**1. What is RabbitMQ?**

RabbitMQ is a **message broker** that enables applications to communicate asynchronously using a **queue-based messaging system**.

**Key Features of RabbitMQ**

✅ **Message Queuing** – Stores and forwards messages between producers and consumers.  
✅ **Asynchronous Processing** – Improves system efficiency by handling tasks in the background.  
✅ **Reliable Delivery** – Ensures messages are not lost even if the system crashes.  
✅ **Scalability** – Supports distributed messaging across multiple servers.  
✅ **Supports Multiple Protocols** – Works with AMQP, MQTT, and WebSockets.

**How RabbitMQ Works?**

1. **Producer** sends a message to a queue.
2. **Queue** stores the message temporarily.
3. **Consumer** fetches the message from the queue and processes it.

**RabbitMQ Use Cases:**

🔹 **Email Processing** – Background email sending (e.g., password reset emails).  
🔹 **Microservices Communication** – Allows independent services to communicate.  
🔹 **Task Processing** – Offloads long-running tasks from the main application.  
🔹 **Real-Time Notifications** – Sends instant updates (e.g., chat applications).

**2. What is SMTP (Simple Mail Transfer Protocol)?**

SMTP is a **protocol for sending emails** over the internet. It is used to transfer messages between **email clients (Gmail, Outlook)** and **email servers**.

**Key Features of SMTP**

✅ **Reliable Email Delivery** – Ensures emails reach the recipient’s inbox.  
✅ **Authentication** – Uses usernames and passwords for security.  
✅ **Encryption Support** – Works with TLS/SSL for secure communication.  
✅ **Works with Other Protocols** – SMTP sends emails, while IMAP/POP3 receives them.

**How SMTP Works?**

1. **User sends an email** using an SMTP client.
2. **SMTP server forwards** the email to the recipient's email server.
3. **Recipient retrieves** the email via IMAP/POP3.

**SMTP Use Cases:**

🔹 **Transactional Emails** – Sending password reset, order confirmations, and OTPs.  
🔹 **Marketing Emails** – Bulk email campaigns.  
🔹 **Automated Notifications** – Alerts for system events.

**RabbitMQ vs SMTP – What’s the Difference?**

| **Feature** | **RabbitMQ** | **SMTP** |
| --- | --- | --- |
| Purpose | Message queuing | Sending emails |
| Asynchronous | Yes | No |
| Reliability | High (Message persistence) | Medium (Depends on mail servers) |
| Use Case | Background job processing | Email communication |

👉 **RabbitMQ is used to queue email requests**, while **SMTP is used to actually send emails**.

**RabbitMQ Implementation in Forgot Password API with JWT Token & SMTP in ASP.NET Core**

In this guide, we'll implement a **Forgot Password API** in **ASP.NET Core** that:

1. **Generates a JWT token** for password reset.
2. **Publishes a message to RabbitMQ** containing the reset token and user email.
3. **Consumes the message** and **sends an email** using SMTP.

**Step 1: Install Required Packages**

Run the following commands in the terminal:

**dotnet add package RabbitMQ.Client**

**dotnet add package MailKit**

**dotnet add package MimeKit**

**dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer**

**dotnet add package System.IdentityModel.Tokens.Jwt**

**Step 2: Configure RabbitMQ in appsettings.json**

{

"RabbitMQ": {

"Host": "localhost",

"Username": "guest",

"Password": "guest",

"QueueName": "email\_queue"

},

"SMTP": {

"Host": "smtp.gmail.com",

"Port": 587,

"Username": "your-email@gmail.com",

"Password": "your-app-password"

},

"Jwt": {

"Key": "YourSuperSecretKey",

"Issuer": "yourdomain.com",

"Audience": "yourdomain.com",

"ExpiryMinutes": 30

}

}

**Step 3: Implement JWT Token Generation (JwtService.cs)**

using System;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

using Microsoft.Extensions.Configuration;

using Microsoft.IdentityModel.Tokens;

public class JwtService

{

private readonly IConfiguration \_config;

public JwtService(IConfiguration config)

{

\_config = config;

