication and role-based access for three types of users: students, tutors, and administrators. Each user has a dedicated dashboard tailored to their role. Students can customize their profiles by adding details such as subjects of interest, preferred language of learning, and available time slots. Tutors are able to create and manage courses by uploading videos, documents, and quizzes, while students can enroll, track progress, and earn certificates upon completion. A major requirement of the system is the integration of in-app communication. This includes real-time messaging, dedicated course chat channels, and an AI assistant that supports both Bangla and English to address academic and platform-related queries. Another significant requirement is intelligent evaluation: tutors can create assignments with multiple question types, and the system integrates plagiarism detection as well as AI-generated grading suggestions. Also, teachers can make question papers with the help of AI assistants. Finally, administrators must be able to manage user roles, control access, and monitor platform activities.

Business value:

TutorConnect delivers value to all stakeholders. For students, the system creates a personalized learning experience with interactive content, progress tracking, and instant bilingual assistance. Tutors benefit from automated grading suggestions, plagiarism detection, making AI based question papers with their resources and structured access to student feedback analytics. Administrators gain a secure and centralized control system that allows them to monitor users, manage roles, and ensure compliance with platform policies. At the organizational level, the system reduces inefficiencies, enhances scalability, and increases the overall quality of online education by offering a modern, AI-driven learning environment.

Special issues or constraints:

There are several constraints associated with the development and deployment of TutorConnect. The system relies heavily on MongoDB Atlas for cloud-based database management, which requires careful attention to data privacy and security. The AI chatbot requires continuous dataset enrichment and must be frequently retrained to remain accurate and reliable in both Bangla and English. Real-time communication features such as messaging and live tutoring sessions depend on WebSocket performance, which requires stable internet connectivity and proper server scaling. In addition, plagiarism detection mechanisms may rely on third-party APIs, which could introduce licensing and cost-related challenges.

2. Functional Requirements

MODULE 1: User & Profile Management

Focus: Role-based access, onboarding, personalization, and feedback

1. [Shayonto Rayhan] Secure sign-up and login for students, tutors, and admins.
2. [Md. Ann-Am Akbar Saad] User profile customization including bio, subjects, languages, and availability.
3. [MD.Niamatullah] Students rate tutors and courses; tutors access feedback analytics.
4. [S.M. Saidur Rahman] Admins manage user roles and control access.

MODULE 2: Learning & Tutoring System

Focus: Course management, learning tools, and tutoring workflow

1. [Md. Ann-Am Akbar Saad] Students enroll in courses, track progress, and earn certificates.
2. [MD.Niamatullah] Tutors create courses and upload videos, documents, and quizzes.
3. [Shayonto Rayhan] Interactive tools include assignments, forums, and quizzes for engagement.
4. [S.M. Saidur Rahman] Students book live tutoring sessions; tutors approve or reject requests.

MODULE 3: In-App Communication & AI Assistant

Focus: Messaging, intelligent assistance, and platform guidance

1. [MD.Niamatullah] Secure real-time messaging between students and tutors.
2. [Md. Ann-Am Akbar Saad] Dedicated chat channels for courses and tutoring sessions.
3. [Shayonto Rayhan] Bilingual AI chatbot assists with academic questions in Bangla and English for registered and unregistered users.
4. [S.M. Saidur Rahman] AI-powered FAQ answers platform-related queries using internal data.

MODULE 4: Smart Assignment & Evaluation Module for Tutors

Focus: RAG-powered assignments, plagiarism detection, and intelligent grading support.

1. [S.M. Saidur Rahman]Tutors create assignments and questions (MCQ, short, essay) from the RAG course database.

2. [Shayonto Rayhan] AI & plagiarism detector scans student submissions and generates highlighted reports.

3. [Md. Ann-Am Akbar Saad] RAG-powered chatbot auto-evaluates short answers and provides grading suggestions.

4. [MD.Niamatullah] RAG integrated with MongoDB Atlas lets tutors update course materials dynamically without redeploying the AI module.

3. Technology (Framework, Languages)

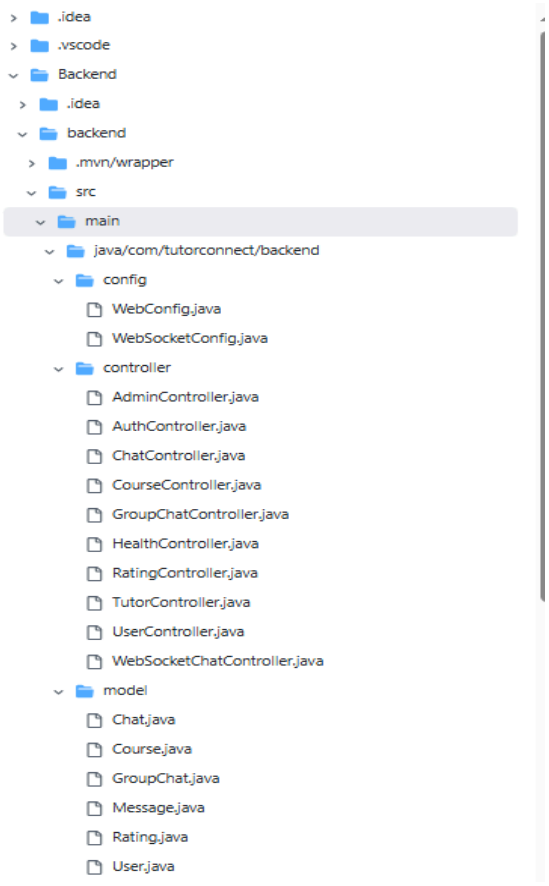
1. **Frontend**:
    a. HTML, CSS, Js

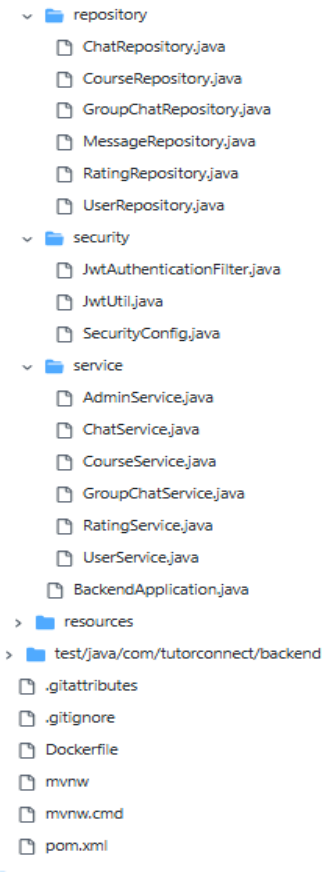
2. **Backend**:
    a. SpringBoot, Spring Security, JWT, WebSocket

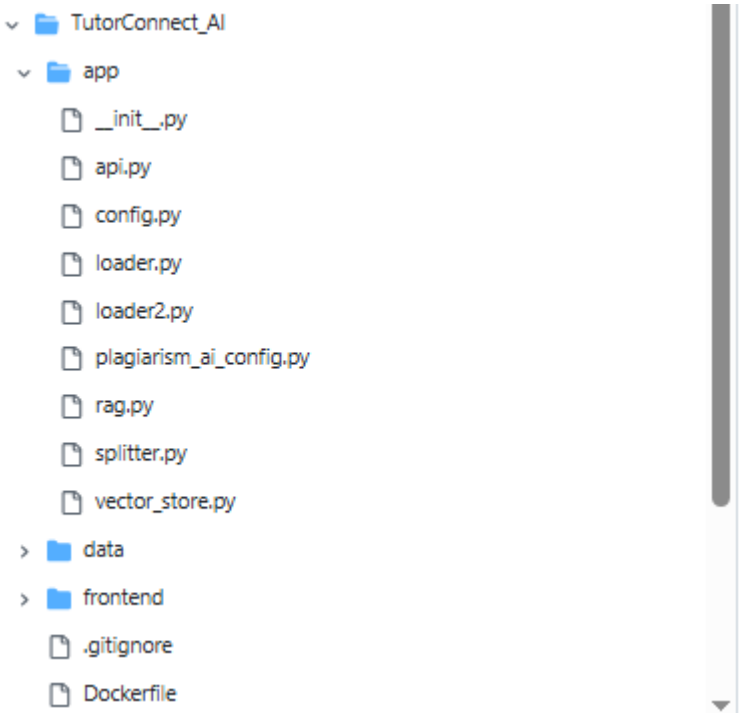
3. **Database**:
    a. MongoDB ATLAS

4. Backend Development

Backed Folder Structure







AuthController

```java
package com.tutorconnect.backend.controller;

import com.tutorconnect.backend.model.User;
import com.tutorconnect.backend.service.UserService;
import com.tutorconnect.backend.security.*;
import java.util.Map;
import java.util.List;
import java.util.HashMap;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/auth")
```

TutorConnect-Personalized-online-tutoring-and-course-delivery-system / Backend / backend / src / main / java / com / tutorconnect / ba

Code | Blame | 78 lines (69 loc) · 3.01 KB

```java
    @Autowired
    private JwtUtil jwtUtil;

    @PostMapping("/signup")
    public ResponseEntity<?> register(@RequestBody User user) {
        if (userService.existsByEmail(user.getEmail())) {
            return ResponseEntity.badRequest().body("{\"success\":false,\"message\":\"Email already exists\"}");
        }
        userService.registerUser(user);
        return ResponseEntity.ok("{\"success\":true}");
    }

    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestBody Map<String, String> loginData) {
        String email = loginData.get("email");
        String password = loginData.get("password");
        Optional<User> userOpt = userService.findByEmail(email);
        if (userOpt.isEmpty()) {
            return ResponseEntity.status(401).body(Map.of("message", "Invalid credentials"));
        }
        User user = userOpt.get();
        if (!userService.checkPassword(password, user.getPasswordHash())) {
            return ResponseEntity.status(401).body(Map.of("message", "Invalid credentials"));
        }
        String token = jwtUtil.generateToken(user);
        return ResponseEntity.ok(Map.of(
            "token", token,
            "role", user.getRole()
        ));
    }

    // Search users for chat functionality (authenticated users only)
    @GetMapping("/users/search")
    public ResponseEntity<?> searchUsers(@RequestParam String query, Authentication authentication) {
        if (authentication == null) {
            return ResponseEntity.status(401).body(Map.of("message", "Unauthorized"));
        }

        try {
            List<User> users = userService.searchUsersByUsernameOrEmail(query);

            // Return safe user info (no password hash or sensitive data)
            List<Map<String, Object>> safeUsers = users.stream().map(user -> {
                Map<String, Object> userMap = new HashMap<>();
                userMap.put("id", user.getId());
                userMap.put("username", user.getUsername() != null ? user.getUsername() : user.getEmail());
                userMap.put("email", user.getEmail());
                userMap.put("role", user.getRole());
                return userMap;
            }).toList();

            return ResponseEntity.ok(safeUsers);
        } catch (Exception e) {
            return ResponseEntity.status(500).body(Map.of("message", "Error searching users"));
        }
    }
}
```

Admin Controller

```
1
2   package com.tutorconnect.backend.controller;
3
4   import com.tutorconnect.backend.model.User;
5   import com.tutorconnect.backend.service.AdminService;
6   import org.springframework.beans.factory.annotation.Autowired;
7   import org.springframework.http.ResponseEntity;
8   import org.springframework.web.bind.annotation.*;
9   import org.springframework.security.access.prepost.PreAuthorize;
10
11  @RestController
12  @RequestMapping("/api/admin")
13  @CrossOrigin(origins = "*", allowedHeaders = "*")
14  public class AdminController {
15      @PreAuthorize("hasRole('ADMIN')")
16      @PostMapping("/create-course")
17      public ResponseEntity<?> createCourse(@RequestBody com.tutorconnect.backend.model.Course course) {
18          boolean created = adminService.createCourse(course);
19          return ResponseEntity.ok(java.util.Map.of("success", created));
20      }
21
22      @PreAuthorize("hasRole('ADMIN')")
23      @PutMapping("/edit-course/{id}")
24      public ResponseEntity<?> editCourse(@PathVariable String id, @RequestBody com.tutorconnect.backend.model.Course updatedCourse) {
25          boolean updated = adminService.editCourse(id, updatedCourse);
26          if (updated) {
27              return ResponseEntity.ok(java.util.Map.of("success", true));
28          } else {
29              return ResponseEntity.badRequest().body(java.util.Map.of("success", false, "message", "Course not found or update failed"));
30          }
31      }
32
33      @PreAuthorize("hasRole('ADMIN')")
34      @GetMapping("/search-course")
35      public ResponseEntity<?> searchCourse(@RequestParam String query) {
36          var courses = adminService.searchCourses(query);
37          var safeCourses = courses.stream().map(c -> {
38              var map = new java.util.HashMap<String, Object>();
39              map.put("id", c.getId());
40              map.put("title", c.getTitle());
41              map.put("description", c.getDescription());
42              map.put("tutorId", c.getTutorId());
43              map.put("subjects", c.getSubjects());
44              map.put("language", c.getLanguage());
45              map.put("price", c.getPrice());
46              map.put("duration", c.getDuration());
47              map.put("studentsEnrolled", c.getStudentsEnrolled());
```

itorConnect-Personalized-online-tutoring-and-course-delivery-system / Backend / backend / src / main / java / com / tutorconnect / backend / controller / Adm

ode | Blame | 132 lines (123 loc) · 5.31 KB

```
14  public class AdminController {
35      public ResponseEntity<?> searchCourse(@RequestParam String query) {
54
55      @PreAuthorize("hasRole('ADMIN')")
56      @GetMapping("/courses")
57      public ResponseEntity<?> getAllCourses() {
58          var courses = adminService.getAllCourses();
59          var safeCourses = courses.stream().map(c -> {
60              var map = new java.util.HashMap<String, Object>();
61              map.put("id", c.getId());
62              map.put("title", c.getTitle());
63              map.put("description", c.getDescription());
64              map.put("tutorId", c.getTutorId());
65              map.put("subjects", c.getSubjects());
66              map.put("language", c.getLanguage());
67              map.put("price", c.getPrice());
68              map.put("duration", c.getDuration());
69              map.put("studentsEnrolled", c.getStudentsEnrolled());
70              map.put("createdAt", c.getCreatedAt());
71              map.put("extra", c.getExtra());
72              return map;
73          }).toList();
74          return ResponseEntity.ok(safeCourses);
75      }
```

```
14    public class AdminController {
77        @GetMapping("/search-tutor")
78        public ResponseEntity<?> searchTutor(@RequestParam String query) {
79            var tutors = adminService.searchTutors(query);
80            var safeTutors = tutors.stream().map(t -> java.util.Map.of(
81                "id", t.getId(),
82                "username", t.getUsername(),
83                "email", t.getEmail(),
84                "role", t.getRole(),
85                "bio", t.getBio(),
86                "subjects", t.getSubjects(),
87                "languages", t.getLanguages(),
88                "availability", t.getAvailability()
89            )).toList();
90            return ResponseEntity.ok(safeTutors);
91        }
92        @Autowired
93        private AdminService adminService;
94
95        @PreAuthorize("hasRole('ADMIN')")
96        @GetMapping("/tutors")
97        public ResponseEntity<?> getAllTutors() {
98            var tutors = adminService.getAllTutors();
99            var safeTutors = tutors.stream().map(t -> java.util.Map.of(
100               "id", t.getId(),
101               "username", t.getUsername(),
102               "email", t.getEmail(),
103               "role", t.getRole(),
104               "bio", t.getBio(),
105               "subjects", t.getSubjects(),
106               "languages", t.getLanguages(),
107               "availability", t.getAvailability()
108           )).toList();
109           return ResponseEntity.ok(safeTutors);
110       }
111
112       @PreAuthorize("hasRole('ADMIN')")
113       @PostMapping("/create-user")
114       public ResponseEntity<?> createUser(@RequestBody User user) {
115           boolean created = adminService.createUser(user);
116           if (!created) {
117               return ResponseEntity.badRequest().body("{\"success\":false,\"message\":\"Email already exists\"}");
118           }
119           return ResponseEntity.ok("{\"success\":true}");
120       }
121
122       @PreAuthorize("hasRole('ADMIN')")
123       @PutMapping("/edit-tutor/{id}")
124       public ResponseEntity<?> editTutor(@PathVariable String id, @RequestBody User updatedTutor) {
125           boolean updated = adminService.updateTutorProfile(id, updatedTutor);
126           if (updated) {
127               return ResponseEntity.ok("{\"success\":true}");
128           } else {
129               return ResponseEntity.badRequest().body("{\"success\":false,\"message\":\"Tutor not found or update failed\"}");
130           }
131       }
132   }
```

Tutor Controller

```
1
2   package com.tutorconnect.backend.controller;
3
4   import com.tutorconnect.backend.service.AdminService;
5   import org.springframework.beans.factory.annotation.Autowired;
6   import org.springframework.http.ResponseEntity;
7   import org.springframework.security.access.prepost.PreAuthorize;
8   import org.springframework.web.bind.annotation.*;
9   import org.springframework.security.core.annotation.AuthenticationPrincipal;
10  import org.springframework.security.core.context.SecurityContextHolder;
11  import java.util.HashMap;
12
13  @RestController
14  @RequestMapping("/api/tutor")
15  @CrossOrigin(origins = "*", allowedHeaders = "*")
16  public class TutorController {
17      @Autowired
18      private AdminService adminService;
19
20      @PreAuthorize("hasRole('TUTOR')")
21      @GetMapping("/courses")
```

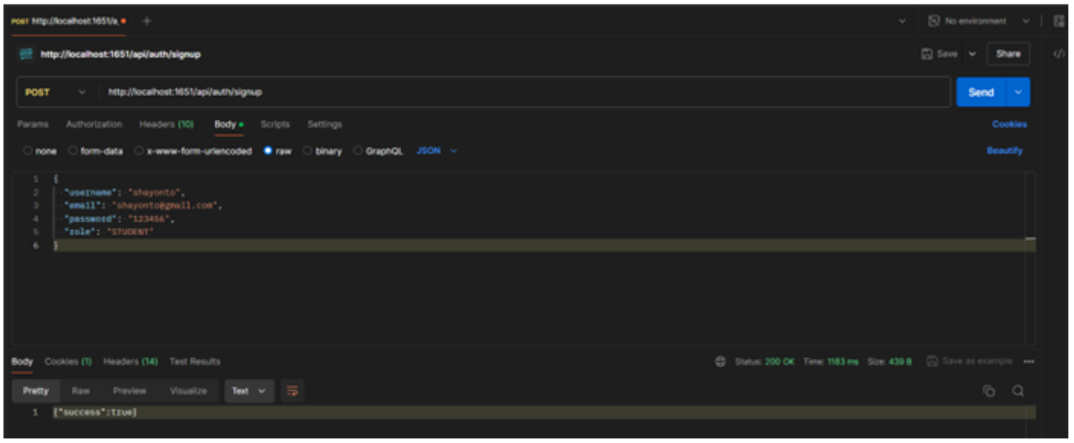
TutorConnect-Personalized-online-tutoring-and-course-delivery-system / Backend / backend / src / main / java / com / tutorconnect / backend / controller / TutorController.java

Code | Blame | 83 lines (79 loc) · 3.91 KB

```
22      public ResponseEntity<?> getAssignedCourses(@AuthenticationPrincipal String email) {
28          }
29          var courses = adminService.getAllCourses();
30          final String finalTutorEmail = tutorEmail;
31          var assignedCourses = courses.stream()
32              .filter(c -> finalTutorEmail != null && finalTutorEmail.equalsIgnoreCase(c.getTutorId()))
33              .map(c -> {
34                  var map = new HashMap<String, Object>();
35                  map.put("id", c.getId());
36                  map.put("title", c.getTitle());
37                  map.put("description", c.getDescription());
38                  map.put("tutorId", c.getTutorId());
39                  map.put("subjects", c.getSubjects());
40                  map.put("language", c.getLanguage());
41                  map.put("price", c.getPrice());
42                  map.put("duration", c.getDuration());
43                  map.put("studentsEnrolled", c.getStudentsEnrolled());
44                  map.put("createdAt", c.getCreatedAt());
45                  map.put("extra", c.getExtra());
46                  return map;
47              }).toList();
48          return ResponseEntity.ok(assignedCourses);
49      }
50      @PreAuthorize("hasRole('TUTOR')")
51      @PutMapping("/courses/{courseId}/resources")
52      public ResponseEntity<?> updateCourseResources(
53          @PathVariable String courseId,
54          @RequestBody java.util.Map<String, Object> resources,
55          @AuthenticationPrincipal String email
56      ) {
57          String tutorEmail = email;
58          if (tutorEmail == null) {
59              Object principal = org.springframework.security.core.context.SecurityContextHolder.getContext().getAuthentication().getPrincipal();
60              tutorEmail = principal instanceof String ? (String) principal : null;
61          }
62          var courseOpt = adminService.getAllCourses().stream()
63              .filter(c -> c.getId().equals(courseId))
64              .findFirst();
65          if (courseOpt.isEmpty()) {
66              return ResponseEntity.status(404).body(java.util.Map.of("success", false, "message", "Course not found"));
67          }
68          var course = courseOpt.get();
69          if (tutorEmail == null || !tutorEmail.equalsIgnoreCase(course.getTutorId())) {
70              return ResponseEntity.status(403).body(java.util.Map.of("success", false, "message", "Not authorized"));
71          }
72          // Accept quizzes as part of resources: quizzes should be a List<Map<String, Object>>
73          // Example quiz: { "question": "...", "options": ["A", "B", "C"], "answer": "A" }
74          // Merge resources into existing extra, preserving students
75          var extra = course.getExtra() instanceof java.util.Map ? (java.util.Map<String, Object>) course.getExtra() : new java.util.HashMap<>();
76          if (extra.containsKey("students")) {
77              resources.put("students", extra.get("students"));
78          }
79          course.setExtra(resources);
80          adminService.saveCourse(course);
81          return ResponseEntity.ok(java.util.Map.of("success", true));
82      }
83  }
```

