

Foi desenvolvido um backend utilizando C# e .NET 8, com a finalidade de gerenciar usuários de forma eficiente. O sistema implementa uma série de endpoints que possibilitam as seguintes funcionalidades: cadastro, edição e listagem de usuários. Ademais, um endpoint GET foi disponibilizado para recuperar uma lista de e-mails cadastrados.

Outra funcionalidade importante é um endpoint GET dedicado à personalização do tema, permitindo que o usuário especifique a cor desejada. O armazenamento dos dados dos usuários é realizado por meio do banco de dados MongoDB, e as requisições estão sendo testadas utilizando a ferramenta Postman, garantindo a qualidade e a integridade das operações realizadas.

1. Controller que executa o endpoint para busca, criação ou edição do usuário.

```

using Microsoft.AspNetCore.Mvc;
using email_api.Models;
using email_api.Services;

namespace email_api.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class UserController : ControllerBase
    {
        private readonly UserService _userService;

        public UserController(UserService userService)
        {
            _userService = userService;
        }

        [HttpPost]
        public async Task<IActionResult> Post([FromBody] User user)
        {
            if (user == null)
            {
                return BadRequest("User cannot be null");
            }

            bool emailExists = await _userService.EmailExistsAsync(user.Email);
            if (emailExists)
            {
                return Conflict("Email already exists");
            }

            await _userService.CreateUserAsync(user);
            return CreatedAtAction(nameof(Get), new { id = user.Id }, user);
        }

        [HttpGet]
        public async Task<IActionResult> Get(...)

        [HttpPut("{id}")]
        public async Task<IActionResult> Put(string id, [FromBody] User updatedUser)
    }
}

```

2. Método post, responsável por criar o usuário, verificando se o e-mail que o usuário está passando já é existente na base, se sim, ele retorna um erro e não irá salvar.

```

[HttpPost]
public async Task<IActionResult> Post([FromBody] User user)
{
    if (user == null)
    {
        return BadRequest("User cannot be null");
    }

    bool emailExists = await _userService.EmailExistsAsync(user.Email);
    if (emailExists)
    {
        return Conflict("Email already exists");
    }

    await _userService.CreateUserAsync(user);
    return CreatedAtAction(nameof(Get), new { id = user.Id }, user);
}

```

3. Método Get responsável por retornar os usuários na base.

```

[HttpGet]
public async Task<IActionResult> Get()
{
    var users = await _userService.GetUsersAsync();
    return Ok(users);
}

```

4. Método Put responsável por editar o usuário.

```

[HttpPut("{id}")]
public async Task<IActionResult> Put(string id, [FromBody] User updatedUser)
{
    if (updatedUser == null)
    {
        return BadRequest("User cannot be null");
    }

    var existingUser = await _userService.GetUserByIdAsync(id);
    if (existingUser == null)
    {
        return NotFound("User not found");
    }

    if (existingUser.Email != updatedUser.Email)
    {
        bool emailExists = await _userService.EmailExistsAsync(updatedUser.Email);
        if (emailExists)
        {
            return Conflict("Email already exists");
        }
    }

    updatedUser.Id = id;
    await _userService.UpdateUserAsync(id, updatedUser);

    return NoContent();
}