}

public string GenerateResetToken(string email)

{

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_config["Jwt:Key"]));

var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

var claims = new[]

{

new Claim(ClaimTypes.Email, email),

new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())

};

var token = new JwtSecurityToken(

\_config["Jwt:Issuer"],

\_config["Jwt:Audience"],

claims,

expires: DateTime.UtcNow.AddMinutes(Convert.ToDouble(\_config["Jwt:ExpiryMinutes"])),

signingCredentials: creds

);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

**Step 4: Implement RabbitMQ Producer (RabbitMQProducer.cs)**

using System.Text;

using Microsoft.Extensions.Configuration;

using Newtonsoft.Json;

using RabbitMQ.Client;

public class RabbitMQProducer

{

private readonly IConfiguration \_config;

public RabbitMQProducer(IConfiguration config)

{

\_config = config;

}

public void PublishMessage(object message)

{

var factory = new ConnectionFactory

{

HostName = \_config["RabbitMQ:Host"],

UserName = \_config["RabbitMQ:Username"],

Password = \_config["RabbitMQ:Password"]

};

using var connection = factory.CreateConnection();

using var channel = connection.CreateModel();

channel.QueueDeclare(queue: \_config["RabbitMQ:QueueName"], durable: false, exclusive: false, autoDelete: false, arguments: null);

var json = JsonConvert.SerializeObject(message);

var body = Encoding.UTF8.GetBytes(json);

channel.BasicPublish(exchange: "", routingKey: \_config["RabbitMQ:QueueName"], basicProperties: null, body: body);

}

}

**Step 5: Implement Forgot Password API (AuthController.cs)**

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Configuration;

using System.Threading.Tasks;

[Route("api/auth")]

[ApiController]

public class AuthController : ControllerBase

{

private readonly JwtService \_jwtService;

private readonly RabbitMQProducer \_rabbitMQProducer;

public AuthController(JwtService jwtService, RabbitMQProducer rabbitMQProducer)

{

\_jwtService = jwtService;

\_rabbitMQProducer = rabbitMQProducer;

}

[HttpPost("forgot-password")]

public IActionResult ForgotPassword([FromBody] ForgotPasswordRequest request)

{

if (string.IsNullOrEmpty(request.Email))

{

return BadRequest("Email is required.");

}

// Generate JWT token for password reset

var resetToken = \_jwtService.GenerateResetToken(request.Email);

// Create email payload

var message = new

{

To = request.Email,

Subject = "Reset Your Password",

Body = $"Click the link to reset your password: https://yourdomain.com/reset-password?token={resetToken}"

};

// Publish message to RabbitMQ

\_rabbitMQProducer.PublishMessage(message);

return Ok("Password reset email has been sent.");

}

}

public class ForgotPasswordRequest

{

public string Email { get; set; }

}

**Step 6: Implement RabbitMQ Consumer (RabbitMQConsumer.cs)**

using System;

using System.Text;

using Microsoft.Extensions.Configuration;

using Newtonsoft.Json;

using RabbitMQ.Client;

using RabbitMQ.Client.Events;

using System.Threading.Tasks;

public class RabbitMQConsumer

{

private readonly IConfiguration \_config;

private readonly EmailService \_emailService;

public RabbitMQConsumer(IConfiguration config, EmailService emailService)

{

\_config = config;

\_emailService = emailService;

}

public void StartListening()

{

var factory = new ConnectionFactory

{

HostName = \_config["RabbitMQ:Host"],

UserName = \_config["RabbitMQ:Username"],

Password = \_config["RabbitMQ:Password"]

};

var connection = factory.CreateConnection();

var channel = connection.CreateModel();

channel.QueueDeclare(queue: \_config["RabbitMQ:QueueName"], durable: false, exclusive: false, autoDelete: false, arguments: null);

var consumer = new EventingBasicConsumer(channel);

consumer.Received += async (model, ea) =>

{

var body = ea.Body.ToArray();

var message = Encoding.UTF8.GetString(body);

var emailMessage = JsonConvert.DeserializeObject<EmailMessage>(message);

await \_emailService.SendEmailAsync(emailMessage.To, emailMessage.Subject, emailMessage.Body);