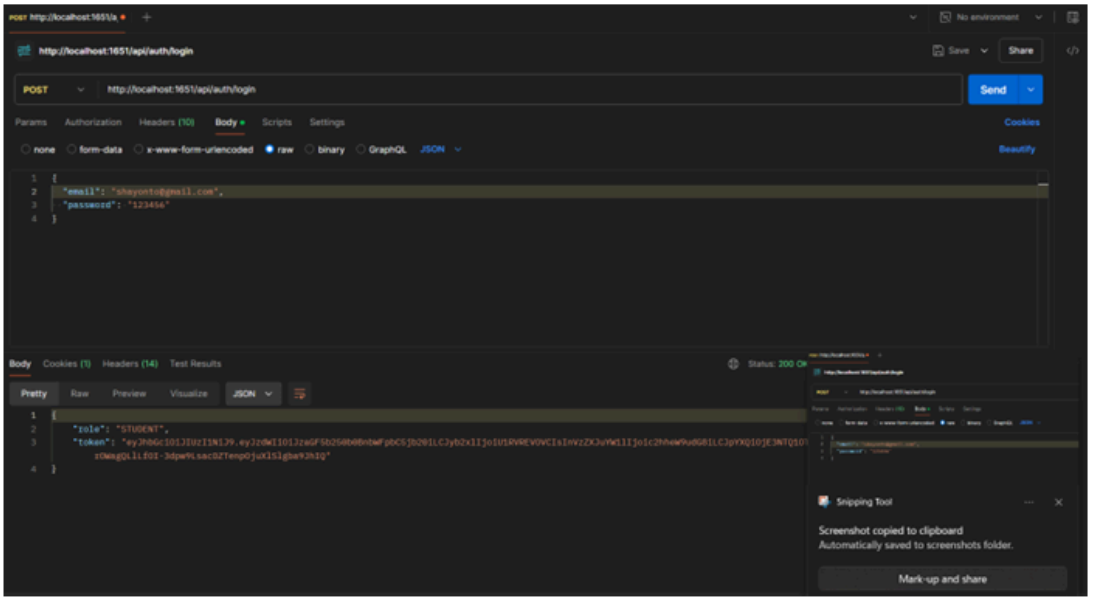
User can sign up with necessary information

```java
@CrossOrigin(origins = "*")
public class AuthController {
    @Autowired  4 usages
    private UserService userService;

    @Autowired  1 usage
    private JwtUtil jwtUtil;

    @PostMapping("/signup")  no usages
    public ResponseEntity<?> register(@RequestBody User user) {
        if (userService.existsByEmail(user.getEmail())) {
            return ResponseEntity.badRequest().body("{\"success\":false,\"message\":\"Email already exists\"}");
        }
        userService.registerUser(user);
        return ResponseEntity.ok( body: "{\"success\":true}");
    }
```



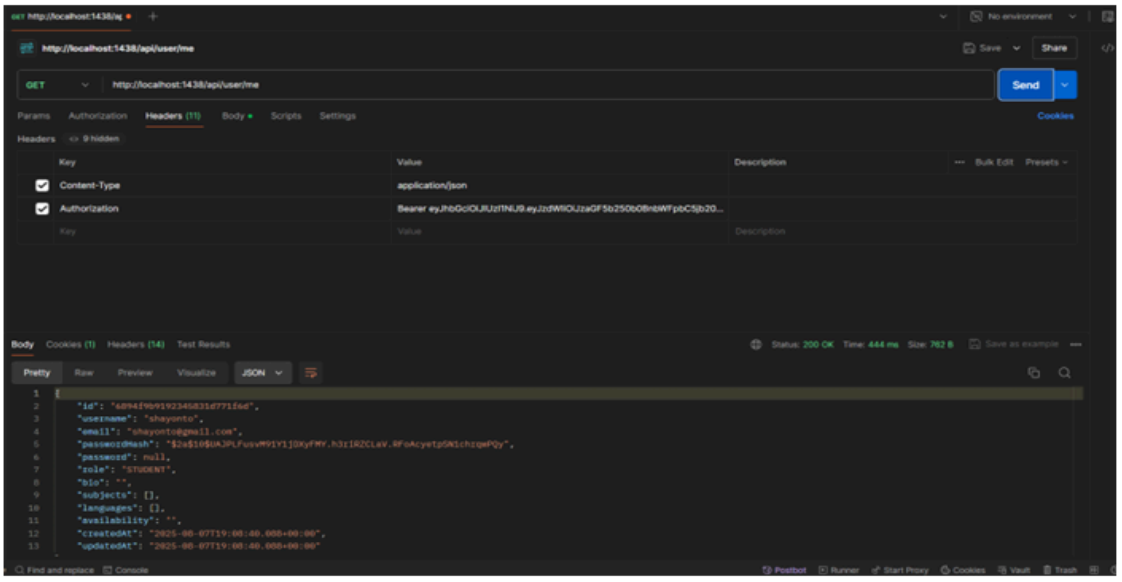
User can log in with necessary information

```java
@PostMapping("/login")  no usages
public ResponseEntity<?> login(@RequestBody Map<String, String> loginData) {
    String email = loginData.get("email");
    String password = loginData.get("password");
    Optional<User> userOpt = userService.findByEmail(email);
    if (userOpt.isEmpty()) {
        return ResponseEntity.status(401).body(Map.of( k1: "message", v1: "Invalid credentials"));
    }
    User user = userOpt.get();
    if (!userService.checkPassword(password, user.getPasswordHash())) {
        return ResponseEntity.status(401).body(Map.of( k1: "message", v1: "Invalid credentials"));
    }
    String token = jwtUtil.generateToken(user);
    return ResponseEntity.ok(Map.of(
        k1: "token", token,
        k2: "role", user.getRole()
    ));
}
}
```



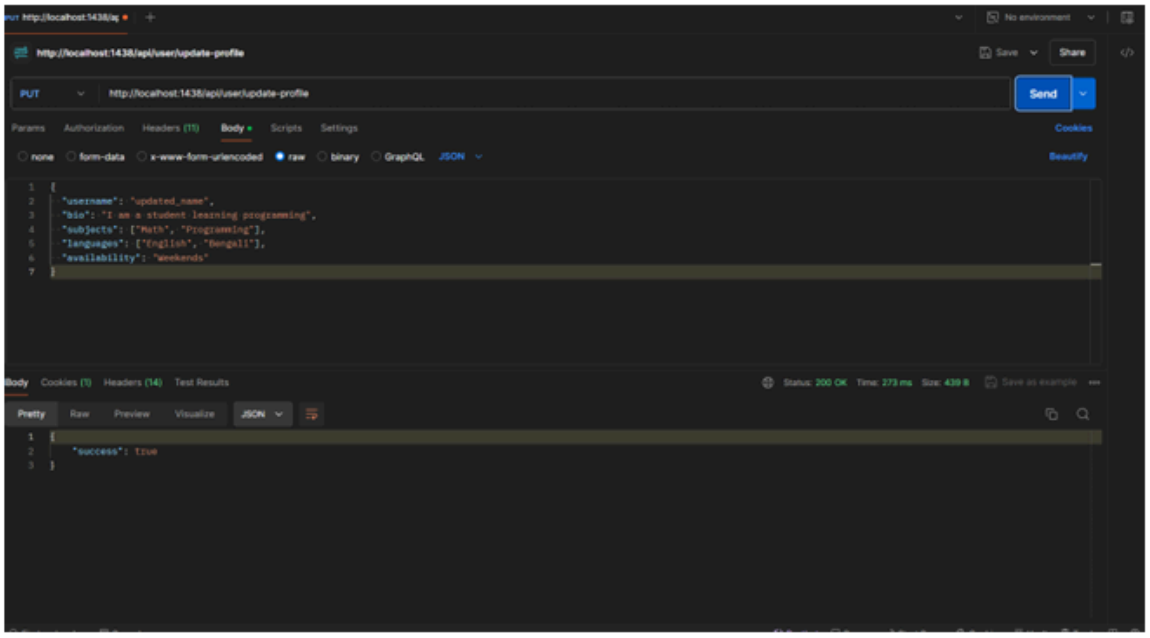
User can see its current profile

```java
@GetMapping("/me")  no usages
public ResponseEntity<?> getProfile(@AuthenticationPrincipal String email) {
    if (email == null) {
        return ResponseEntity.status(401).body(Map.of( k1: "message",  v1: "Unauthorized"));
    }
    Optional<User> userOpt = userService.findByEmail(email);
    if (userOpt.isPresent()) {
        return ResponseEntity.ok(userOpt.get());
    } else {
        return ResponseEntity.status(404).body(Map.of( k1: "message",  v1: "User not found"));
    }
}
```



User can update its profile information like username, bio, subjects etc

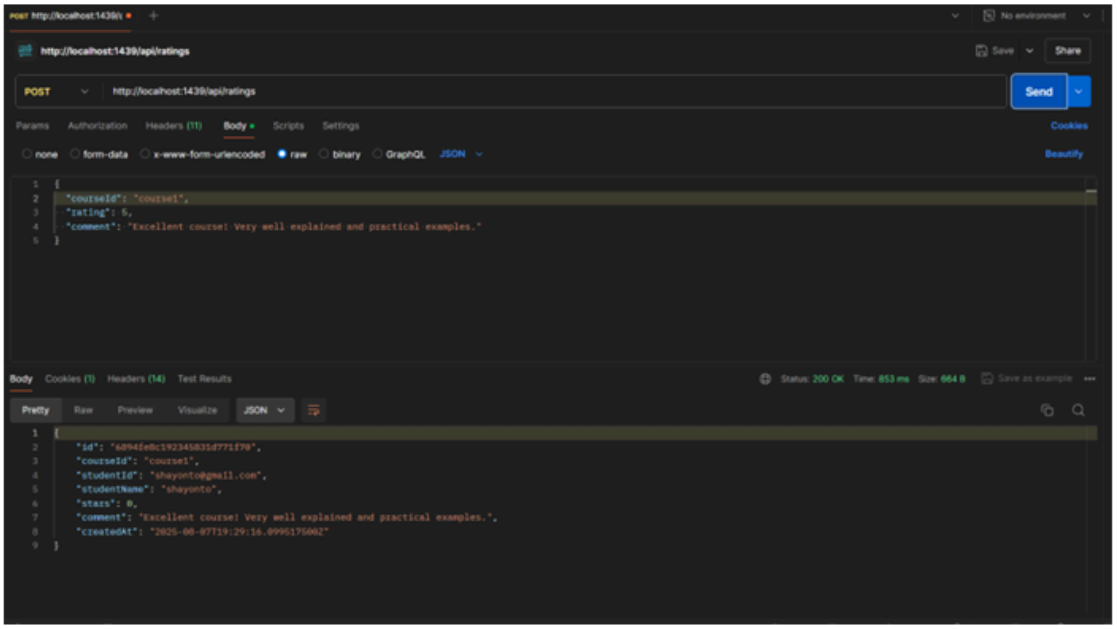
```java
@PutMapping("/update-profile")  no usages
public ResponseEntity<?> updateProfile(@AuthenticationPrincipal String email, @RequestBody Map<String, Obj
    if (email == null) {
        return ResponseEntity.status(401).body(Map.of( k1: "message",  v1: "Unauthorized"));
    }
    Optional<User> userOpt = userService.findByEmail(email);
    if (userOpt.isEmpty()) {
        return ResponseEntity.status(404).body(Map.of( k1: "success",  v1: false,  k2: "message",  v2: "User no
    }
    User user = userOpt.get();
    if (updates.containsKey("username")) user.setUsername((String) updates.get("username"));
    if (updates.containsKey("email")) user.setEmail((String) updates.get("email"));
    if (updates.containsKey("bio")) user.setBio((String) updates.get("bio"));
    if (updates.containsKey("subjects")) {
        var subjRaw = updates.get("subjects");
        if (subjRaw instanceof java.util.List<?>) {
            java.util.List<?> rawList = (java.util.List<?>) subjRaw;
            java.util.List<String> strList = rawList.stream().map(Object::toString).toList();
            user.setSubjects(strList);
        }
    }
    if (updates.containsKey("languages")) {
        var langRaw = updates.get("languages");
        if (langRaw instanceof java.util.List<?>) {
            java.util.List<?> rawList = (java.util.List<?>) langRaw;
            java.util.List<String> strList = rawList.stream().map(Object::toString).toList();
            user.setLanguages(strList);
        }
```



User (Student and Loged in) can add Review/Comment/ Rating to courses

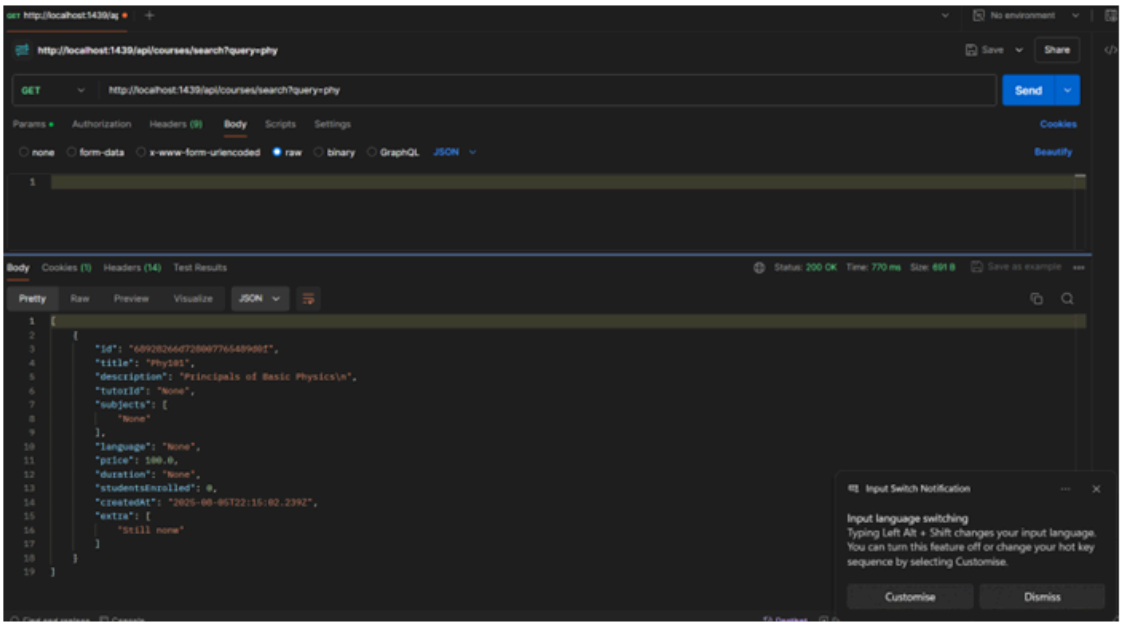
```java
@RestController  no usages
@RequestMapping("/api/ratings")
public class RatingController {
    private final RatingService ratingService;  4 usages

    public RatingController(RatingService ratingService) { this.ratingService = ratingService; }

    // Only allow students to add ratings
    @PostMapping  no usages
    public Rating addRating(@RequestBody Rating rating, Authentication authentication, HttpServletRequest requ
        boolean isStudent = authentication.getAuthorities().stream() Stream<capture of extends GrantedAuthority>
            .map(GrantedAuthority::getAuthority) Stream<String>
            .anyMatch(role -> role.equals("ROLE_STUDENT"));
        if (!isStudent) {
            throw new RuntimeException("Only students can add ratings.");
        }

        // Extract username from JWT token
        String authHeader = request.getHeader( s: "Authorization");
        String username = authentication.getName(); // fallback to email
        if (authHeader != null && authHeader.startsWith("Bearer ")) {
            try {
                String token = authHeader.substring( beginIndex: 7);
                Claims claims = Jwts.parserBuilder() JwtParserBuilder
                    .setSigningKey(java.util.Base64.getDecoder().decode( src: "bXlTdXBlclNlY3JldEtleU1ha2VJdFNh
                    .build() JwtParser
                    .parseClaimsJws(token) Jws<Claims>
```

```java
15    public class RatingController {                                                    ⚠5 ✓2 ^
24        public Rating addRating(@RequestBody Rating rating, Authentication authentication, HttpServletRequest r
37                String token = authHeader.substring( beginIndex: 7);
38                Claims claims = Jwts.parserBuilder() JwtParserBuilder
39                    .setSigningKey(java.util.Base64.getDecoder().decode( src: "bXlTdXBlclNlY3JldEtleU1ha2VJdF
40                    .build() JwtParser
41                    .parseClaimsJws(token) Jws<Claims>
42                    .getBody();
43                username = claims.get( s: "username", String.class);
44                if (username == null) username = authentication.getName();
45            } catch (Exception e) {
46                username = authentication.getName();
47            }
48        }
49
50        rating.setStudentId(authentication.getName());
51        rating.setStudentName(username);
52        rating.setCreatedAt(java.time.Instant.now());
53        return ratingService.addRating(rating);
54    }
```

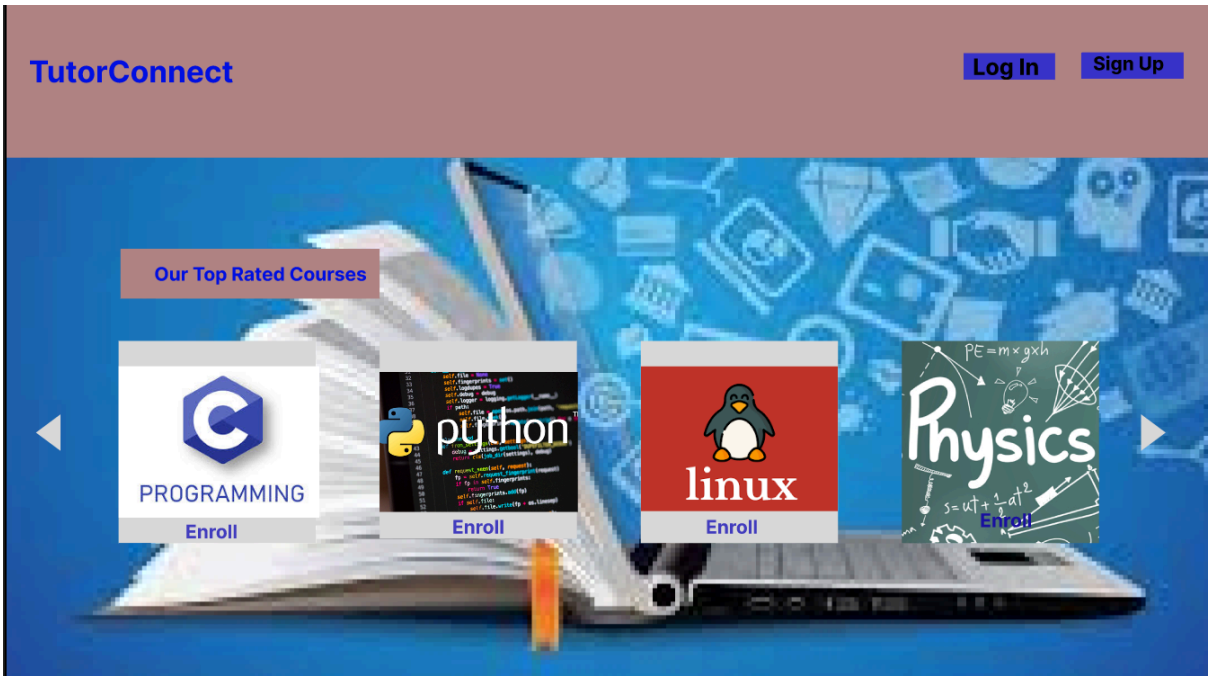


User can search courses for enrolling or posting review

```java
package com.tutorconnect.backend.controller;

import ...

@RestController  no usages
@RequestMapping("/api/courses")
public class CourseController {
    private final CourseService courseService;  3 usages

    public CourseController(CourseService courseService) { this.courseService = courseService; }

    @GetMapping("/search")  no usages
    public List<Course> searchCourses(@RequestParam String query) { return courseService.searchCoursesByTitle(

    @GetMapping("/{id}")  no usages
    public Course getCourseById(@PathVariable String id) { return courseService.getCourseById(id); }
}
```



5. User Interface Design



TUTORCONNECT

Courses that fit
Tutors that care

Home | Course | Chat

★ Unlock the power of personalized education with our cutting edge online tutoring and course delivery platform.

★ Designed to adapt, built to scale; we make learning efficient, engaging, and results driven.

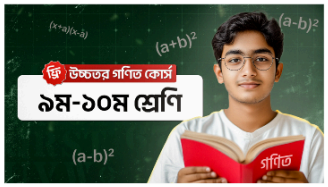
Join Us Now!