```

5. Serviço responsável por executar cada função para cada método deste endpoint /User.

```
UserController.cs  UserService.cs  X
Arquivos Diversos  email_api.Services.UserService

1  using MongoDB.Driver;
2  using email_api.Models;
3
4  namespace email_api.Services
5  {
6      public class UserService
7      {
8          private readonly IMongoCollection<User> _users;
9
10         public UserService(IMongoClient mongoClient)
11         {
12             var database = mongoClient.GetDatabase("baseEmailCode");
13             _users = database.GetCollection<User>("Users");
14         }
15
16
17
18
19         public async Task CreateUserAsync(User user)
20         {
21             await _users.InsertOneAsync(user);
22         }
23
24
25
26         public async Task<List<User>> GetUsersAsync()
27         {
28             return await _users.Find(user => true).ToListAsync();
29         }
30
31
32
33         public async Task<User?> GetUserByIdAsync(string id)
34         {
35             return await _users.Find(user => user.Id == id).FirstOrDefaultAsync();
36         }
37
38
39
40         public async Task<bool> EmailExistsAsync(string email)
41         {
42             var user = await _users.Find(u => u.Email == email).FirstOrDefaultAsync();
43             return user != null;
44         }
45
46         public async Task UpdateUserAsync(string id, User updatedUser)
47         {
48             var filter = Builders<User>.Filter.Eq(u => u.Id, id);
49             var update = Builders<User>.Update
50                 .Set(u => u.Name, updatedUser.Name)
51                 .Set(u => u.Email, updatedUser.Email)
52                 .Set(u => u.Telefone, updatedUser.Telefone);
53
54             await _users.UpdateOneAsync(filter, update);
55         }
56     }
57 }
```

6. Controller do e-mail, que realiza um put responsável por passar uma chave como parâmetro para retornar a listagem dos e-mails.

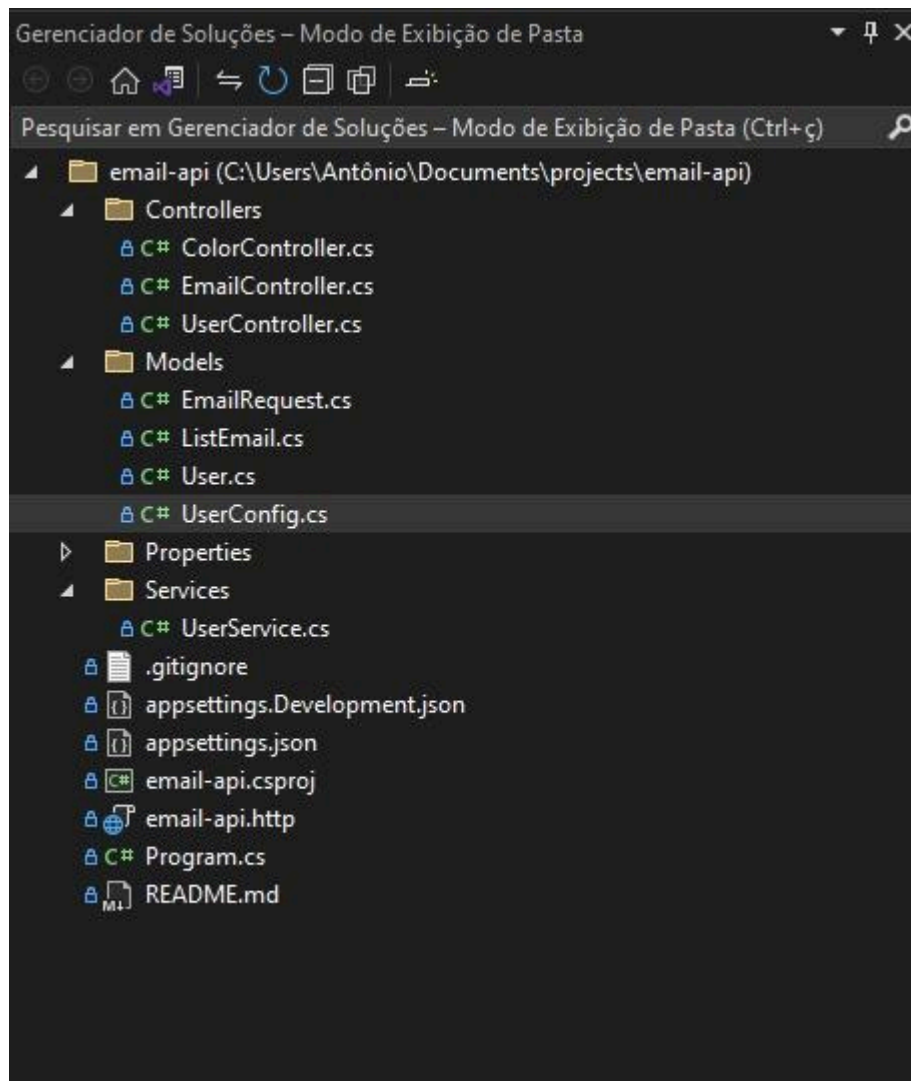
```
EmailController.cs
1 using Microsoft.AspNetCore.Mvc;
2 using email_api.Models;
3
4 namespace email_api.Controllers
5 {
6     [ApiController]
7     [Route("api/[controller]")]
8     public class EmailController : ControllerBase
9     {
10         private const string RequiredKey = "ausdh12342111haishiasaaauihudsfnh";
11
12         [HttpPost]
13         public IActionResult Post([FromBody] EmailRequest request)
14         {
15             if (request == null)
16             {
17                 return BadRequest("Dados inválidos.");
18             }
19
20             if (string.IsNullOrEmpty(request.SendEmail))
21             {
22                 return BadRequest("O email é obrigatório.");
23             }
24
25             if (string.IsNullOrEmpty(request.Key))
26             {
27                 return BadRequest("A chave é obrigatória.");
28             }
29
30             if (request.Key != RequiredKey)
31             {
32                 return BadRequest("Chave inválida.");
33             }
34
35             var emails = GetEmails();
36             return Ok(emails);
37         }
38
39         private List<Email> GetEmails()
40         {
41             var emails = new List<Email>
42             {
43                 new Email
44                 {
45                     Remetente = "joao.silva@example.com",
46                     Destinatario = "kamilly.souza@example.com",
47                     Assunto = "Reunião de Projeto",
48                     Conteudo = "Olá Maria, gostaria de confirmar a reunião para amanhã às 10h.",
49                     DataHora = "2024-09-17 09:00"
50                 },
51                 new Email
52                 {
53                     Remetente = "ana.santos@example.com",
54                     Destinatario = "pedro.alves@example.com",
55                     Assunto = "Proposta de Colaboração",
56                     Conteudo = "Olá Pedro, estou enviando a proposta para nossa colaboração.",
57                     DataHora = "2024-09-17 10:00"
58                 },
59                 new Email{...},
60                 new Email{...},
61                 new Email{...}
62             };
63
64             return emails.Take(5).ToList();
65         }
66     }
67 }
```