};

channel.BasicConsume(queue: \_config["RabbitMQ:QueueName"], autoAck: true, consumer: consumer);

}

}

public class EmailMessage

{

public string To { get; set; }

public string Subject { get; set; }

public string Body { get; set; }

}

**Step 7: Implement SMTP Email Sender (EmailService.cs)**

using System;

using System.Net;

using System.Net.Mail;

using System.Threading.Tasks;

using Microsoft.Extensions.Configuration;

public class EmailService

{

private readonly IConfiguration \_config;

public EmailService(IConfiguration config)

{

\_config = config;

}

public async Task SendEmailAsync(string to, string subject, string body)

{

using var client = new SmtpClient(\_config["SMTP:Host"], int.Parse(\_config["SMTP:Port"]))

{

Credentials = new NetworkCredential(\_config["SMTP:Username"], \_config["SMTP:Password"]),

EnableSsl = true

};

var mailMessage = new MailMessage

{

From = new MailAddress(\_config["SMTP:Username"]),

Subject = subject,

Body = body,

IsBodyHtml = true

};

mailMessage.To.Add(to);

await client.SendMailAsync(mailMessage);

}

}

**Step 8: Register Services in Program.cs**

builder.Services.AddSingleton<JwtService>();

builder.Services.AddSingleton<RabbitMQProducer>();

builder.Services.AddSingleton<EmailService>();

builder.Services.AddSingleton<RabbitMQConsumer>();

var app = builder.Build();

// Start RabbitMQ Consumer

var rabbitConsumer = app.Services.GetRequiredService<RabbitMQConsumer>();

Task.Run(() => rabbitConsumer.StartListening());

app.UseAuthorization();

app.MapControllers();

app.Run();

**Conclusion**

✔ **JWT Token Generation** for password reset  
✔ **RabbitMQ Producer-Consumer Pattern** for email processing  
✔ **SMTP Integration** for sending reset emails

TASK:-

Implement RabbitMq with Smtp in forget password and also create reset password after authorization.

1)HelloAppController.cs

| using Microsoft.AspNetCore.Mvc; using BusinessLayer.Interface; using ModelLayer.DTO; using Microsoft.Extensions.Logging; using System; using System.Threading.Tasks;  namespace HelloApp.Controllers {  [ApiController]  [Route("[controller]")]  public class HelloAppController : ControllerBase  {  private readonly IRegisterHelloBL \_registerHelloBL;  private readonly ILogger<HelloAppController> \_logger;   public HelloAppController(ILogger<HelloAppController> logger, IRegisterHelloBL registerHelloBL)  {  \_registerHelloBL = registerHelloBL;  \_logger = logger;  }   [HttpPost("login")]  public async Task<IActionResult> LoginUser([FromBody] LoginDTO loginDTO)  {  try  {  \_logger.LogInformation("Login attempt for user: {Username}", loginDTO.Email);   var token = await \_registerHelloBL.LoginUser(loginDTO);   if (string.IsNullOrEmpty(token))  {  return Unauthorized(new ResponseModel<string>  {  Success = false,  Message = "Invalid email or password",  Data = string.Empty  });  }   return Ok(new ResponseModel<string>  {  Success = true,  Message = "Login successful",  Data = token  });  }  catch (Exception ex)  {  \_logger.LogError(ex, "Exception during login for user: {Username}", loginDTO.Email);   return StatusCode(500, new ResponseModel<string>  {  Success = false,  Message = "An error occurred while processing login",  Data = ex.Message  });  }  }   [HttpPost("register")]  public async Task<IActionResult> RegistrationUser([FromBody] RegisterDTO registerDTO)  {  try  {  var result = await \_registerHelloBL.RegisterBL(registerDTO);   var response = new ResponseModel<RegistrationResponse>  {  Success = true,  Message = "Registration successful",  Data = result  };   return Created("User created", response);  }  catch (Exception ex)  {  \_logger.LogError(ex, "Error during user registration");   return StatusCode(500, new ResponseModel<string>  {  Success = false,  Message = "An error occurred during registration.",  Data = ex.Message  });  }  }  [Authorize]  [HttpPost("forgot-password")]  public async Task<IActionResult> ForgotPassword([FromBody] ForgotPasswordDTO forgotPasswordDTO)  {  try  {  var result = await \_registerHelloBL.ForgotPassword(forgotPasswordDTO);  return Ok(new ResponseModel<string>  {  Success = true,  Message = result,  Data = null  });  }  catch (Exception ex)  {  \_logger.LogError(ex, "Error during forgot password request");  return StatusCode(500, new ResponseModel<string>  {  Success = false,  Message = "An error occurred while processing forgot password request.",  Data = ex.Message  });  }  }    [Authorize]  [HttpPost("reset-password")]  public async Task<IActionResult> ResetPassword([FromBody] ResetPasswordDTO resetPasswordDTO)  {  try  {  var result = await \_registerHelloBL.ResetPassword(resetPasswordDTO);  return Ok(new ResponseModel<string>  {  Success = result,  Message = result ? "Password reset successfully" : "Invalid token or expired",  Data = null  });  }  catch (Exception ex)  {  \_logger.LogError(ex, "Error during reset password request");  return StatusCode(500, new ResponseModel<string>  {  Success = false,  Message = "An error occurred while processing reset password request.",  Data = ex.Message  });  }  }   } } |
| --- |