TUTORCONNECT

Courses that fit
Tutors that care

Home | Course | Chat

EXPLORE PROGRAMS



See Details

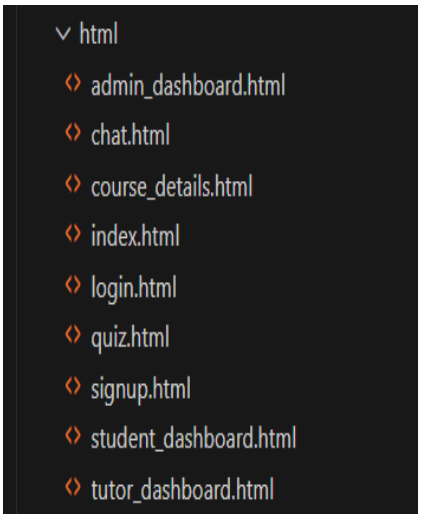
See Details

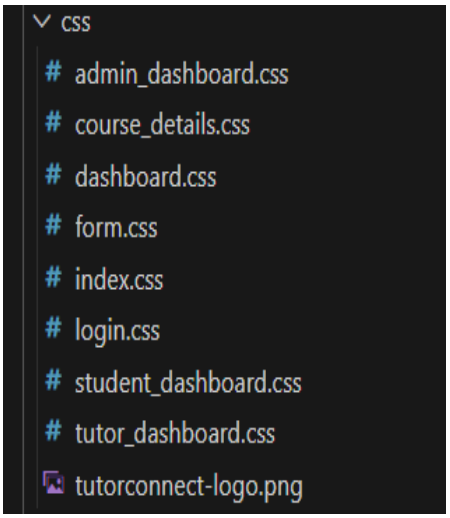
See Details

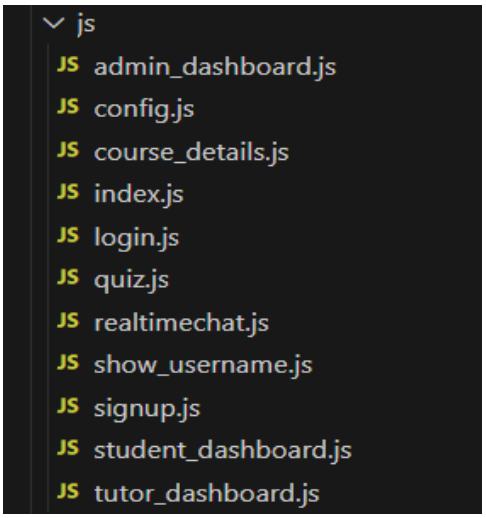
Figma Link:

https://www.figma.com/proto/aSYYQAR4GWliVR6lIAAl2R/22101438_Md.Ann-Am-Akbar-Saad_Sec09_Assignment2?node-id=1-2&p=f&t=IpDlCN3aQLJSgay7-0&scaling=scale-down&content-scaling=fixed&page-id=0%3A1&starting-point-node-id=1%3A2

6. Frontend Development







Login and Signup:

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Log In - TutorConnect</title>
  <link rel="stylesheet" href="../css/form.css">
</head>
<body>
  <div class="form-container">
    <h2>Log In</h2>
    <form id="loginForm">
      <input type="email" name="email" placeholder="Email" required>
      <input type="password" name="password" placeholder="Password" required>
      <button type="submit" class="login-btn">Log In</button>
    </form>
    <p>Don't have an account? <a href="signup.html">Sign Up</a></p>
  </div>
  <script src="../js/login.js"></script>
  <script src="../js/show_username.js"></script>
</body>
</html>
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Sign Up - TutorConnect</title>
  <link rel="stylesheet" href="../css/index.css">
  <link rel="stylesheet" href="../css/form.css">
</head>
<body>
  <div class="form-container">
    <h2>Sign Up</h2>
    <form id="signupForm">
      <input type="text" name="username" placeholder="Username" required>
      <input type="email" name="email" placeholder="Email" required>
      <input type="password" name="password" placeholder="Password" required>
      <button type="submit" class="register-btn">Sign Up</button>
    </form>
    <p>Already have an account? <a href="login.html">Log In</a></p>
  </div>
  <script src="../js/signup.js"></script>
  <script src="../js/show_username.js"></script>
</body>
</html>
```

html

```javascript
// login.js
// API_BASE is provided by js/config.js

document.getElementById('loginForm').addEventListener('submit', function(e) {
  e.preventDefault();
  const form = e.target;
  const data = {
    email: form.email.value,
    password: form.password.value
  };
  fetch(`${API_BASE}/api/auth/login`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(data)
  })
  .then(res => res.json())
  .then(result => {
    if(result.token) {
      alert('Login successful!');
      localStorage.setItem('jwt', result.token);
      // Redirect based on role
      if(result.role === 'STUDENT') {
        window.location.href = 'student_dashboard.html';
      } else if(result.role === 'TUTOR') {
        window.location.href = 'tutor_dashboard.html';
      } else if(result.role === 'ADMIN') {
        window.location.href = 'admin_dashboard.html';
      } else {
        window.location.href = 'dashboard.html';
      }
    } else {
      alert(result.message || 'Login failed');
    }
  })
  .catch(() => alert('Error connecting to server'));
});
```

```javascript
// signup.js

document.getElementById('signupForm').addEventListener('submit', function(e) {
  e.preventDefault();
  const form = e.target;
  const data = {
    username: form.username.value,
    email: form.email.value,
    passwordHash: form.password.value
  };
  fetch(`${API_BASE}/api/auth/signup`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(data)
  })
  .then(res => res.json())
  .then(result => {
    if(result.success) {
      alert('Sign up successful! Please log in.');
      window.location.href = 'login.html';
    } else {
      alert(result.message || 'Sign up failed');
    }
  })
  .catch(() => alert('Error connecting to server'));
});
```

Js

```css
/* Add form styles for signup and login pages */
.form-container {
  max-width: 400px;
  margin: 3rem auto;
  background: #fff;
  padding: 2rem;
  border-radius: 8px;
  box-shadow: 0 2px 8px rgba(0,0,0,0.08);
  text-align: center;
}
.form-container h2 {
  margin-bottom: 1.5rem;
}
.form-container input {
  width: 90%;
  padding: 0.7rem;
  margin-bottom: 1rem;
  border-radius: 4px;
  border: 1px solid #ccc;
  font-size: 1rem;
}
.form-container button {
  width: 100%;
  padding: 0.7rem;
  background: #ff6600;
  color: #fff;
  border: none;
  border-radius: 4px;
  font-size: 1rem;
  font-weight: bold;
  cursor: pointer;
}
.form-container a {
  color: #003366;
  text-decoration: underline;
}
```

Css

Student_dashboard:

After successfully completing login, student can see his dashboard

Frontend > html > student_dashboard.html > ...

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Student Dashboard - TutorConnect</title>
  <link rel="stylesheet" href="../css/index.css">
  <script src="https://www.youtube.com/iframe_api"></script>
</head>
<body>
  <!-- Top Navigation Bar -->
  <div class="top-nav">
    <div class="nav-brand">
      <img src="../css/tutorconnect-logo.png" alt="TutorConnect Logo" class="course-logo">
      <h3>TutorConnect</h3>
    </div>
    <button id="logoutBtn" class="logout-btn">  Log Out</button>
  </div>

  <div class="main-container">
    <!-- Main Dashboard -->
    <div id="mainDashboard">
      <div class="welcome-section">
        <h1>Welcome Back, Student!  </h1>
        <p>Your personalized learning dashboard awaits</p>
      </div>
      <div class="dashboard-buttons">
        <button id="continueLearningBtn" class="dashboard-btn primary">
          <span class="btn-icon">  </span>
          <span class="btn-text">Continue Learning</span>
        </button>
        <button id="browseCoursesBtn" class="dashboard-btn secondary">
          <span class="btn-icon">  </span>
          <span class="btn-text">Browse Courses</span>
        </button>
        <button id="certificatesBtn" class="dashboard-btn certificate">
          <span class="btn-icon">  </span>
          <span class="btn-text">Certificates</span>
        </button>
```

```
<!-- Browse Courses Section -->
<div id="browseCoursesSection" style="display:none;">
  <div class="section-header">
    <h2>🔍 Browse Courses</h2>
    <button id="backToMainFromBrowseBtn" class="back-btn">← Back to Dashboard</button>
  </div>
  <div class="search-container">
    <input type="text" id="studentSearchBar" class="search-input" placeholder="🔍 Search our courses...">
  </div>
  <div id="studentSearchResults" class="courses-grid"></div>
</div>

<!-- Certificates Section -->
<div id="certificatesSection" style="display:none;">
  <div class="section-header">
    <h2>🏆 My Certificates</h2>
    <button id="backToMainFromCertificatesBtn" class="back-btn">← Back to Dashboard</button>
  </div>
  <div id="certificatesList" class="courses-grid">Loading certificates...</div>
</div>

<!-- Edit Profile Section -->
<div id="editProfileSection" style="display:none;">
  <div class="section-header">
    <h2>👤 Edit Profile</h2>
    <button id="backToMainFromProfileBtn" class="back-btn">← Back to Dashboard</button>
  </div>
  <div id="profileEditContainer" class="profile-container"></div>
</div>
```

Html

```
// API_BASE is provided by js/config.js

// New Student Dashboard with improved navigation structure
let currentCourse = null;

document.addEventListener('DOMContentLoaded', function() {
  initializeDashboard();
});

function initializeDashboard() {
  // Main dashboard navigation
  setupMainNavigation();

  // Load user profile data
  loadUserProfile();
}

function setupMainNavigation() {
  // RAG Chatbot button
  document.getElementById('ragChatbotBtn').addEventListener('click', function() {
    const modal = document.getElementById('ragChatbotModal');
    const loading = document.getElementById('ragChatbotLoading');
    const iframe = document.getElementById('ragChatbotIframe');
    modal.style.display = 'flex';
    loading.style.display = 'flex';
    iframe.style.display = 'none';
    // Remove any previous error message
    if (document.getElementById('ragChatbotError')) {
      document.getElementById('ragChatbotError').remove();
    }
    // Listen for iframe load or error
    iframe.onload = function() {
      loading.style.display = 'none';
      iframe.style.display = 'block';
```

```
  document.getElementById('backToCoursesBtn').addEventListener('click', function() {
    showSection('continueLearningSection');
  });

  document.getElementById('backToMainFromBrowseBtn').addEventListener('click', function() {
    showSection('mainDashboard');
  });

  document.getElementById('backToMainFromProfileBtn').addEventListener('click', function() {
    showSection('mainDashboard');
  });

  document.getElementById('backToMainFromCertificatesBtn').addEventListener('click', function() {
    showSection('mainDashboard');
  });
}

function setupCourseActionButtons() {
  document.getElementById('videoBtn').addEventListener('click', function() {
    showCourseContent('video');
  });

  document.getElementById('documentBtn').addEventListener('click', function() {
    showCourseContent('document');
  });

  document.getElementById('assignmentBtn').addEventListener('click', function() {
    showCourseContent('assignment');
  });

  document.getElementById('practiceBtn').addEventListener('click', function() {
    showCourseContent('practice');
  });

  document.getElementById('requestSessionBtn').addEventListener('click', function() {
```

Js

```
.dashboard-btn:hover {
  transform: translateY(-5px);
  box-shadow: 0 15px 35px rgba(0, 0, 0, 0.2);
}

.dashboard-btn.primary { border-left: 5px solid #27ae60; }
.dashboard-btn.secondary { border-left: 5px solid #3498db; }
.dashboard-btn.tertiary { border-left: 5px solid #e67e22; }
.dashboard-btn.quaternary { border-left: 5px solid #9b59b6; }
.dashboard-btn.fifth { border-left: 5px solid #e91e63; }
.dashboard-btn.sixth { border-left: 5px solid #607d8b; }

.btn-icon {
  font-size: 2.5rem;
}

.btn-text {
  font-size: 1.1rem;
  font-weight: 600;
  color: #2c3e50;
}

.section-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 30px;
  background: rgba(255, 255, 255, 0.95);
  padding: 20px 30px;
  border-radius: 15px;
  box-shadow: 0 5px 20px rgba(0, 0, 0, 0.1);
}

.section-header h2 {
  color: #2c3e50;
  font-size: 1.8rem;
  font-weight: 600;
}
```

Css

Tutor_dashboard:

After successfully completing login, tutor can see his dashboard

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Tutor Dashboard - TutorConnect</title>
  <link rel="stylesheet" href="../css/index.css">
</head>
<body>
  <!-- Top Navigation Bar -->
  <div class="top-nav">
    <div class="nav-brand">
      <img src="../css/tutorconnect-logo.png" alt="TutorConnect Logo" class="course-logo">
      <h3>TutorConnect</h3>
    </div>
    <button id="logoutBtn" class="logout-btn">  Log Out</button>
  </div>

  <div class="main-container">
    <!-- Main Dashboard -->
    <div id="mainDashboard">
      <div class="welcome-section">
        <h1>Welcome Back, Tutor! </h1>
        <p>Your teaching control panel awaits</p>
      </div>
      <div class="dashboard-buttons">
        <button id="assignedCoursesBtn" class="dashboard-btn primary">
          <span class="btn-icon"></span>
          <span class="btn-text">My Assigned Courses</span>
        </button>
        <button id="generateLinkBtn" class="dashboard-btn secondary">
          <span class="btn-icon">🔗</span>
          <span class="btn-text">Generate Link</span>
        </button>
        <button id="editProfileBtn" class="dashboard-btn tertiary">
          <span class="btn-icon">👤</span>
          <span class="btn-text">Edit Profile</span>
        </button>
        <button id="chatBtn" class="dashboard-btn quaternary">
```

```html
    <!-- Course Details Section -->
    <div id="courseDetailsSection" style="display:none;">
      <div class="section-header">
        <h2 id="courseDetailsTitle">  Course Details</h2>
        <button id="backToCoursesBtn" class="back-btn">← Back to Courses</button>
      </div>
      <div class="course-action-buttons">
        <button id="enrolledStudentsBtn" class="course-action-btn primary">
          <span class="btn-icon">👥</span>
          <span>Enrolled Students</span>
        </button>
        <button id="editResourcesBtn" class="course-action-btn secondary">
          <span class="btn-icon">📝</span>
          <span>Edit/Add Resources</span>
        </button>
        <button id="sessionRequestsBtn" class="course-action-btn tertiary">
          <span class="btn-icon">🔔</span>
          <span>Session Requests</span>
        </button>
      </div>
      <div id="courseActionContent" class="admin-container"></div>
    </div>

    <!-- Generate Link Section -->
    <div id="generateLinkSection" style="display:none;">
      <div class="section-header">
        <h2>🔗 Generate Jitsi Meet Link</h2>
        <button id="backToMainFromLinkBtn" class="back-btn">← Back to Dashboard</button>
      </div>
      <div class="admin-container">
        <div class="link-generator">
          <button id="generateJitsiBtn" class="generate-btn">Generate New Meeting Link</button>
          <div id="jitsiLinkContainer" class="link-display"></div>
        </div>
      </div>
```

Html

```
// API_BASE is provided by js/config.js

// New Tutor Dashboard with improved navigation structure - Fixed version
document.addEventListener('DOMContentLoaded', function() {
  initializeTutorDashboard();
});

function initializeTutorDashboard() {
  setupTutorNavigation();
  setupJitsiButtonHandler();
}

function setupTutorNavigation() {
  // Assigned Courses button
  document.getElementById('assignedCoursesBtn').addEventListener('click', function() {
    showTutorSection('assignedCoursesSection');
    loadAssignedCourses();
  });

  // Generate Link button
  document.getElementById('generateLinkBtn').addEventListener('click', function() {
    showTutorSection('generateLinkSection');
    setupLinkGenerator();
  });

  // Edit Profile button
  document.getElementById('editProfileBtn').addEventListener('click', function() {
    showTutorSection('editProfileSection');
    loadEditProfileForm();
  });

  // Chat button
  document.getElementById('chatBtn').addEventListener('click', function() {
    window.location.href = 'chat.html';
  });

  // RAG Chatbot button
  document.getElementById('ragChatbotBtn').addEventListener('click', function() {
    const modal = document.getElementById('ragChatbotModal');
```

```
function showTutorSection(sectionId) {
  // Hide all sections
  const sections = ['mainDashboard', 'assignedCoursesSection', 'courseDetailsSection', 'generateLinkSection', 'editProfileSection'];
  sections.forEach(id => {
    const element = document.getElementById(id);
    if (element) element.style.display = 'none';
  });

  // Show the requested section
  const targetElement = document.getElementById(sectionId);
  if (targetElement) targetElement.style.display = 'block';
}

let currentCourseId = null;
let currentCourseTitle = '';

function loadAssignedCourses() {
  const token = localStorage.getItem('jwt');
  if (!token) {
    alert('You are not logged in. Please log in first.');
    window.location.href = 'login.html';
    return;
  }

  const coursesList = document.getElementById('coursesList');
  if (!coursesList) return;

  coursesList.innerHTML = '<p style="text-align: center; margin: 40px 0;">Loading courses...</p>';

  fetch(`${API_BASE}/api/tutor/courses`, {
    headers: { 'Authorization': 'Bearer ' + token }
```

Js

```css
.logout-btn:hover {
  background: #c0392b;
  transform: translateY(-2px);
  box-shadow: 0 6px 20px rgba(231, 76, 60, 0.4);
}

.main-container {
  max-width: 1200px;
  margin: 0 auto;
  padding: 40px 20px;
}

.welcome-section {
  text-align: center;
  margin-bottom: 50px;
  color: white;
}

.welcome-section h1 {
  font-size: 2.5rem;
  margin-bottom: 10px;
  font-weight: 700;
}

.welcome-section p {
  font-size: 1.2rem;
  opacity: 0.9;
}

.dashboard-buttons {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(280px, 1fr));
  gap: 25px;
  margin: 40px 0;
}
```

Css

Admin_dashboard:

After successfully completing login, admin can see his dashboard

```html
<!-- Create User Section -->
<div id="createUserSection" style="display:none;">
  <div class="section-header">
    <h2>➕ Create User/Tutor</h2>
    <button id="backToMainFromCreateBtn" class="back-btn">← Back to Dashboard</button>
  </div>
  <div class="admin-container">
    <div id="adminForms"></div>
    <div id="result" style="text-align:center;margin-top:20px;"></div>
  </div>
</div>

<!-- Edit Tutor Section -->
<div id="editTutorSection" style="display:none;">
  <div class="section-header">
    <h2>✏️ Edit Tutor Profile</h2>
    <button id="backToMainFromEditBtn" class="back-btn">← Back to Dashboard</button>
  </div>
  <div class="admin-container">
    <div id="editTutorForms"></div>
    <div id="editTutorResult" style="text-align:center;margin-top:20px;"></div>
  </div>
</div>

<!-- Manage Courses Section -->
<div id="manageCourseSection" style="display:none;">
  <div class="section-header">
    <h2>📚 Manage Courses</h2>
    <button id="backToMainFromCourseBtn" class="back-btn">← Back to Dashboard</button>
  </div>
  <div class="admin-container">
    <div id="courseForms"></div>
    <div id="courseResult" style="text-align:center;margin-top:20px;"></div>
  </div>
```

Html

```
async function fetchAndDisplayCourses(type, query) {
  const token = localStorage.getItem('jwt');
  let url = '';
  if(type === 'all') url = `${API_BASE}/api/admin/courses`;
  else url = `${API_BASE}/api/admin/search-course?query=` + encodeURIComponent(query);
  const res = await fetch(url, {
    headers: { 'Authorization': 'Bearer ' + token }
  });
  let courses = [];
  if(res.ok) courses = await res.json();
  const listDiv = document.getElementById('courseList');
  if(courses.length === 0) {
    listDiv.innerHTML = '<p>No courses found.</p>';
    return;
  }
  let html = '<table border="1" style="width:100%;text-align:left;"><tr>' +
    '<th>Title</th><th>Description</th><th>Tutor ID</th><th>Subjects</th><th>Language</th><th>Price</th><th>Duration</th><th>Enrolled</th><
  courses.forEach(c => {
    html += `<tr>
      <td>${c.title}</td>
      <td>${c.description}</td>
      <td>${c.tutorId}</td>
      <td>${Array.isArray(c.subjects) ? c.subjects.join(', ') : ''}</td>
      <td>${c.language}</td>
      <td>${c.price}</td>
      <td>${c.duration}</td>
      <td>${c.studentsEnrolled}</td>
      <td>${Array.isArray(c.extra) ? c.extra.join(', ') : ''}</td>
      <td><button onclick="editCourse('${c.id}', '${encodeURIComponent(JSON.stringify(c))}')">Edit</button></td>
    </tr>`;
  });
  html += '</table>';
  listDiv.innerHTML = html;
}
```

```
async function fetchAndDisplayTutors(type, query) {
  const token = localStorage.getItem('jwt');
  let url = '';
  if(type === 'all') url = `${API_BASE}/api/admin/tutors`;
  else url = `${API_BASE}/api/admin/search-tutor?query=` + encodeURIComponent(query);
  const res = await fetch(url, {
    headers: { 'Authorization': 'Bearer ' + token }
  });
  let tutors = [];
  if(res.ok) tutors = await res.json();
  const listDiv = document.getElementById('tutorList');
  if(tutors.length === 0) {
    listDiv.innerHTML = '<p>No tutors found.</p>';
    return;
  }
  let html = '<table border="1" style="width:100%;text-align:left;"><tr>' +
    '<th>Username</th><th>Email</th><th>Bio</th><th>Subjects</th><th>Languages</th><th>Availability</th><th>Edit</th></tr>';
  tutors.forEach(t => {
    html += `<tr>
      <td>${t.username}</td>
      <td>${t.email}</td>
      <td>${t.bio}</td>
      <td>${Array.isArray(t.subjects) ? t.subjects.join(', ') : ''}</td>
      <td>${Array.isArray(t.languages) ? t.languages.join(', ') : ''}</td>
      <td>${t.availability}</td>
      <td><button onclick="editTutor('${t.id}', '${encodeURIComponent(JSON.stringify(t))}')">Edit</button></td>
    </tr>`;
  });
  html += '</table>';
  listDiv.innerHTML = html;
}
```

Js

```css
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

body {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
  min-height: 100vh;
  color: #333;
}

.top-nav {
  background: rgba(255, 255, 255, 0.95);
  backdrop-filter: blur(10px);
  padding: 15px 30px;
  display: flex;
  justify-content: space-between;
  align-items: center;
  box-shadow: 0 2px 20px rgba(0, 0, 0, 0.1);
  position: sticky;
  top: 0;
  z-index: 1000;
}

.nav-brand {
  display: flex;
  align-items: center;
  gap: 15px;
}

.nav-brand h3 {
  color: #667eea;
  font-weight: 700;
  font-size: 1.5rem;
  margin: 0;
}
```

Css

Course_details:
Students can view course details after log in

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Course Details - TutorConnect</title>
  <link rel="stylesheet" href="../css/course_details.css">
</head>
<body>
  <header class="header">
    <div class="header-content">
      <div class="logo-container">
        <img src="../css/tutorconnect-logo.png" alt="TutorConnect Logo" class="course-logo">
        <h1 class="site-title">TutorConnect</h1>
      </div>
      <div class="user-info">
        <span class="user-greeting">Welcome, <span id="usernameDisplay">Student</span>!</span>
        <a href="student_dashboard.html" class="dashboard-btn">🏠 Dashboard</a>
      </div>
    </div>
  </header>

  <div class="main-container">
    <div id="courseDetails" class="course-details-container">
      <!-- Course details will be loaded here -->
    </div>
  </div>

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

Html

```
function renderCourseDetails(course) {
  // Get student info from JWT
  const token = localStorage.getItem('jwt');
  let studentEmail = null;
  let isStudent = false;
  let enrolled = false;
  if (token) {
    try {
      const payload = JSON.parse(atob(token.split('.')[1]));
      studentEmail = payload.sub || payload.email;
      const role = payload.role || payload.authorities;
      isStudent = role === 'STUDENT' || (Array.isArray(role) && role.includes('STUDENT'));
    } catch {}
  }
  if (course.extra && course.extra.students && Array.isArray(course.extra.students) && studentEmail) {
    enrolled = course.extra.students.some(s => s.email === studentEmail);
  }
```