7. Controller - responsável pela cor do tema da aplicação, executando um post, onde o usuário passar por parâmetro a cor desejada, e essa cor é retornada nesse método POST, alterando o tema e cores do App.

```
ColorController.cs  X
email-api  SeuNamespace.Controllers.ColorController

1  using Microsoft.AspNetCore.Mvc;
2
3  namespace SeuNamespace.Controllers
4  {
5      [ApiController]
6      [Route("[controller]")]
7      public class ColorController : ControllerBase
8      {
9          [HttpPost]
10         public IActionResult Post([FromBody] ColorRequest request)
11         {
12
13             if (request == null)
14             {
15                 return BadRequest("Dados inválidos.");
16             }
17
18             if (string.IsNullOrEmpty(request.ColorName))
19             {
20                 return BadRequest("O nome da cor é obrigatório.");
21             }
22
23             string responseMessage = $"A cor {request.ColorName} tem intensidade {request.Intensity}.";
24             return Ok(responseMessage);
25         }
26     }
27 }
28
29
```

8. Estrutura de pastas.



9. Configuração do projeto e conexão do banco MongoDB


```
appsettings.json  email-api.csproj  UserConfig.cs
Esquema: https://json.schemastore.org/appsettings.json
1  {
2    "Logging": {
3      "LogLevel": {
4        "Default": "Information",
5        "Microsoft.AspNetCore": "Warning"
6      }
7    },
8    "AllowedHosts": "*",
9    "ConnectionStrings": {
10     "MongoDB": "mongodb://localhost:27017" // Conexão banco
11   }
12 }
13
```

10. Banco com a tabela de usuários criadas.