2) Program.cs

| using BusinessLayer.Interface; using BusinessLayer.Service; using Microsoft.EntityFrameworkCore; using NLog; using NLog.Web; using RepositoryLayer.Context; using RepositoryLayer.Interface; using RepositoryLayer.Service; using Microsoft.AspNetCore.Authentication.JwtBearer; using Microsoft.IdentityModel.Tokens; using Microsoft.OpenApi.Models; using RepositoryLayer.Helper; using System.Text; using StackExchange.Redis;  var logger = NLog.LogManager.Setup().LoadConfigurationFromFile("nlog.config").GetCurrentClassLogger();  try {  logger.Info("Starting application...");   var builder = WebApplication.CreateBuilder(args);   // Retrieve the database connection string  var connectionString = builder.Configuration.GetConnectionString("SqlConnection");   // Configure the application's DbContext to use SQL Server  builder.Services.AddDbContext<HelloAppContext>(options =>  options.UseSqlServer(connectionString));   // Add RabbitMQ  builder.Services.AddSingleton<RabbitMQProducer>();  builder.Services.AddSingleton<RabbitMQConsumer>();   //Add SMTP Email Sender  builder.Services.AddSingleton<EmailSender>();   // Add Redis configuration correctly  var redisConnectionString = builder.Configuration.GetConnectionString("Redis");  if (string.IsNullOrEmpty(redisConnectionString))  {  throw new Exception("Redis connection string is missing in configuration");  }   builder.Services.AddSingleton<IConnectionMultiplexer>(sp =>  {  return ConnectionMultiplexer.Connect(redisConnectionString);  });    // Register services for dependency injection  builder.Services.AddScoped<IRegisterHelloBL , RegisterHelloBL>();  builder.Services.AddScoped<IRegisterHelloRL, RegisterHelloRL>();   builder.Services.AddScoped<JwtHelper>();   builder.Services.AddTransient<EmailSender>();   // Add controllers  builder.Services.AddControllers();   // Configure JWT Authentication  var jwtSettings = builder.Configuration.GetSection("Jwt");  var jwtKey = jwtSettings["Key"];   if (string.IsNullOrEmpty(jwtKey))  {  throw new Exception("JWT Key is missing in configuration");  }   var key = Encoding.UTF8.GetBytes(jwtKey);   builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)  .AddJwtBearer(options =>  {  options.TokenValidationParameters = new TokenValidationParameters  {  ValidateIssuerSigningKey = true,  IssuerSigningKey = new SymmetricSecurityKey(key),  ValidateIssuer = true,  ValidateAudience = true,  ValidIssuer = jwtSettings["Issuer"],  ValidAudience = jwtSettings["Audience"],  ValidateLifetime = true,  ClockSkew = TimeSpan.Zero  };  });   // Register Swagger for API documentation  builder.Services.AddEndpointsApiExplorer();  builder.Services.AddSwaggerGen(c =>  {  c.SwaggerDoc("v1", new OpenApiInfo { Title = "User Authentication API", Version = "v1" });   c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme  {  In = ParameterLocation.Header,  Description = "Please enter a valid token",  Name = "Authorization",  Type = SecuritySchemeType.Http,  BearerFormat = "JWT",  Scheme = "Bearer"  });   c.AddSecurityRequirement(new OpenApiSecurityRequirement  {  {  new OpenApiSecurityScheme  {  Reference = new OpenApiReference  {  Type = ReferenceType.SecurityScheme,  Id = "Bearer"  }  },  Array.Empty<string>()  }  });  });    // NLog: Setup NLog for Dependency injection  builder.Logging.ClearProviders();  builder.Host.UseNLog();   var app = builder.Build();   // Configure the HTTP request pipeline  if (app.Environment.IsDevelopment())  {  app.UseSwagger();  app.UseSwaggerUI();   }  app.UseHttpsRedirection();  app.UseAuthentication();  app.UseAuthorization();  app.MapControllers();   app.Run(); } catch (Exception ex) {  logger.Error(ex, "Application stopped due to an exception");  throw; } finally {  LogManager.Shutdown(); } |
| --- |