```
function fetchRatingsList() {
  fetch(`${API_BASE}/api/ratings/course/${courseId}`)
    .then(res => res.json())
    .then(ratings => {
      if (!ratings || ratings.length === 0) {
        document.getElementById('ratingsList').innerHTML = '<p>No ratings yet.</p>';
        return;
      }
      document.getElementById('ratingsList').innerHTML = ratings.map(r => `
        <div class="rating-card">
          <span>${renderStars(r.stars)} (${r.stars})</span>
          <p>${r.comment ? r.comment : ''}</p>
          <small>By ${r.studentName || r.studentId} on ${r.createdAt ? new Date(r.createdAt).toLocaleDateString() : ''}</small>
        </div>
      `).join('');
    })
    .catch(() => {
      document.getElementById('ratingsList').innerHTML = '';
    });
}

function setupRatingForm() {
  const form = document.getElementById('ratingForm');
  if (!form) return;
  form.addEventListener('submit', function(e) {
    e.preventDefault();
    const stars = parseInt(document.getElementById('starsInput').value);
    const comment = document.getElementById('commentInput').value;
    const token = localStorage.getItem('jwt');
    if (!token) {
      document.getElementById('ratingError').textContent = 'You must be logged in as a student to submit a rating.';
      return;
    }
    fetch(`${API_BASE}/api/ratings`, {
      method: 'POST',
      headers: {
```

Js

```css
.logo-container {
  display: flex;
  align-items: center;
  gap: 15px;
}

.course-logo {
  width: 40px;
  height: 40px;
  border-radius: 8px;
}

.logo {
  width: 40px;
  height: 40px;
  border-radius: 8px;
  box-shadow: 0 4px 15px rgba(0, 0, 0, 0.2);
}

.site-title {
  color: #4169E1;
  font-size: 1.5rem;
  font-weight: 600;
}

.user-info {
  display: flex;
  align-items: center;
  gap: 15px;
}

.user-greeting {
  color: #333;
  font-size: 16px;
  font-weight: 500;
}
```

Css

Chat:

Students and tutors can do texting with each other among themselves or within a group

```html
<div class="chat-container">
    <div class="chat-list">
        <div class="chat-list-header">
            <div style="display: flex; justify-content: space-between; align-items: center;">
                <span>Your Chats</span>
                <button id="createGroupBtn" class="create-group-btn">+ Group</button>
            </div>

            <div class="chat-tabs">
                <button id="individualTab" class="chat-tab active">Individual</button>
                <button id="groupTab" class="chat-tab">Groups</button>
            </div>

            <input id="userSearchInput" type="text" placeholder="Search users by name or email..."
                   style="margin-top: 10px; padding: 8px 12px; border-radius: 20px; border: 1px solid #ccc; font-size: 14px; width: 100%;
        </div>
        <div id="userSearchResults" style="display: none; border-bottom: 1px solid #dee2e6;"></div>
        <div id="chatList"></div>
    </div>

    <div class="chat-main">
        <div id="chatHeader" class="chat-header">Select a chat to start messaging</div>
        <div id="messages" class="messages">
            <div class="welcome">Select a chat from the left to start real-time messaging between tutors and students</div>
        </div>
        <div id="inputArea" class="input-area" style="display: none;">
            <input type="text" id="messageInput" class="message-input" placeholder="Type a message...">
            <button id="sendBtn" class="send-btn">Send</button>
        </div>
    </div>
</div>
```

```html
<!-- Group Chat Creation Modal -->
<div id="groupChatModal" class="modal">
    <div class="modal-content">
        <div class="modal-header">Create Group Chat</div>

        <form id="groupChatForm">
            <div class="form-group">
                <label for="groupName">Group Name *</label>
                <input type="text" id="groupName" required placeholder="Enter group name">
            </div>

            <div class="form-group">
                <label for="groupDescription">Description</label>
                <textarea id="groupDescription" placeholder="Enter group description (optional)"></textarea>
            </div>

            <div class="form-group">
                <label for="participantSearch">Add Participants</label>
                <input type="text" id="participantSearch" placeholder="Search users by name or email...">
                <div id="participantSearchResults" style="margin-top: 10px;"></div>
            </div>

            <div class="form-group">
                <label>Selected Participants</label>
                <div id="selectedParticipants" class="participants-list">
                    <div style="text-align: center; color: #666; padding: 20px;">No participants selected</div>
                </div>
            </div>

            <div class="modal-actions">
                <button type="button" class="btn-cancel" onclick="closeGroupChatModal()">Cancel</button>
                <button type="submit" class="btn-create">Create Group</button>
            </div>
```

Html

```javascript
// Load individual chats
async function loadIndividualChats() {
    try {
        const token = localStorage.getItem('jwt');
        const response = await fetch(`${API_BASE}/api/chat/my`, {
            headers: { 'Authorization': `Bearer ${token}` }
        });

        if (response.ok) {
            const chats = await response.json();
            window.individualChats = chats;
            console.log('📄 Loaded individual chats:', chats.length);
        }
    } catch (error) {
        console.error('❌ Error loading individual chats:', error);
    }
}

// Load group chats
async function loadGroupChats() {
    try {
        const token = localStorage.getItem('jwt');
        const response = await fetch(`${API_BASE}/api/groupchat/my`, {
            headers: { 'Authorization': `Bearer ${token}` }
        });

        if (response.ok) {
            const groupChats = await response.json();
            window.groupChats = groupChats;
            console.log('📄 Loaded group chats:', groupChats.length);
        } else {
            window.groupChats = [];
        }
    } catch (error) {
        console.error('❌ Error loading group chats:', error);
        window.groupChats = [];
    }
}
```

```js
// Connect to WebSocket
async function connectWebSocket() {
    try {
        console.log('🔗 Connecting to WebSocket...');

        const socket = new SockJS(`${API_BASE}/ws`);
        stompClient = new StompJs.Client({
            webSocketFactory: () => socket,
            debug: (str) => console.log('STOMP:', str),
            onConnect: () => {
                console.log('✅ WebSocket connected');
                isConnected = true;
                updateConnectionStatus(true);
            },
            onDisconnect: () => {
                console.log('❌ WebSocket disconnected');
                isConnected = false;
                updateConnectionStatus(false);
            },
            onStompError: (frame) => {
                console.error('❌ WebSocket error:', frame);
                updateConnectionStatus(false);
            }
        });

        stompClient.activate();

    } catch (error) {
        console.error('❌ WebSocket connection error:', error);
        updateConnectionStatus(false);
    }
}
```

Js

```js
// Search for participants
async function searchParticipants(query) {
    try {
        const token = localStorage.getItem('jwt');
        const response = await fetch(`${API_BASE}/api/auth/users/search?query=${encodeURIComponent(query)}`, {
            headers: { 'Authorization': `Bearer ${token}` }
        });

        if (response.ok) {
            const users = await response.json();
            displayParticipantSearchResults(users);
        }
    } catch (error) {
        console.error('❌ Error searching participants:', error);
    }
}
```

Quiz:

Students can give practice quiz and set timer and number of questions by their own

```
<body>
  <!-- Top Navigation Bar -->
  <div class="top-nav">
    <div class="nav-brand">
      <img src="../css/tutorconnect-logo.png" alt="TutorConnect Logo" class="course-logo">
      <h3>TutorConnect</h3>
    </div>
    <button onclick="window.location.href='student_dashboard.html'" class="logout-btn">🏠 Dashboard</button>
  </div>

  <div class="main-container">
    <div class="quiz-container">
      <div class="quiz-header">
        <h2>🧠 Practice Quiz</h2>
        <p>Test your knowledge and improve your skills</p>
      </div>

      <div id="quizSettings" class="settings-modal">
        <div class="form-group">
          <label for="numQuestions">📊 Number of Questions:</label>
          <input type="number" id="numQuestions" min="1" value="3" placeholder="Enter number of questions">
        </div>
        <div class="form-group">
          <label for="timeLimit">⏰ Time Limit (minutes):</label>
          <input type="number" id="timeLimit" min="1" value="10" placeholder="Enter time limit">
        </div>
        <button id="startQuizBtn" class="quiz-btn">🚀 Start Quiz</button>
        <div id="settingsError"></div>
      </div>

      <div id="quizArea" style="display:none;">
        <!-- Quiz questions will be dynamically inserted here -->
      </div>

      <div id="resultArea" style="display:none;">
        <!-- Quiz results will be displayed here -->
      </div>
```

Html

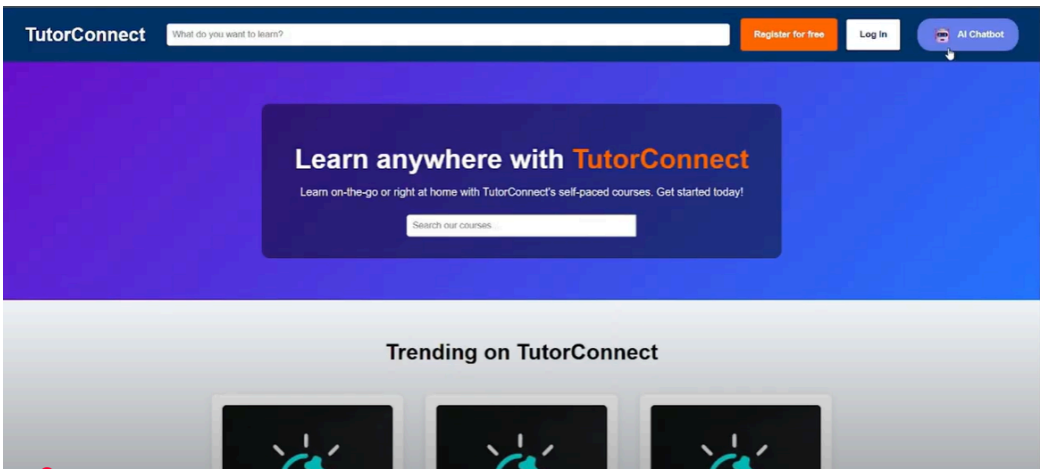
Js

```css
.quiz-btn {
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
  color: white;
  border: none;
  border-radius: 15px;
  padding: 18px 40px;
  font-size: 18px;
  font-weight: 700;
  cursor: pointer;
  transition: all 0.3s ease;
  box-shadow: 0 6px 20px rgba(0, 0, 0, 0.2);
  display: block;
  margin: 30px auto 0;
}

.quiz-btn:hover {
  transform: translateY(-3px);
  box-shadow: 0 8px 25px rgba(0, 0, 0, 0.3);
}

.question-card {
  background: rgba(255, 255, 255, 0.95);
  border-radius: 15px;
  padding: 25px;
  margin: 20px 0;
  border: 1px solid rgba(255, 255, 255, 0.3);
  color: #333;
}