3)RegisterHelloBL.cs

| using System; using System.Threading.Tasks; using BusinessLayer.Interface; using ModelLayer.DTO; using RepositoryLayer.Interface; using Microsoft.Extensions.Logging; using RepositoryLayer.Entity; using StackExchange.Redis; using System.Text.Json; using RepositoryLayer.Helper; using System.Net.Mail;  namespace BusinessLayer.Service {  public class RegisterHelloBL : IRegisterHelloBL  {  private readonly IRegisterHelloRL \_registerHelloRL;  private readonly ILogger<RegisterHelloBL> \_logger;  private readonly IConnectionMultiplexer \_redis;   public RegisterHelloBL(IRegisterHelloRL registerHelloRL, ILogger<RegisterHelloBL> logger, IConnectionMultiplexer redis)  {  \_registerHelloRL = registerHelloRL;  \_logger = logger;  \_redis = redis;  }   public async Task<string> LoginUser(LoginDTO loginDTO)  {  try  {  string cacheKey = $"user\_login\_{loginDTO.Email}";  var cache = \_redis.GetDatabase();   var cachedToken = await cache.StringGetAsync(cacheKey);  if (!cachedToken.IsNullOrEmpty)  {  \_logger.LogInformation("Cache hit: Returning cached token for {Email}", loginDTO.Email);  return cachedToken.ToString();  }   string result = await \_registerHelloRL.Login(loginDTO);   if (!string.IsNullOrEmpty(result))  {  \_logger.LogInformation("Login successful for user: {Email}", loginDTO.Email);  await cache.StringSetAsync(cacheKey, result, TimeSpan.FromMinutes(30));  }  else  {  \_logger.LogWarning("Login failed for user: {Email}", loginDTO.Email);  }   return result;  }  catch (Exception ex)  {  \_logger.LogError(ex, "Error during login for user: {Email}", loginDTO.Email);  throw;  }  }   public async Task<RegistrationResponse> RegisterBL(RegisterDTO registerDTO)  {  try  {  var result = await \_registerHelloRL.Registration(registerDTO);  var response = new RegistrationResponse  {  UserId = result.UserId,  FirstName = result.FirstName,  LastName = result.LastName,  Email = result.Email  };   return response;  }  catch (Exception ex)  {  \_logger.LogError(ex, "Error in user registration");  throw;  }  }   public async Task<string> ForgotPassword(ForgotPasswordDTO forgotPasswordDTO)  {  var result = await \_registerHelloRL.ForgotPassword(forgotPasswordDTO);   return result;  }   public async Task<bool> ResetPassword(ResetPasswordDTO resetPasswordDTO)  {  return await \_registerHelloRL.ResetPassword(resetPasswordDTO);  }   } } |
| --- |

4) RegisterHelloRL.cs

| using System; using ModelLayer.DTO; using RepositoryLayer.Interface; using Microsoft.Extensions.Logging; using RepositoryLayer.Context; using Microsoft.EntityFrameworkCore; using RepositoryLayer.Entity;  namespace RepositoryLayer.Service {  public class RegisterHelloRL : IRegisterHelloRL  {  private readonly ILogger<RegisterHelloRL> \_logger;  private readonly HelloAppContext \_DbContext;   public RegisterHelloRL(ILogger<RegisterHelloRL> logger, HelloAppContext DbContext)  {  \_logger = logger;  \_DbContext = DbContext;  }   public string GetHello(string name)  {  try  {  return name + " Hello from Repository layer ";  }  catch (Exception ex)  {  \_logger.LogError(ex, "Error in GetHello method");  throw;  }  }   public UserEntity GetUsernamePassword(LoginDTO loginDTO)  {  try  {  var existingUser = \_DbContext.Users.FirstOrDefault<UserEntity>(e =>  e.Email == loginDTO.Email && e.Password == loginDTO.Password   );   return existingUser;  }  catch (Exception ex)  {  \_logger.LogError(ex, "Error fetching username and password");  throw;  }  }   public UserEntity Registration(RegisterDTO registerDTO)  {  var existingUser = \_DbContext.Users.FirstOrDefault<UserEntity>(e =>  e.Email== registerDTO.Email  );   if(existingUser == null)  {  var newUser = new UserEntity  {   FirstName = registerDTO.FirstName,  LastName = registerDTO.LastName,  Email = registerDTO.Email,  Password = registerDTO.Password  };   \_DbContext.Users.Add(newUser);  \_DbContext.SaveChanges();  return newUser;  }     return existingUser;  }  } } |
| --- |

5)EmailSender.cs

| using System; using System.Collections.Generic; using System.Linq; using System.Net; using System.Net.Mail; using System.Text; using System.Threading.Tasks;  namespace RepositoryLayer.Helper {  public class EmailSender  {  public Task SendEmailAsync(string Email , string Subject , string Message)  {  var mailSender = "rohitdixit570@gmail.com";  var password = "nddn mtrf kstf qtjf";   var client = new SmtpClient("smtp.gmail.com")  {  Port = 587,  EnableSsl = true,  Credentials = new NetworkCredential(mailSender, password)  };   return client.SendMailAsync(  new MailMessage(  from: mailSender,  to: Email,  Subject,  Message  )  );  }  } } |
| --- |

6) JwtHelper.cs

| using System; using System.Collections.Generic; using System.IdentityModel.Tokens.Jwt; using System.Security.Claims; using System.Text; using Microsoft.IdentityModel.Tokens; using Microsoft.Extensions.Configuration; using RepositoryLayer.Entity;  namespace RepositoryLayer.Helper {  public class JwtHelper  {  private readonly IConfiguration \_configuration;  public JwtHelper(IConfiguration configuration)  {  \_configuration = configuration;  }   public string Tokenize(UserEntity user)  {  var claims = new[]  {  new Claim(JwtRegisteredClaimNames.Sub, \_configuration["Jwt:Subject"]),  new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),  new Claim("UserId", user.UserId.ToString()),  new Claim("Email", user.Email),  new Claim("Name", $"{user.FirstName} {user.LastName}")  };   var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_configuration["Jwt:Key"]));  var signIn = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);  var token = new JwtSecurityToken(  \_configuration["Jwt:Issuer"],  \_configuration["Jwt:Audience"],  claims,  expires: DateTime.UtcNow.AddMinutes(60),  signingCredentials: signIn  );   return new JwtSecurityTokenHandler().WriteToken(token);  }   public string GenerateResetToken(UserEntity user)  {  var claims = new[]  {  new Claim("Email", user.Email),  new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())  };   var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_configuration["Jwt:Key"]));  var signIn = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);  var token = new JwtSecurityToken(  \_configuration["Jwt:Issuer"],  \_configuration["Jwt:Audience"],  claims,  expires: DateTime.UtcNow.AddMinutes(15),  signingCredentials: signIn  );   return new JwtSecurityTokenHandler().WriteToken(token);  }   } } |
| --- |