.question-text {
  color: #333;
  font-size: 1.3rem;
  margin-bottom: 25px;
  font-weight: 600;
  line-height: 1.5;
```

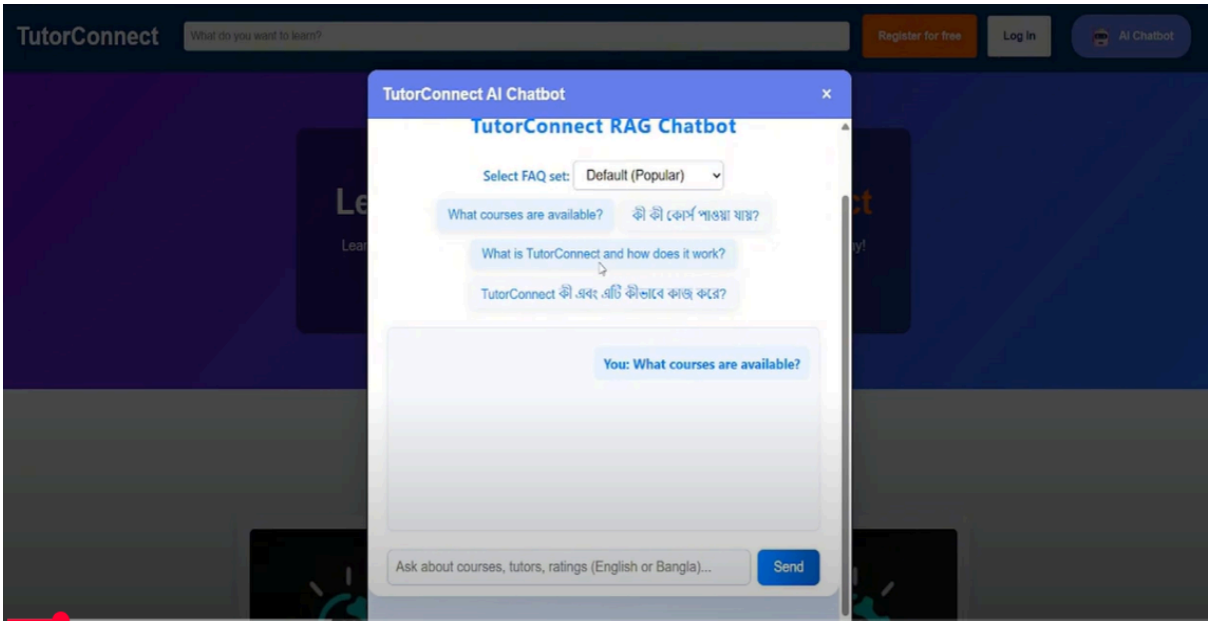
```javascript
function startQuiz(questions, timeLimit) {
  settingsDiv.style.display = 'none';
  quizArea.style.display = '';
  resultArea.style.display = 'none';
  let timer = timeLimit * 60;
  let interval;
  let answers = Array(questions.length).fill(null);
  quizArea.innerHTML = `<div id="timer" style="font-weight:bold;margin-bottom:10px;"></div>`;
  questions.forEach((q, idx) => {
    quizArea.innerHTML += `
      <div class="quiz-question" style="margin-bottom:20px;">
        <b>Q${idx+1}:</b> ${q.question}<br>
        ${q.options.map((opt, oidx) => `<label><input type="radio" name="q${idx}" value="${opt}"> ${opt}</label><br>`).join('')}
      </div>
    `;
  });
  quizArea.innerHTML += `<button id="submitQuizBtn">Submit Quiz</button>`;
  // Timer logic
  function updateTimer() {
    let min = Math.floor(timer/60);
    let sec = timer%60;
    document.getElementById('timer').textContent = `Time left: ${min}:${sec.toString().padStart(2,'0')}`;
    if (timer <= 0) {
      clearInterval(interval);
      submitQuiz();
    }
    timer--;
  }
  interval = setInterval(updateTimer, 1000);
  updateTimer();
  document.getElementById('submitQuizBtn').onclick = function() {
    clearInterval(interval);
    submitQuiz();
  };
  function submitQuiz() {
    quizArea.style.display = 'none';
```

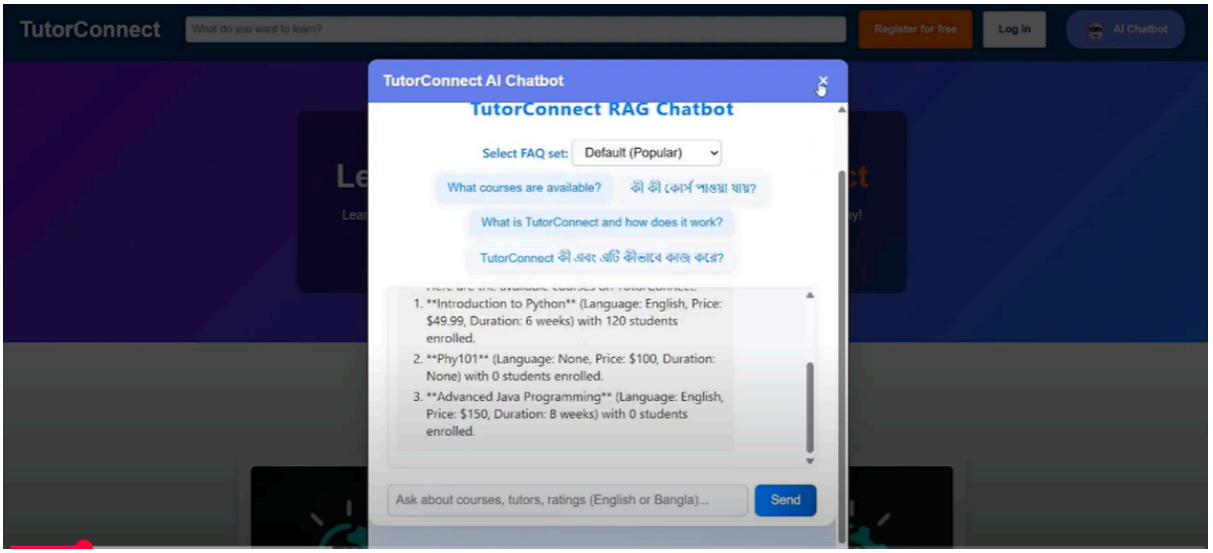
7. User Manual

The homepage or landing page, where the user can choose either to log in or register

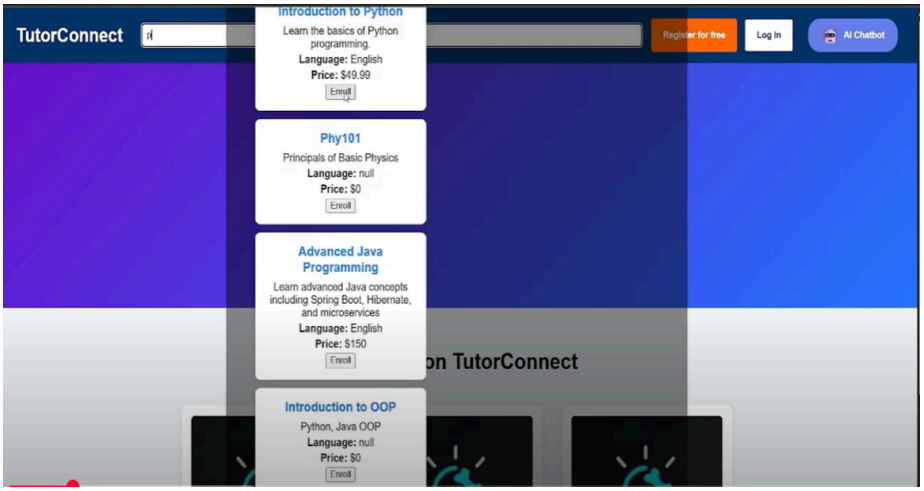


From AI chatbot, visitors can get course related information.

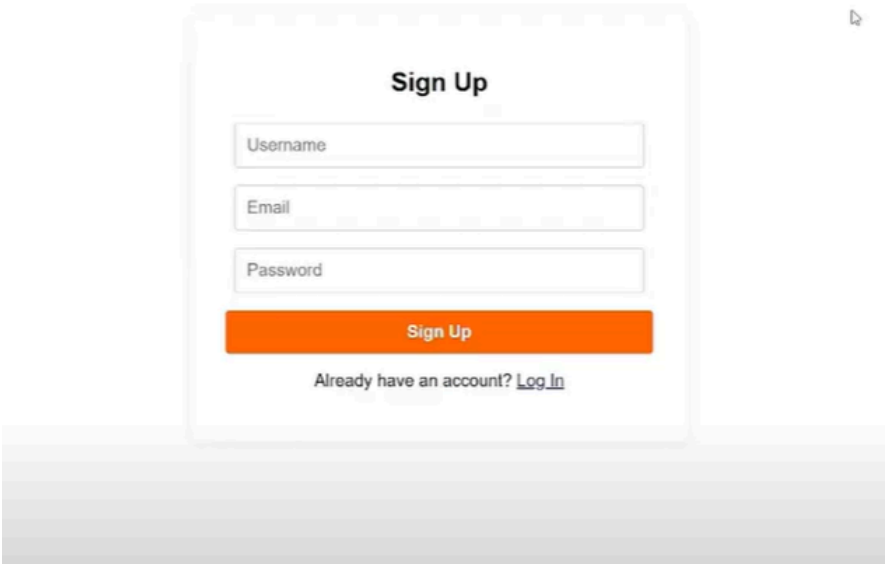


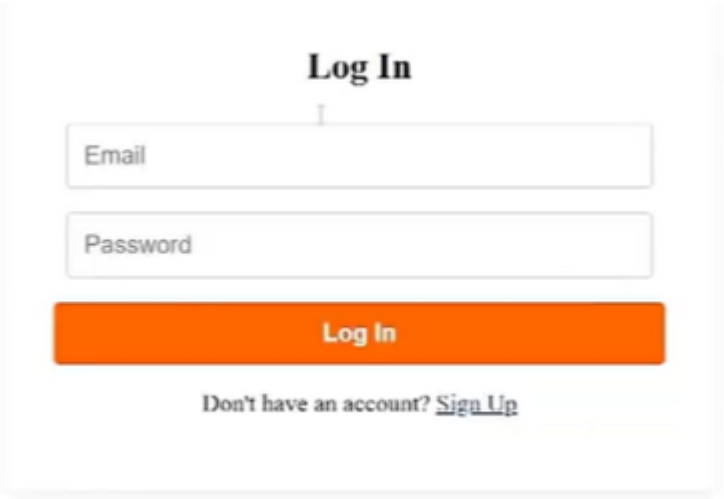


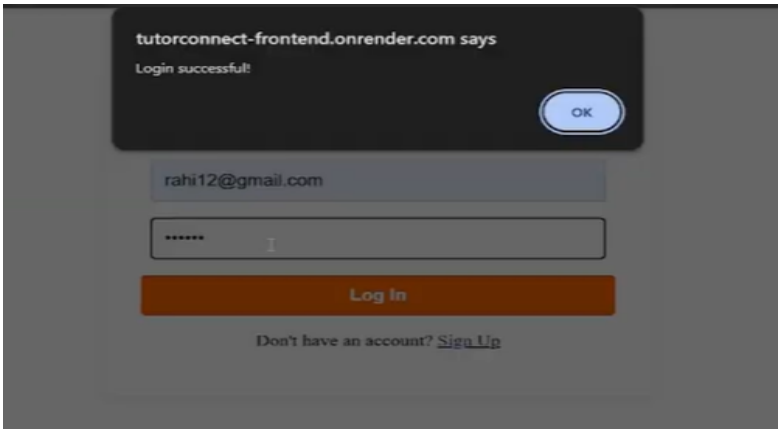
Visitors can search anything about course from search option

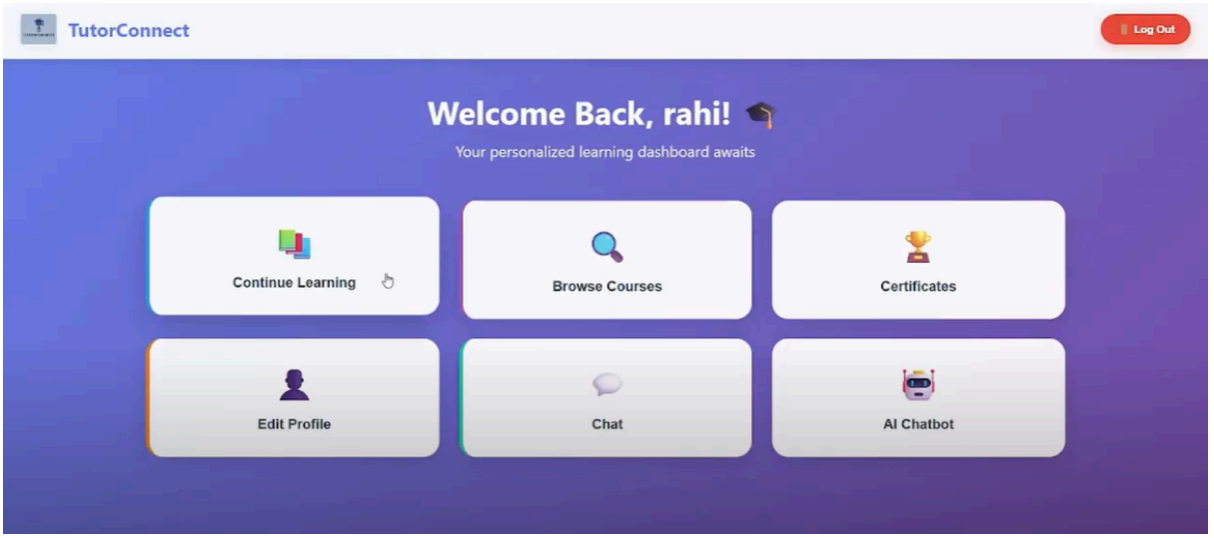


Tutors, Student and Admin can sign up and log in and back to their homepage and can view dashboard and other staffs

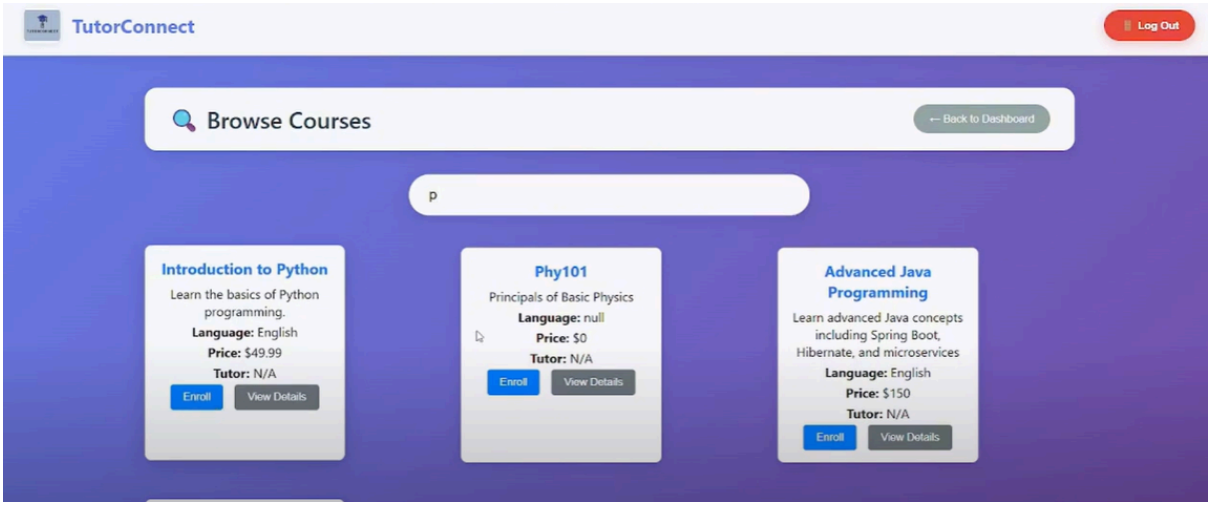




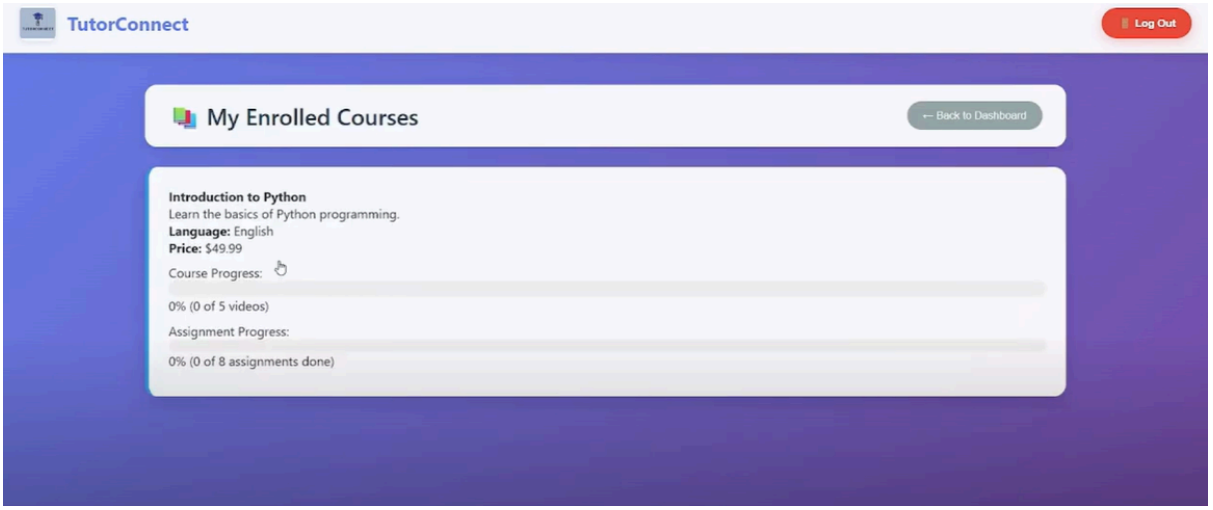




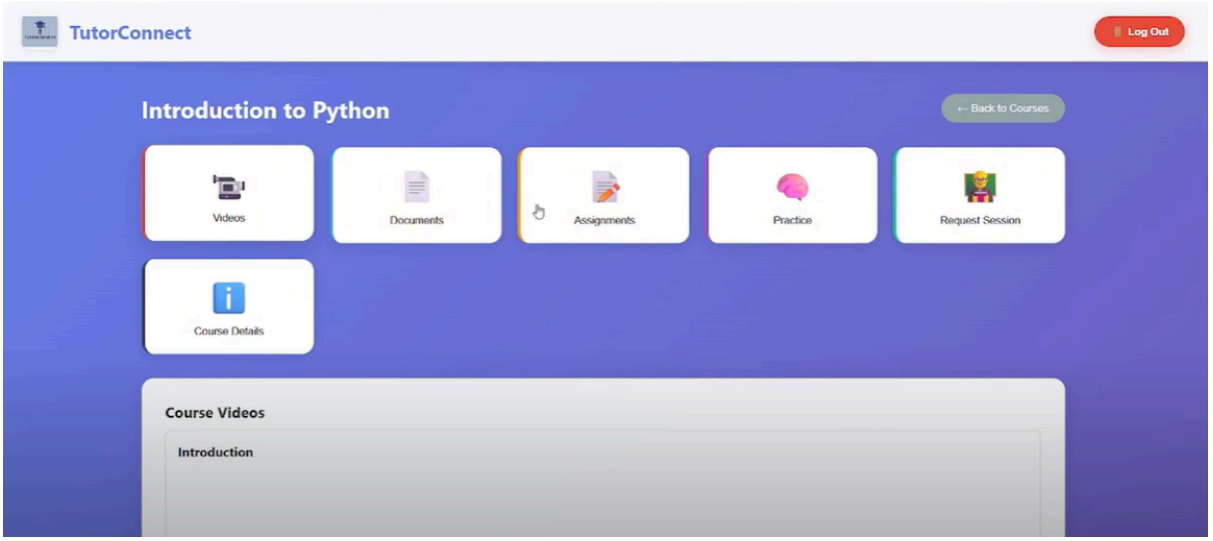
Students can enroll course to continue their learning

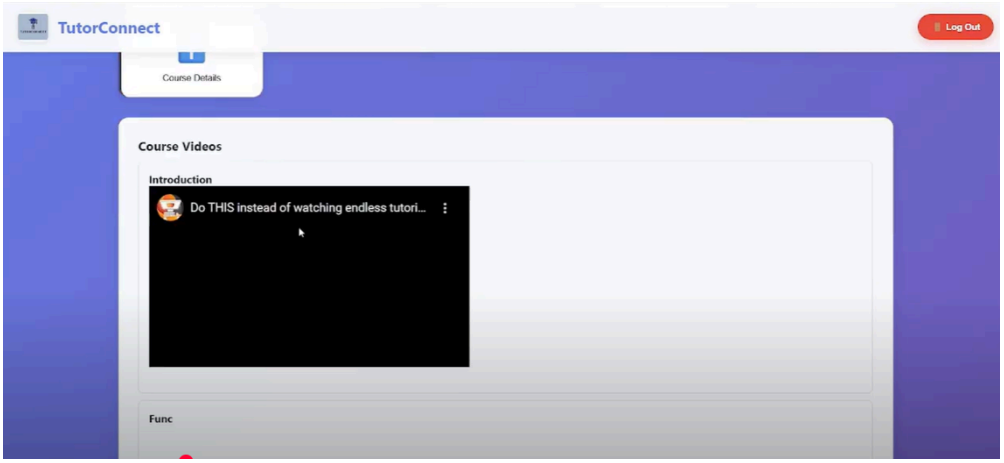