6) RabbitMQConsumer.cs

| using Microsoft.Extensions.Configuration; using RabbitMQ.Client.Events; using RabbitMQ.Client; using Newtonsoft.Json; using System.Net.Mail; using System.Text; using ModelLayer.DTO;  namespace RepositoryLayer.Helper {  public class RabbitMQConsumer  {  private readonly IConfiguration \_config;  private readonly EmailSender \_emailService;   public RabbitMQConsumer(IConfiguration config, EmailSender emailService)  {  \_config = config;  \_emailService = emailService;  }   public void StartListening()  {  var factory = new ConnectionFactory  {  HostName = "localhost",  UserName = "Rohit",  Password = "Rohhit@770"  };   var connection = factory.CreateConnection();  var channel = connection.CreateModel();  channel.QueueDeclare(queue: "Email\_Queue",  durable: false,  exclusive: false,  autoDelete: false,  arguments: null);   var consumer = new EventingBasicConsumer(channel);  consumer.Received += async (model, ea) =>  {  var body = ea.Body.ToArray();  var message = Encoding.UTF8.GetString(body);  var emailMessage = JsonConvert.DeserializeObject<EmailMessage>(message);   await \_emailService.SendEmailAsync(emailMessage.To, emailMessage.Subject, emailMessage.Body);  };   channel.BasicConsume(  queue: "Email\_Queue",  autoAck: true,  consumer: consumer);  }  }  } |
| --- |

7) RabbitMQPRoducer.cs

| using RabbitMQ.Client; using System.Text; using System.Text.Json; using Microsoft.Extensions.Configuration; using Newtonsoft.Json;  namespace RepositoryLayer.Helper {  public class RabbitMQProducer  {  private readonly IConfiguration \_config;   public RabbitMQProducer(IConfiguration config)  {  \_config = config;  }   public void PublishMessage(object message)  {  var factory = new ConnectionFactory() { HostName = "localhost" };  using var connection = factory.CreateConnection();  using var channel = connection.CreateModel();    channel.QueueDeclare(  queue: "emailQueue",  durable: false,  exclusive: false,  autoDelete: false,  arguments: null  );   var json = JsonConvert.SerializeObject(message);  var body = Encoding.UTF8.GetBytes(json);   channel.BasicPublish(  exchange: "",  routingKey: "emailQueue",  basicProperties: null,  body: body  );   }  }  } |
| --- |

8)EmailMessage.cs

| using System; using System.Collections.Generic; using System.Linq; using System.Text; using System.Threading.Tasks;  namespace ModelLayer.DTO {  public class EmailMessage  {  public string? To { get; set; }  public string? Subject { get; set; }  public string? Body { get; set; }  }  } |
| --- |

9) ForgotPasswordDTO.cs

| using System; using System.Collections.Generic; using System.Linq; using System.Text; using System.Threading.Tasks;  namespace ModelLayer.DTO {  public class ForgotPasswordDTO  {  public string? Email { get; set; }  }  } |
| --- |

10) ResetPasswordDTO.cs

| using System; using System.Collections.Generic; using System.Linq; using System.Text; using System.Threading.Tasks;  namespace ModelLayer.DTO {  public class ResetPasswordDTO  {  public string? Token { get; set; }  public string? NewPassword { get; set; }  public string? ConfirmPassword { get; set; }  }  } |
| --- |

11) LoginDTO.cs

| namespace ModelLayer.DTO {  public class LoginDTO  {  public string? Email { get; set; }   public string? Password { get; set; }   public override string ToString()  {  return $"Username = {Email}, Password = {Password}";   }  } } |
| --- |

12)RegisterDTO.cs

| using System;  using System.Collections.Generic; using System.Linq; using System.Text; using System.Threading.Tasks;  namespace ModelLayer.DTO {  public class RegisterDTO  {  public string FirstName { get; set; }  public string LastName { get; set; }  public string Password { get; set; }  public string Email { get; set; }    } } |
| --- |