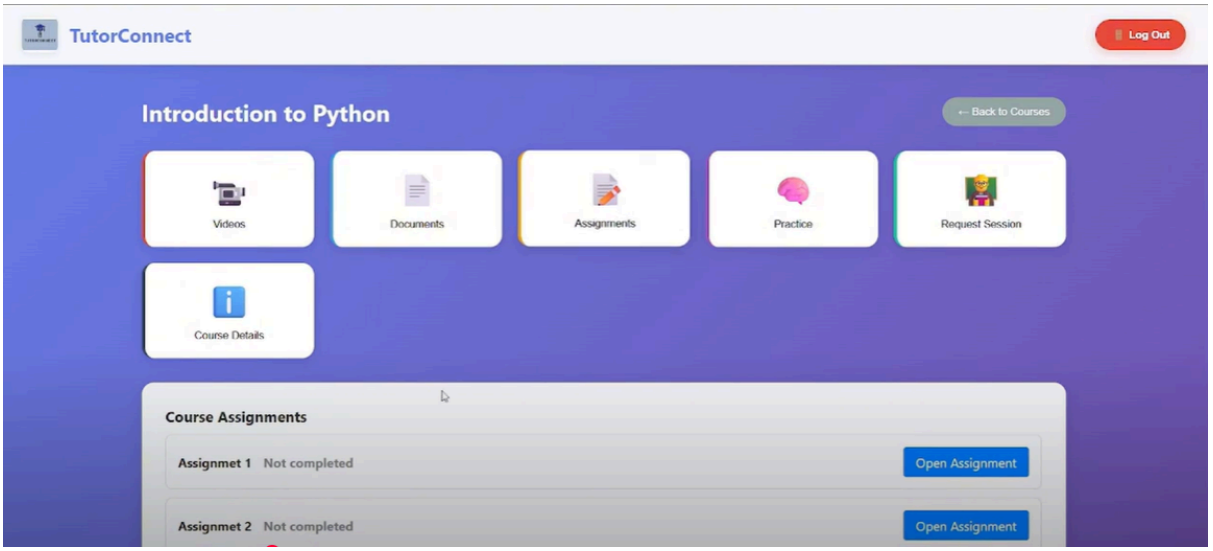
After enrolling, student can view their enrolled course and view their progress

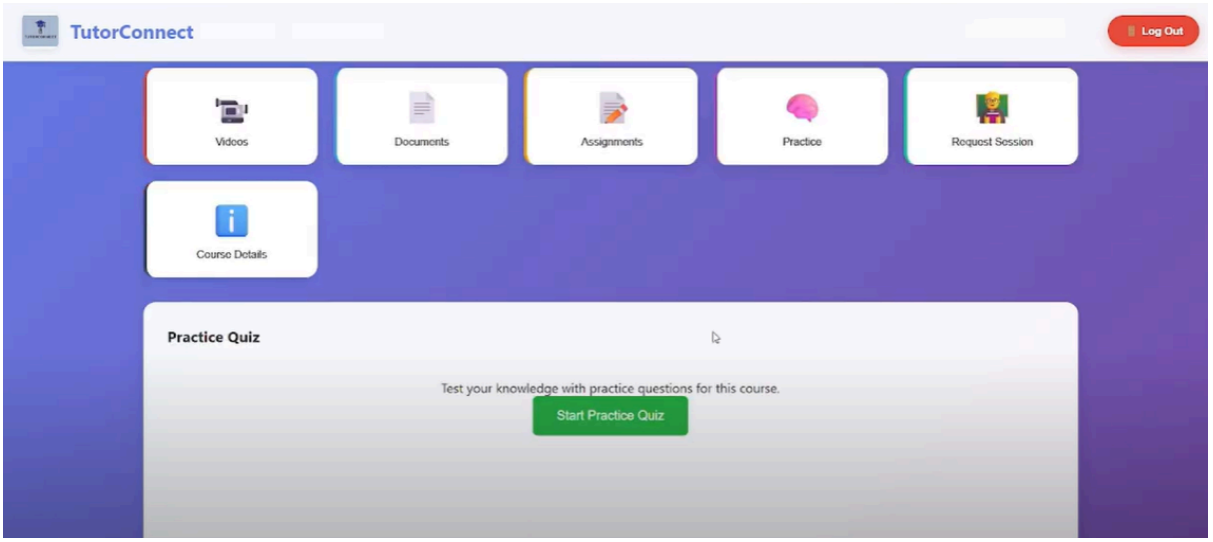


Students can view all the resources from the enrolled course including quiz,assignments, videos etc

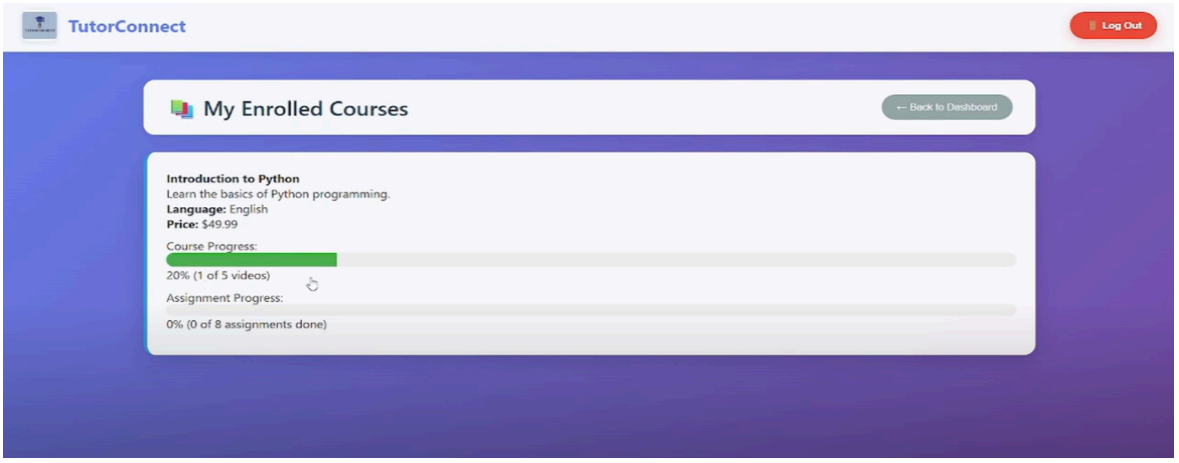




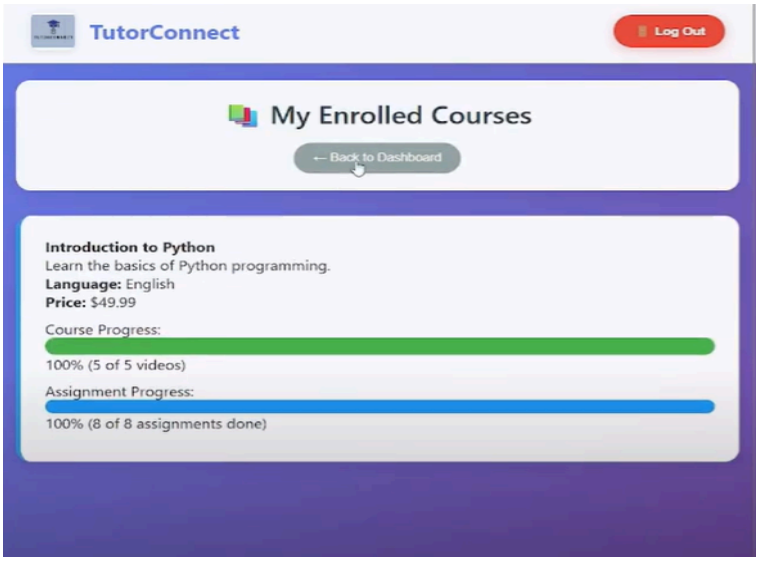


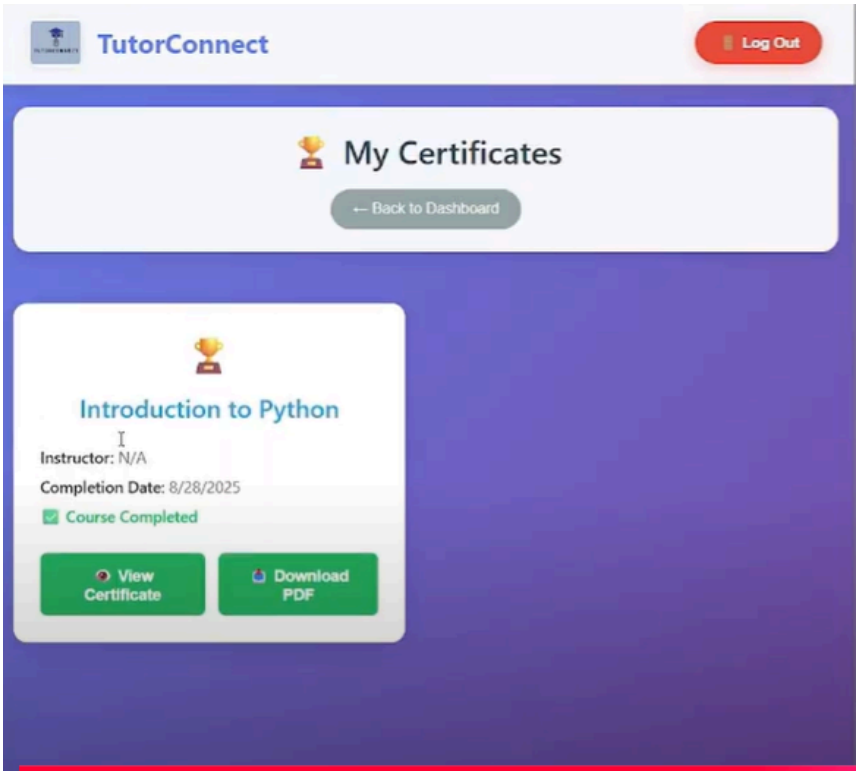


If student complete watching video, the course is shown increased



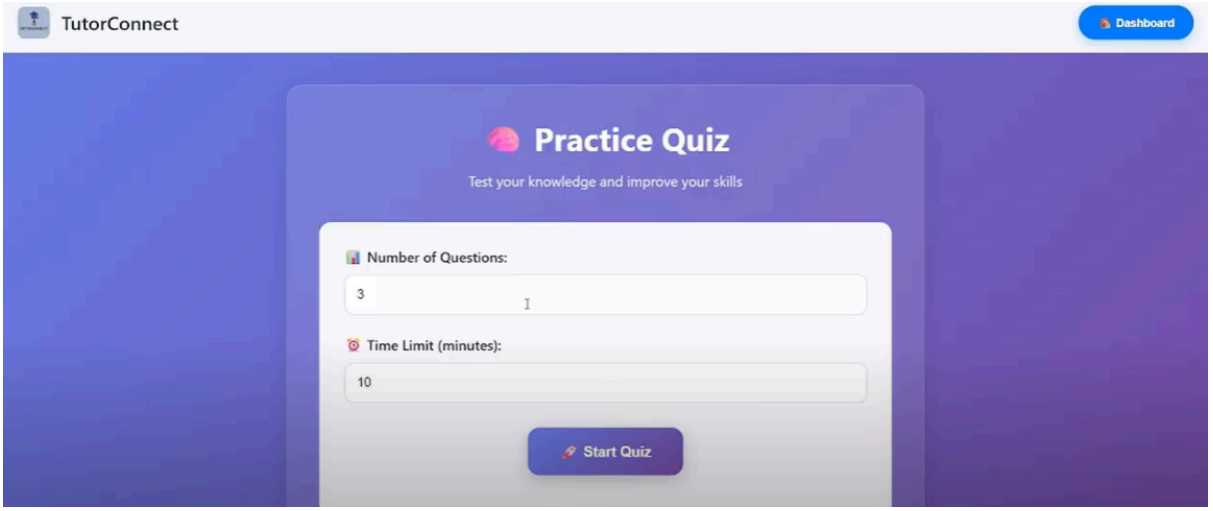
Student can get his certificates by completing all assessments

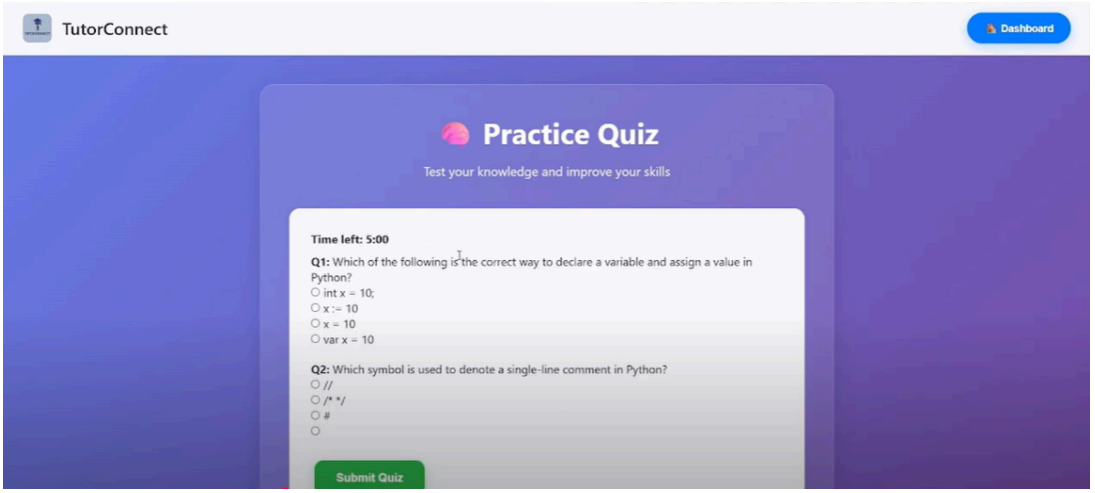



TutorConnect
Log Out
My Certificates
← Back to Dashboard
Introduction to Python
Instructor: N/A
Completion Date: 8/28/2025
Course Completed
View Certificate
Download PDF

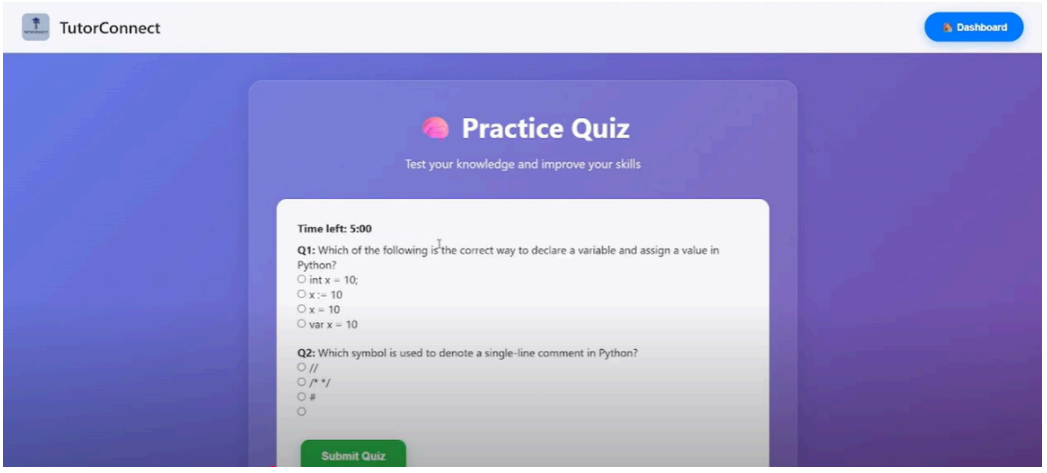

CERTIFICATE OF COMPLETION
TutorConnect
This is to certify that
rahi
has successfully completed the course
"Introduction to Python"
under the guidance of
Instructor: N/A
Completed on: August 28, 2025

Student can start practice quiz by their choosable time frame and number of questions

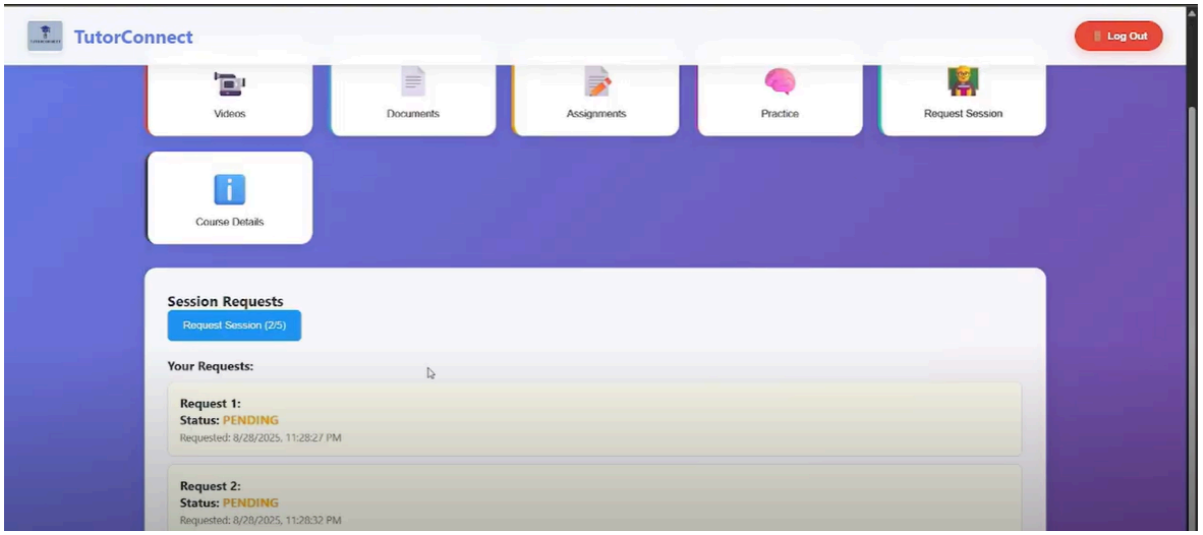




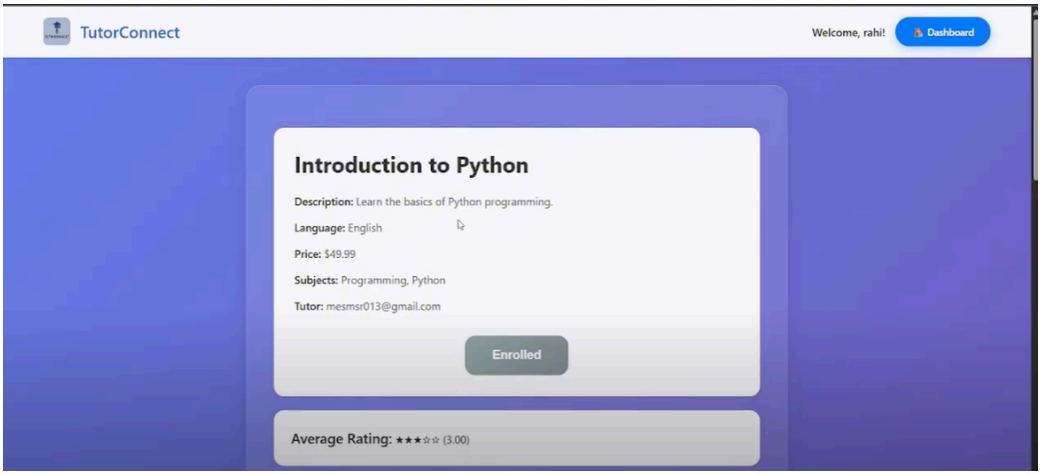
After submitting quiz, student can view their result

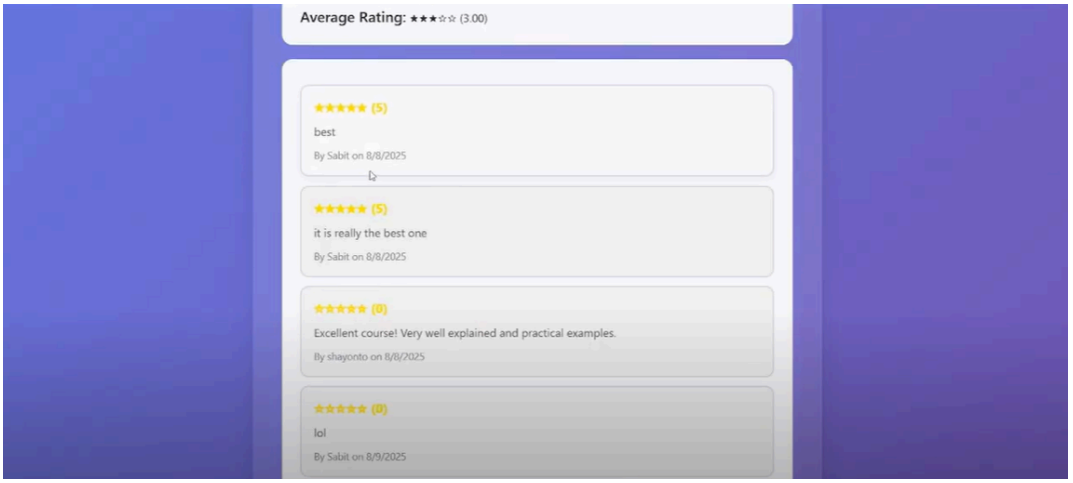


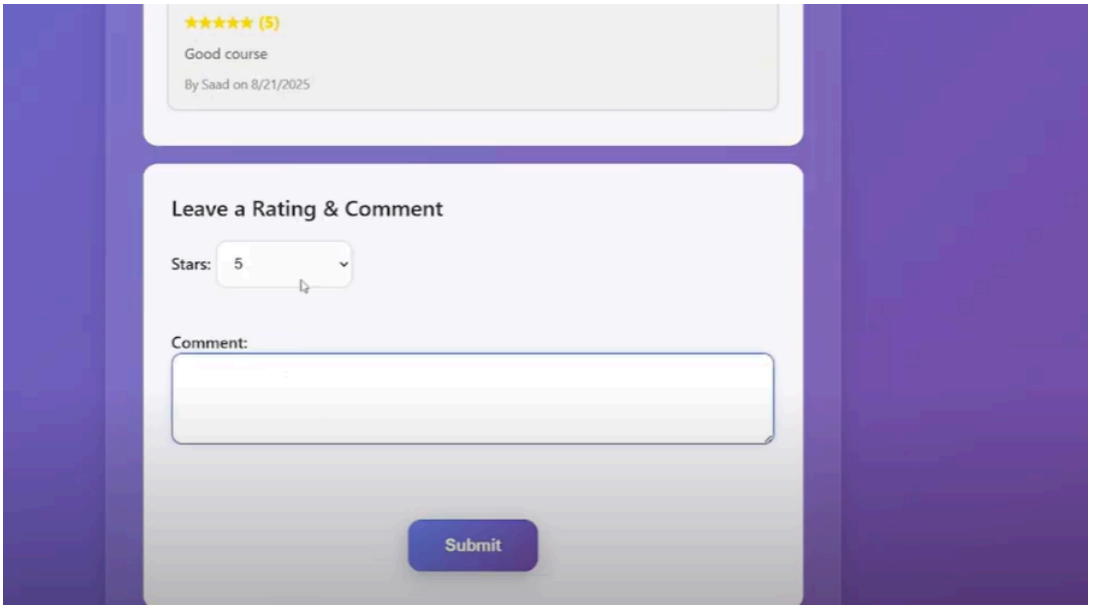
Students can take session requests from tutor



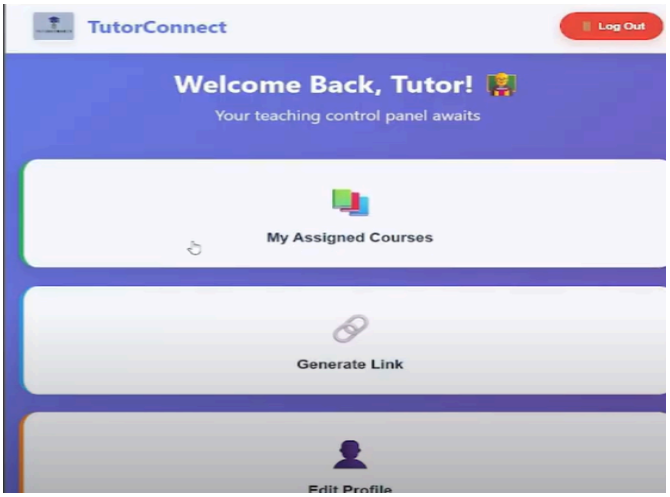
Student can view course details, ratings and reviews and also give feedback

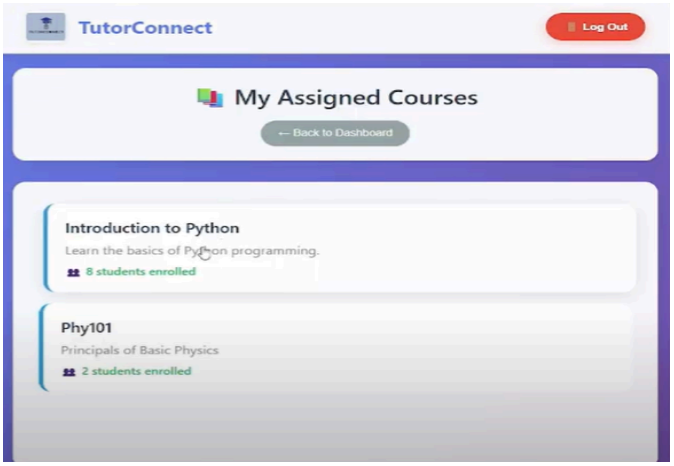




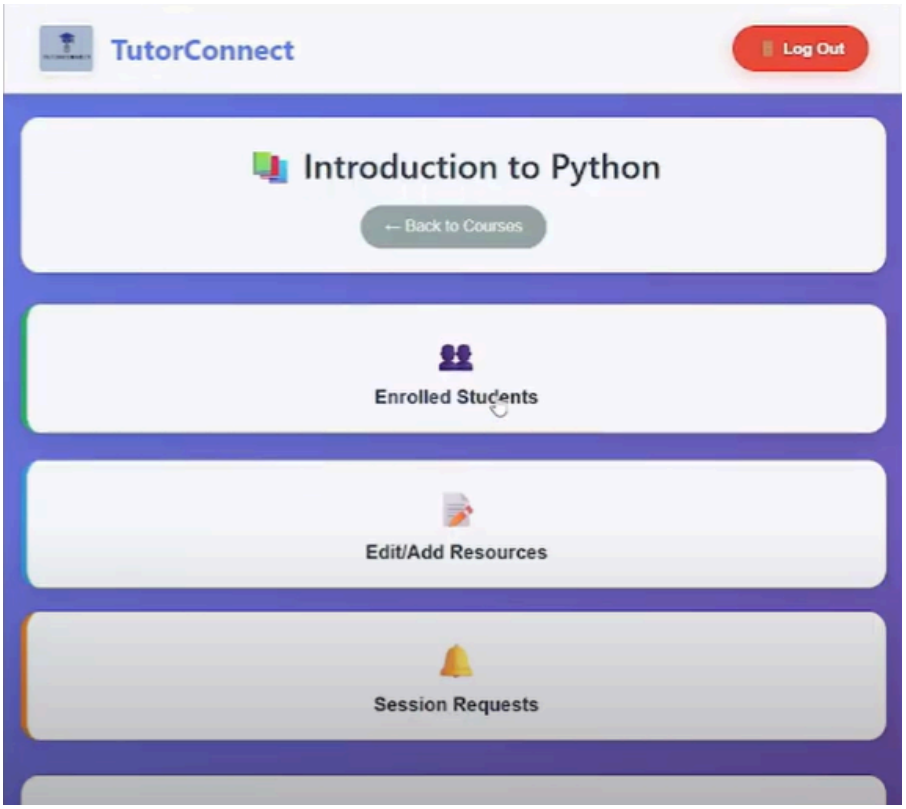


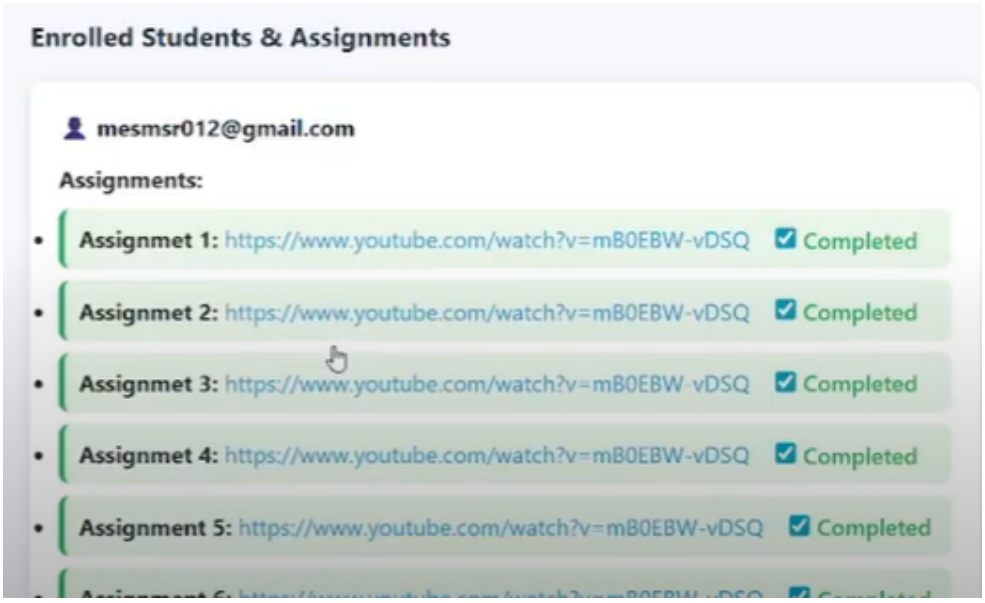
When a teacher log in his account, he will able to view his dashboard



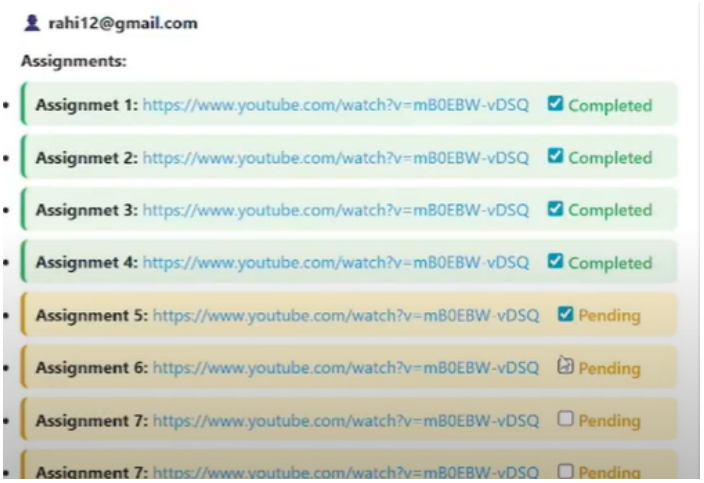


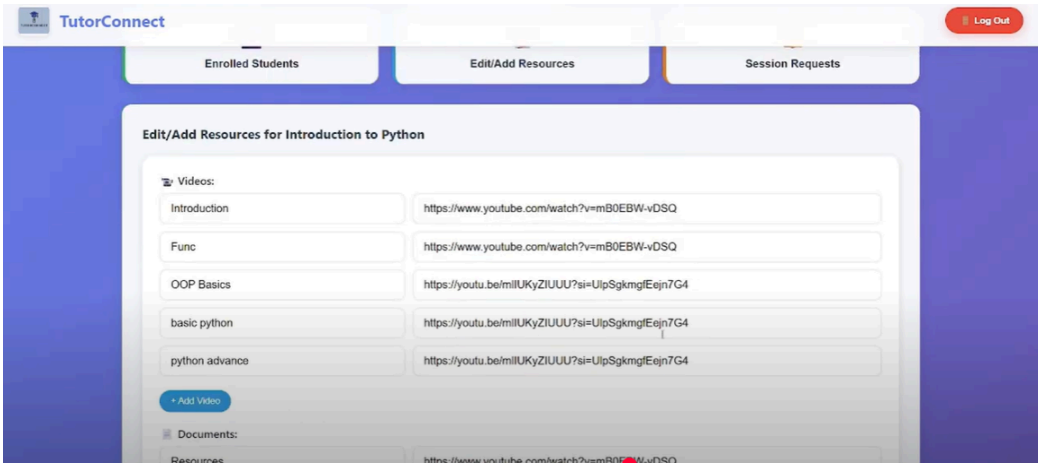
Tutor can view his courses, student list, assign resources and confirm or reject the session from students

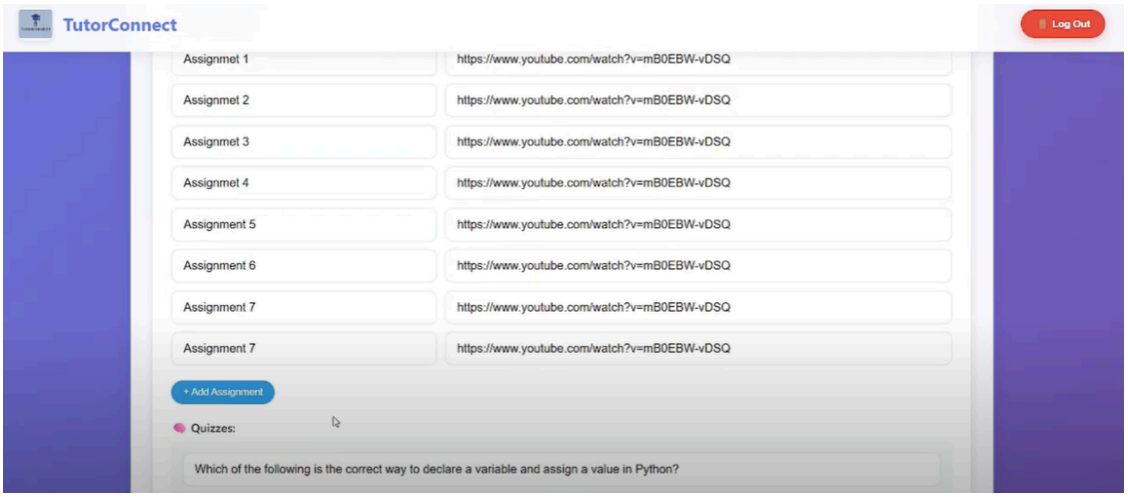




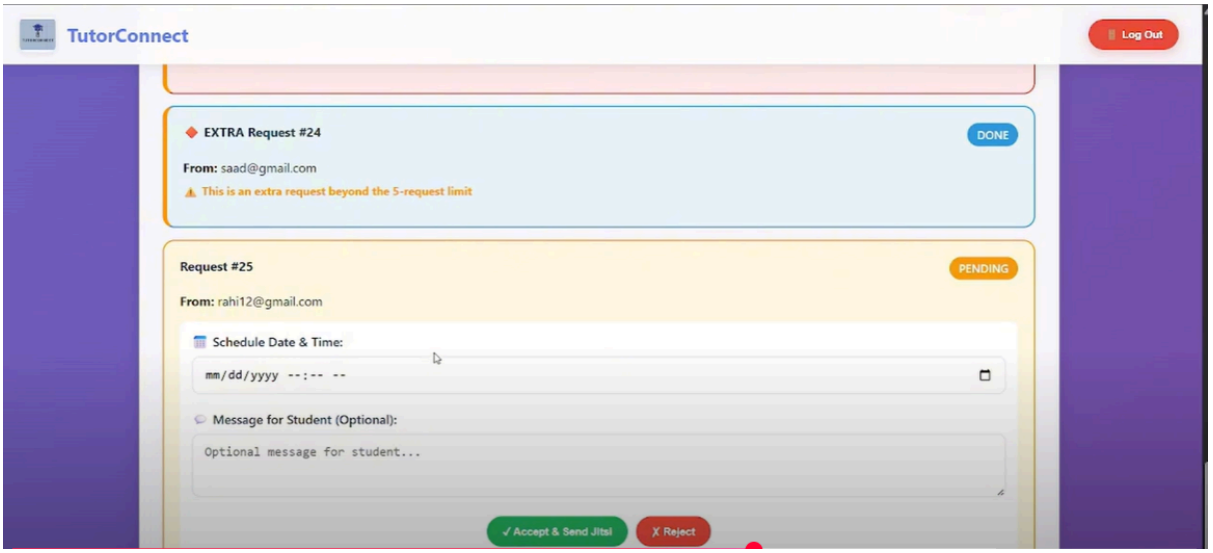
By clicking right, the tutor can approve the student's assignment.

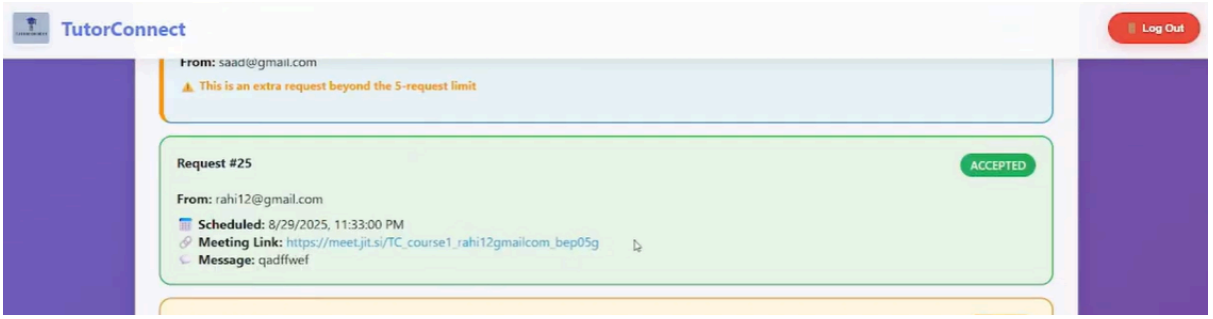




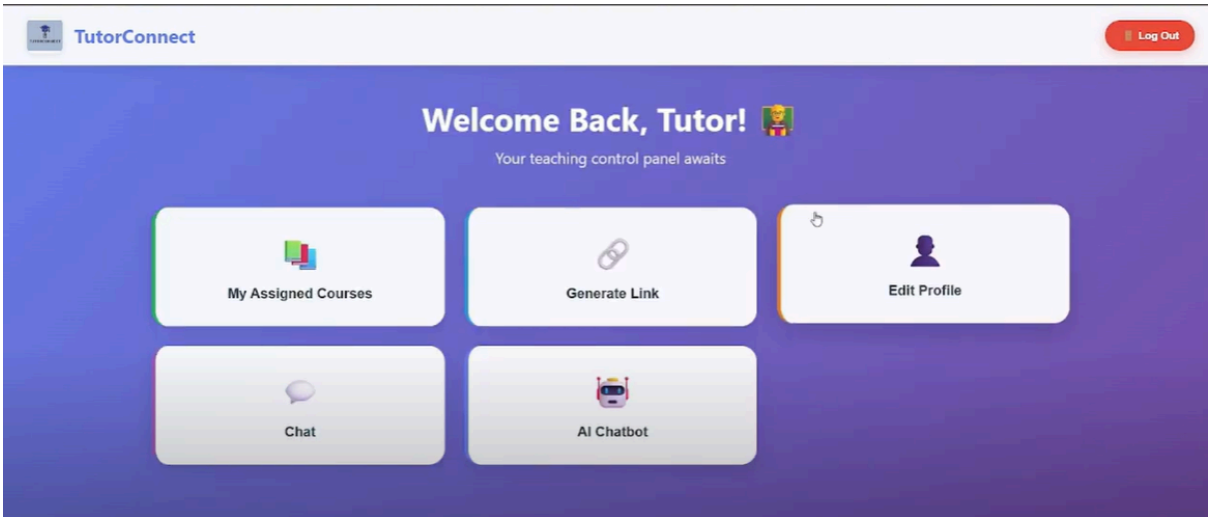


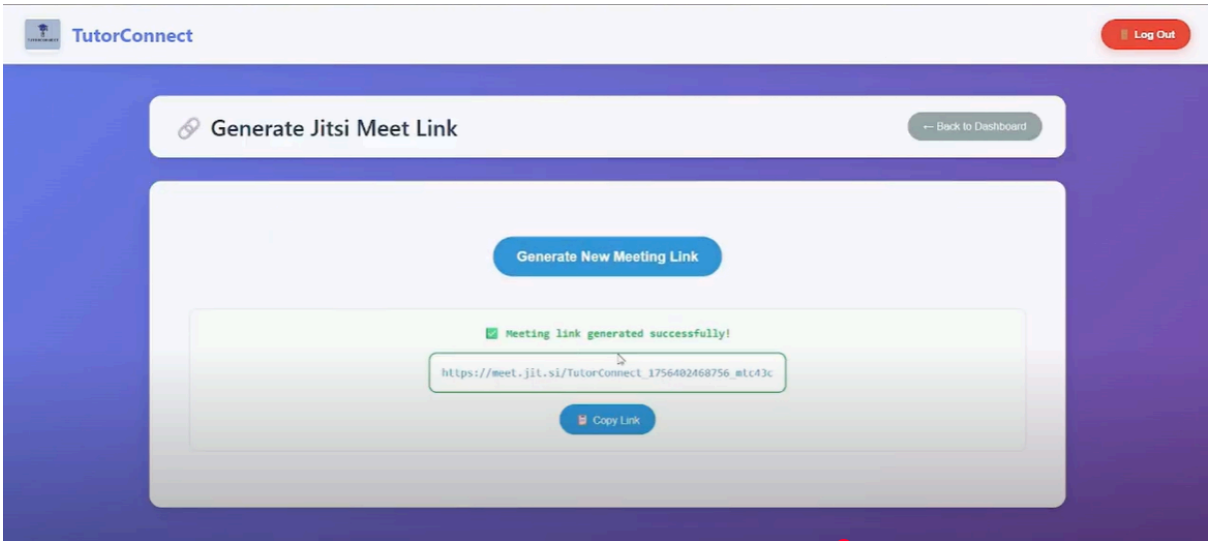
Tutor can confirm and reject Student's session request



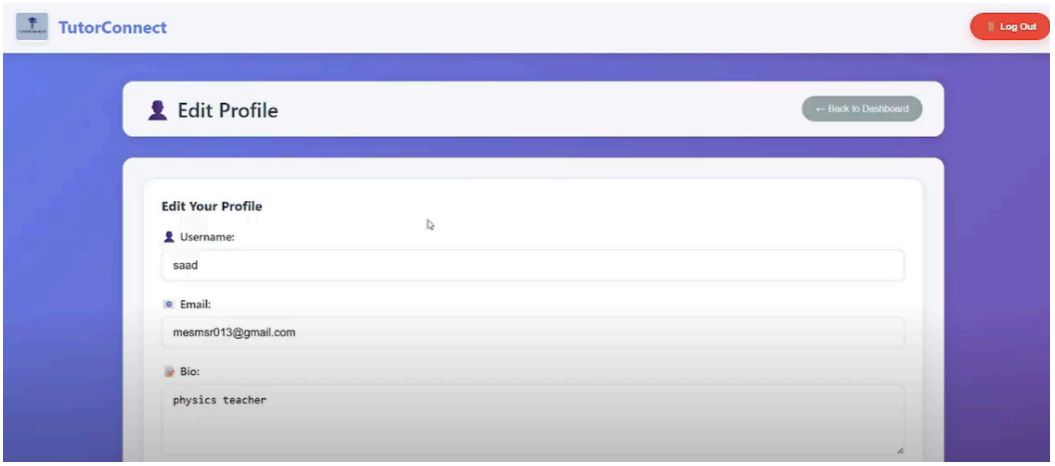


Tutor can generate meet link for the students

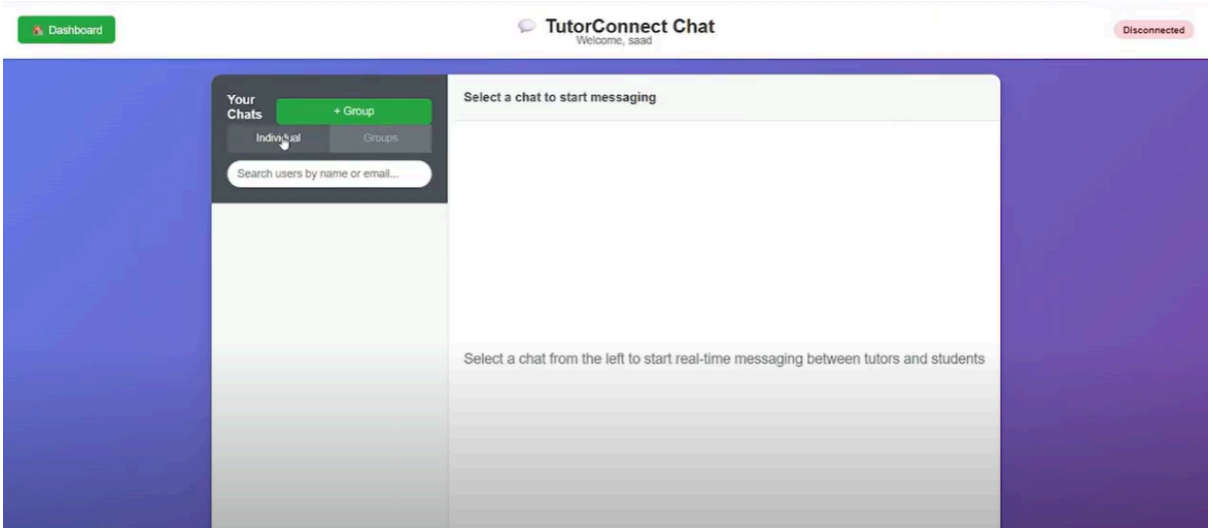


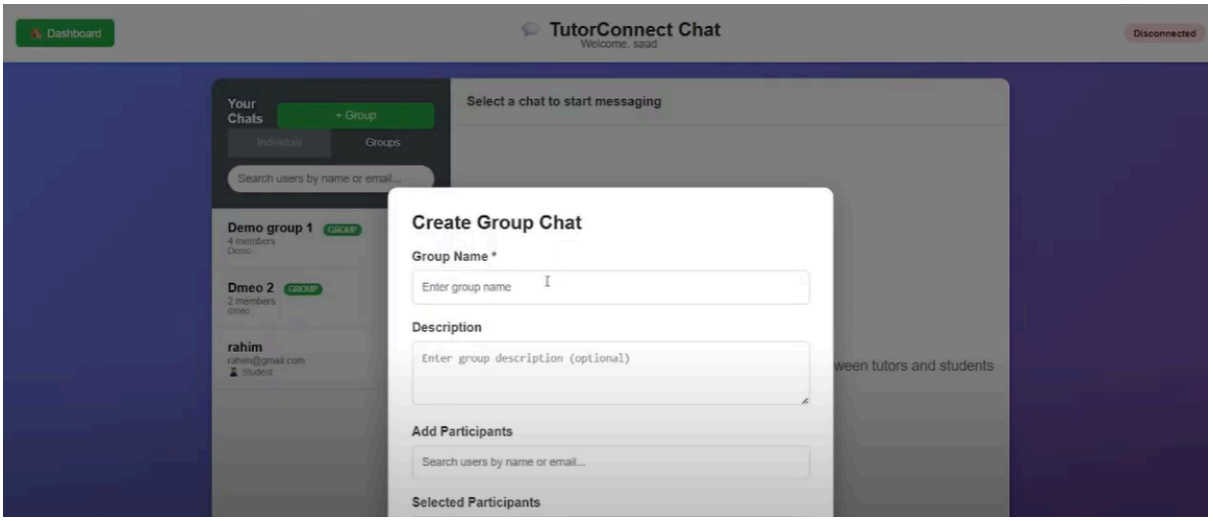


Teacher and student can edit his profile

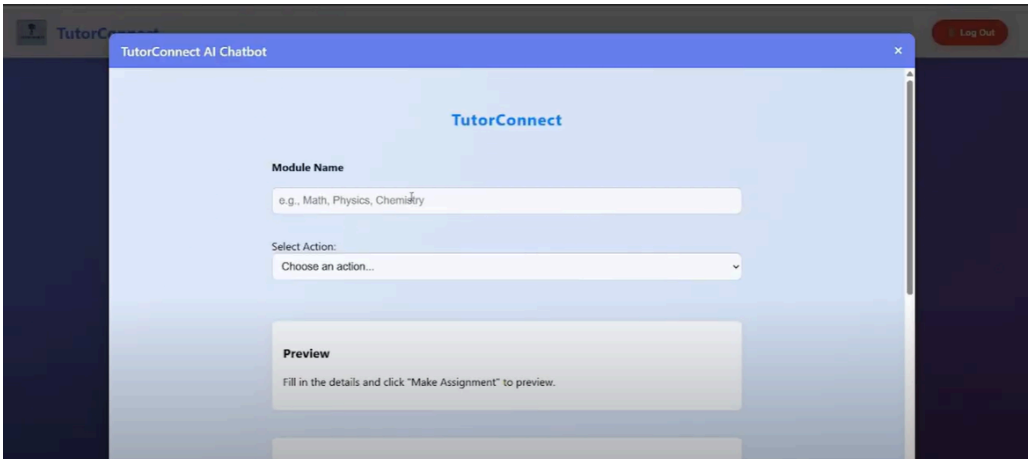


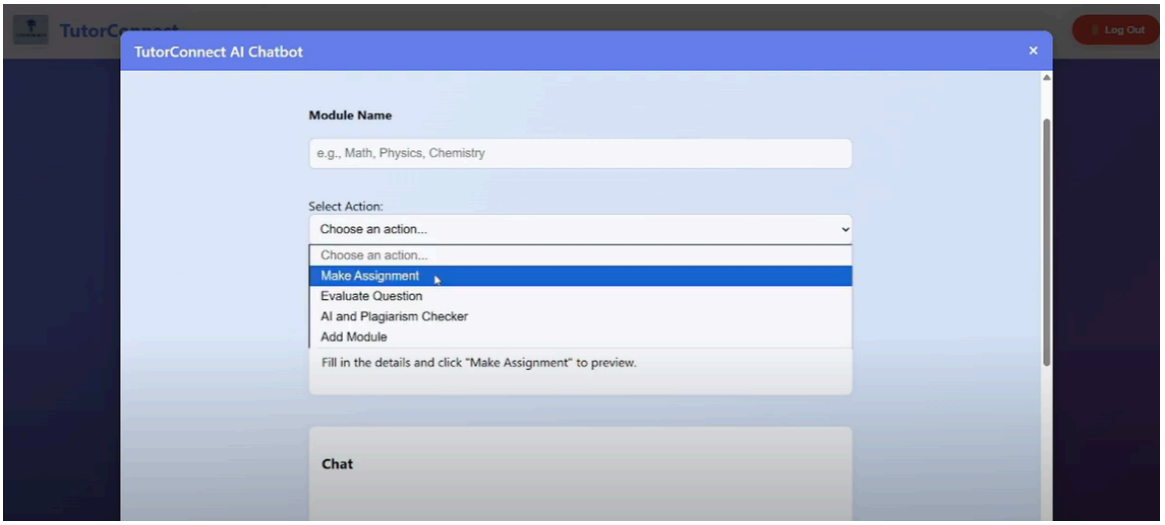
Student can do chat with tutor and also make group chat

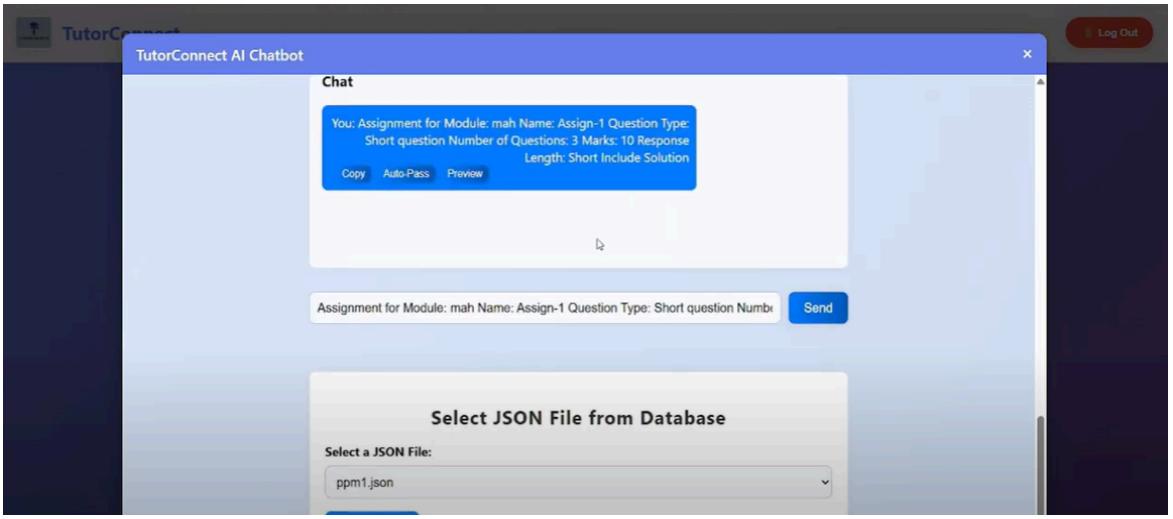


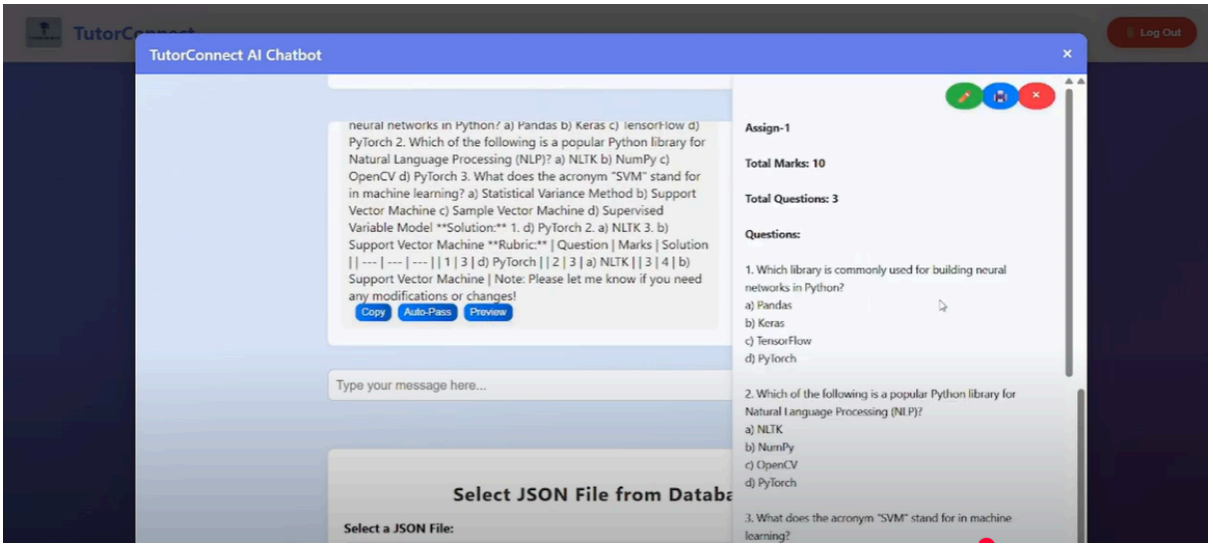


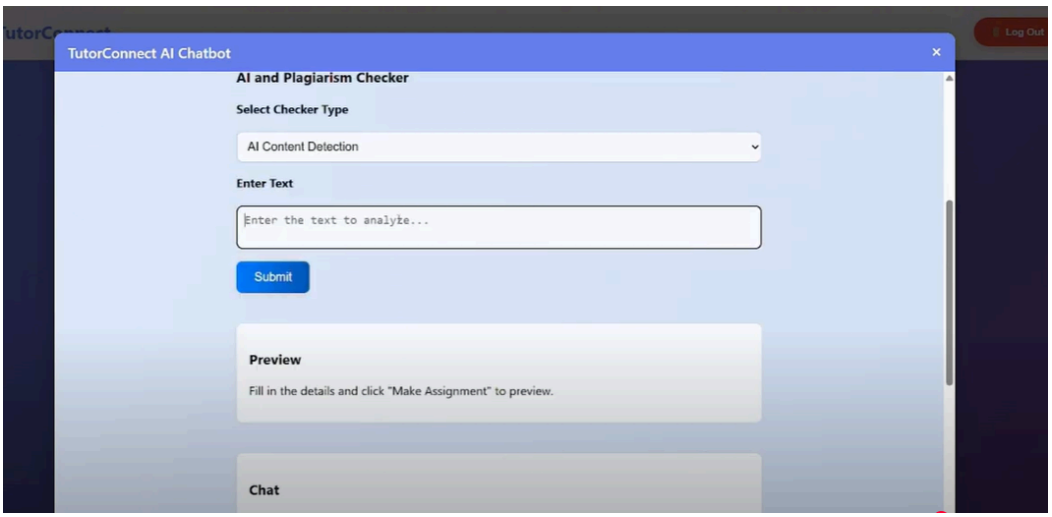
Tutor can make assignments, quiz and other resources and can check AI and Plagiarism percentage with the help of AI Chatbot



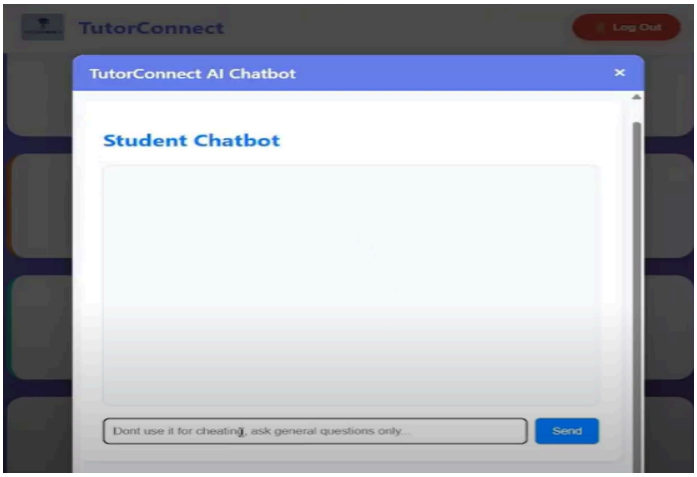



TutorConnect AI Chatbot
Chat
You: Assignment for Module: mah Name: Assign-1 Question Type: Short question Number of Questions: 3 Marks: 10 Response Length: Short Include Solution
Copy
Auto-Pass
Preview
Assignment for Module: mah Name: Assign-1 Question Type: Short question Numbe
Send
Select JSON File from Database
Select a JSON File:
ppm1.json

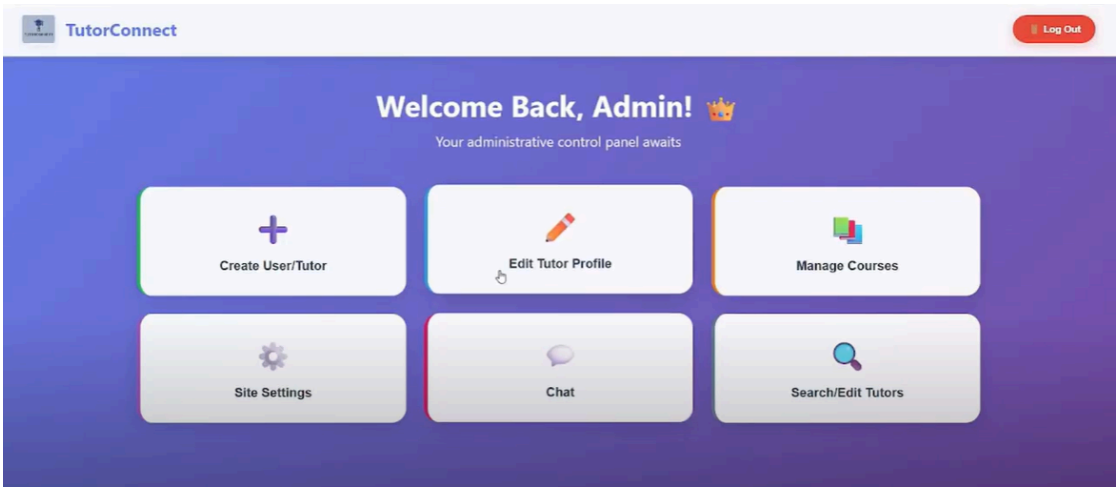

TutorConnect AI Chatbot
Copy
Auto-Pass
Preview
Type your message here...
Select JSON File from Databa
Select a JSON File:
Assign-1
Total Marks: 10
Total Questions: 3
Questions:
1. Which library is commonly used for building neural networks in Python?
a) Pandas
b) Keras
c) TensorFlow
d) Pytorch
2. Which of the following is a popular Python library for Natural Language Processing (NLP)?
a) NLTK
b) NumPy
c) OpenCV
d) PyTorch
3. What does the acronym "SVM" stand for in machine learning?


TutorConnect AI Chatbot
AI and Plagiarism Checker
Select Checker Type
AI Content Detection
Enter Text
Enter the text to analyze...
Submit
Preview
Fill in the details and click "Make Assignment" to preview.
Chat

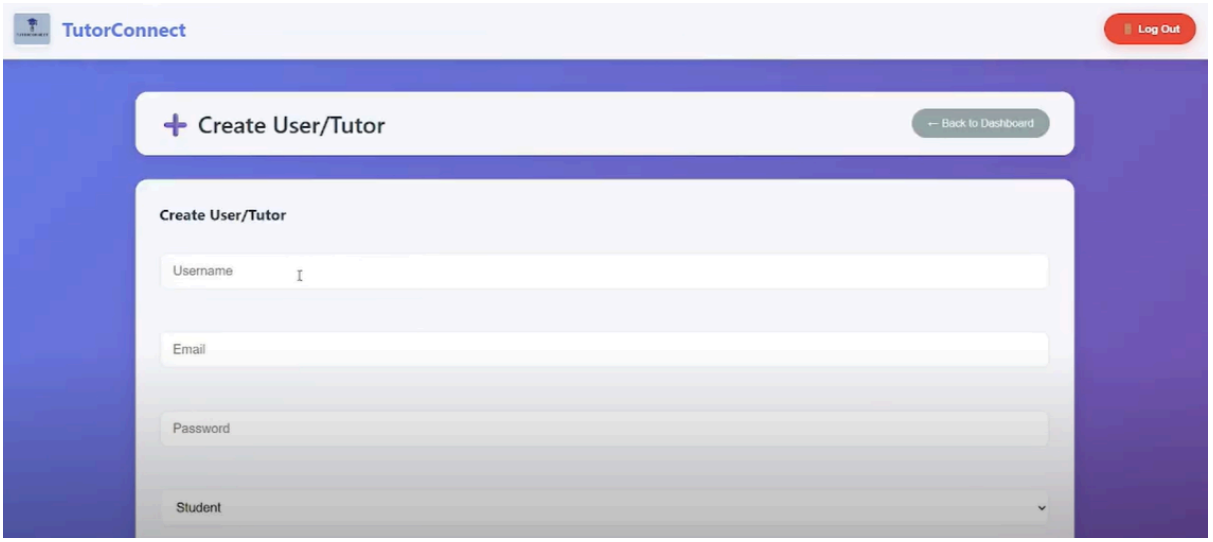
Student can get any course related information with the help of AI Chatbot

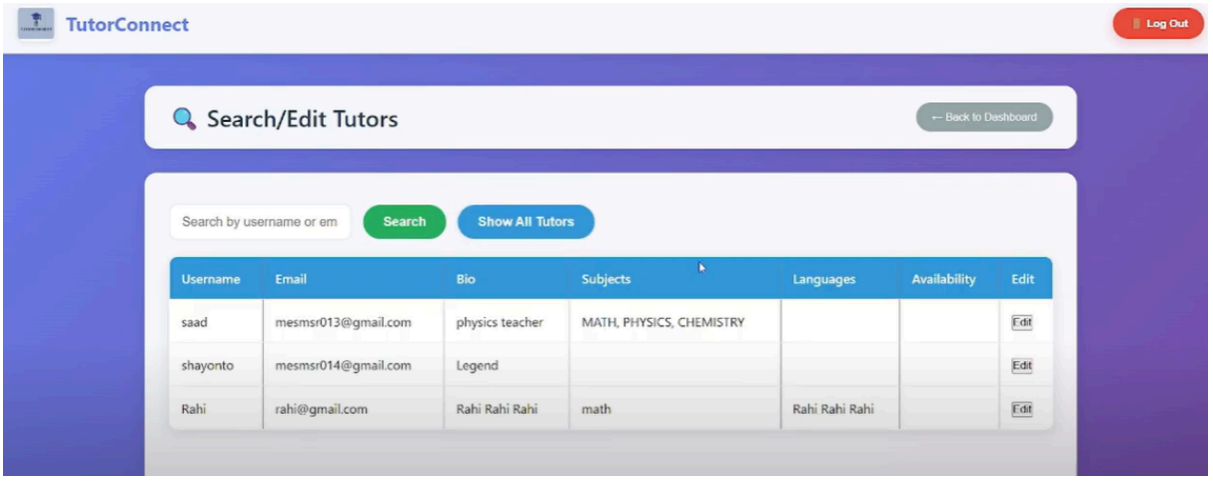


When Admin login their page, he will view his dashboard

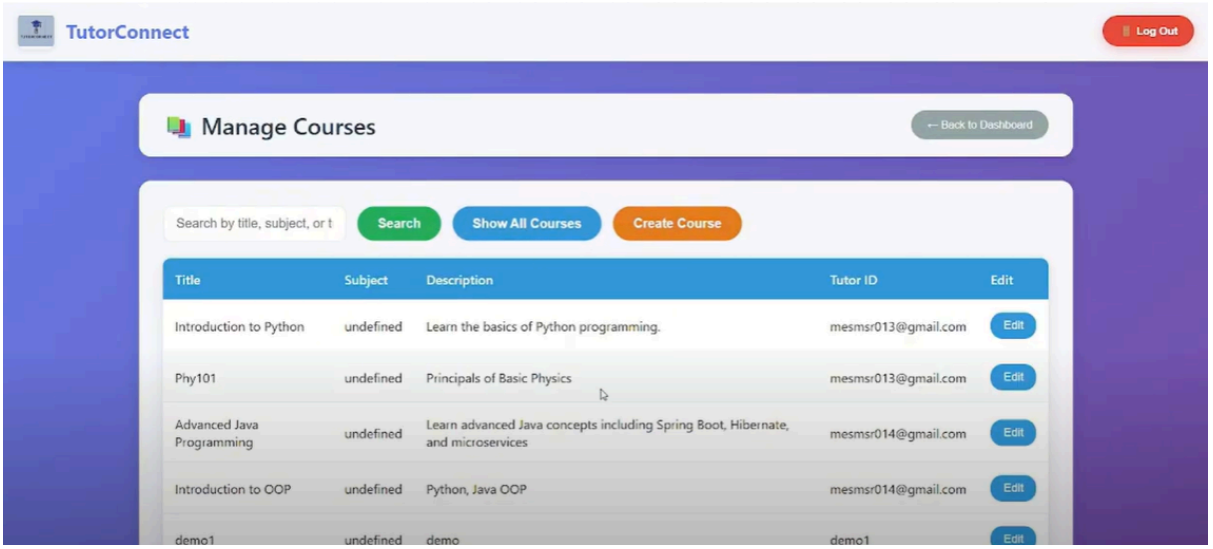


Admin can edit tutor and edit other necessary information of Students and Tutor



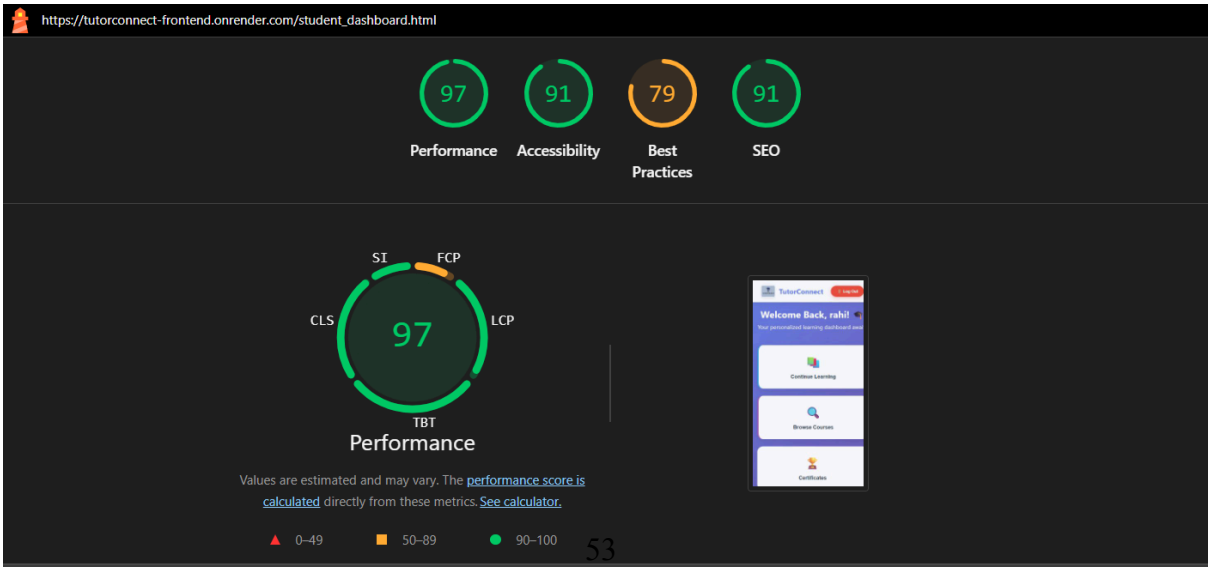


Admin can also manage course

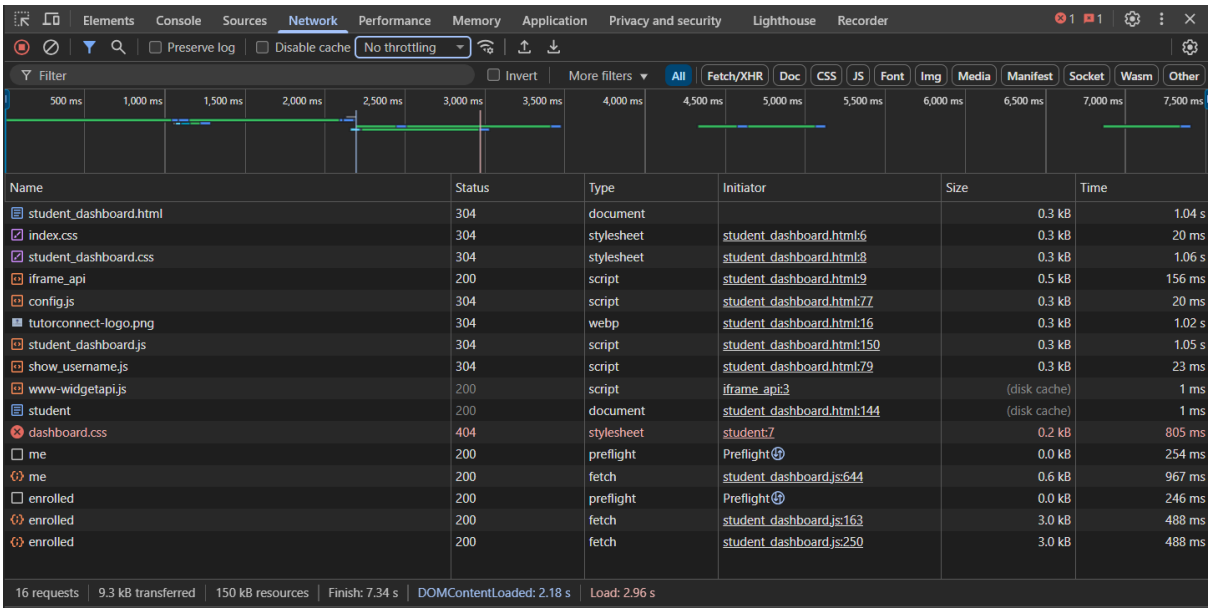


8. Performance and Network Analysis

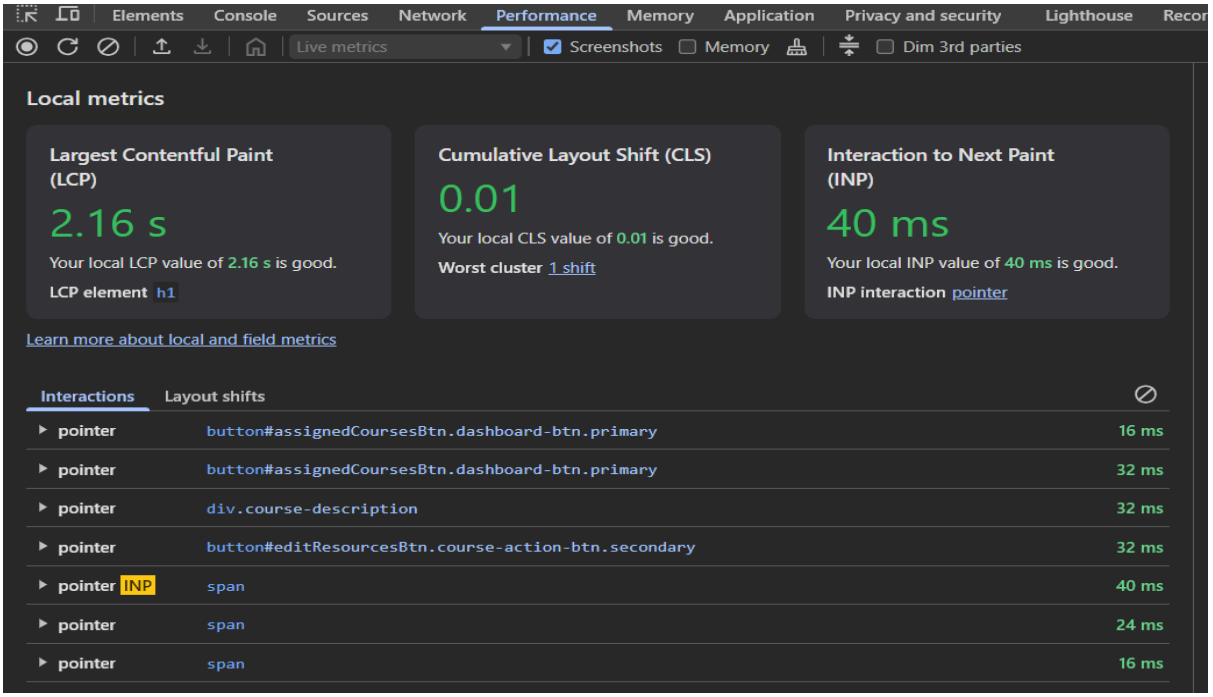
Lighthouse :



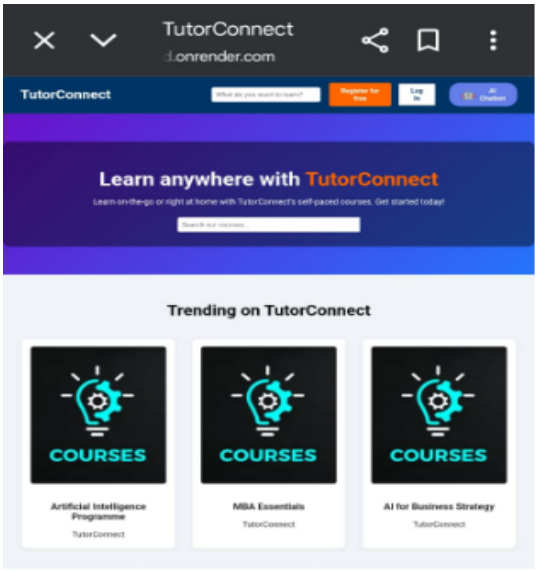
Network Analysis:



Performance:



Mobile viewport:



9. GitHub Repo [Public] Link

https://github.com/iamsmsr/TutorConnect-Personalized-online-tutoring-and-course-delivery-system

10. Link of Deployed Project:

https://tutorconnect-frontend.onrender.com/

11. Individual Contribution

Group member - 01	
Name: S.M. Saidur Rahman	Student ID: 22101451
Functional Requirements that are developed by this member:	
1. Admins manage user roles and control access.	
2. Students book live tutoring sessions; tutors approve or reject requests.	
3. AI-powered FAQ answers platform-related queries using internal data.	
4. Tutors create assignments and questions (MCQ, short, essay) from the RAG course database.	

Group member - 02	
Name: Shayonto Rayhan	Student ID: 22101651
Functional Requirements that are developed by this member:	
1. Secure sign-up and login for students, tutors, and admins.	
2. Interactive tools include assignments, forums, and quizzes for engagement.	
3. Bilingual AI chatbot assists with academic questions in Bangla and English for registered and unregistered users.	
4. AI & plagiarism detector scans student submissions and generates highlighted reports..	

Group member - 03	
Name: MD.Niamatullah	Student ID: 22101439
Functional Requirements that are developed by this member:	
1. Students rate tutors and courses; tutors access feedback analytics.	
2. Tutors create courses and upload videos, documents, and quizzes.	
3. Secure real-time messaging between students and tutors.	
4. RAG integrated with MongoDB Atlas lets tutors update course materials dynamically without redeploying the AI module.	

Group member - 04	
Name: Md. Ann-Am Akbar Saad	Student ID: 22101438
Functional Requirements that are developed by this member:	
1. User profile customization including bio, subjects, languages, and availability.	
2. Students enroll in courses, track progress, and earn certificates	
3. Dedicated chat channels for courses and tutoring sessions.	
4. RAG-powered chatbot auto-evaluates short answers and provides grading suggestions.	

12. References

Spring Boot Documentation – https://spring.io/projects/spring-boot

MongoDB Atlas Documentation – https://www.mongodb.com/atlas

Spring Security with JWT Tutorials

Research Papers on Retrieval-Augmented Generation (RAG) in Education