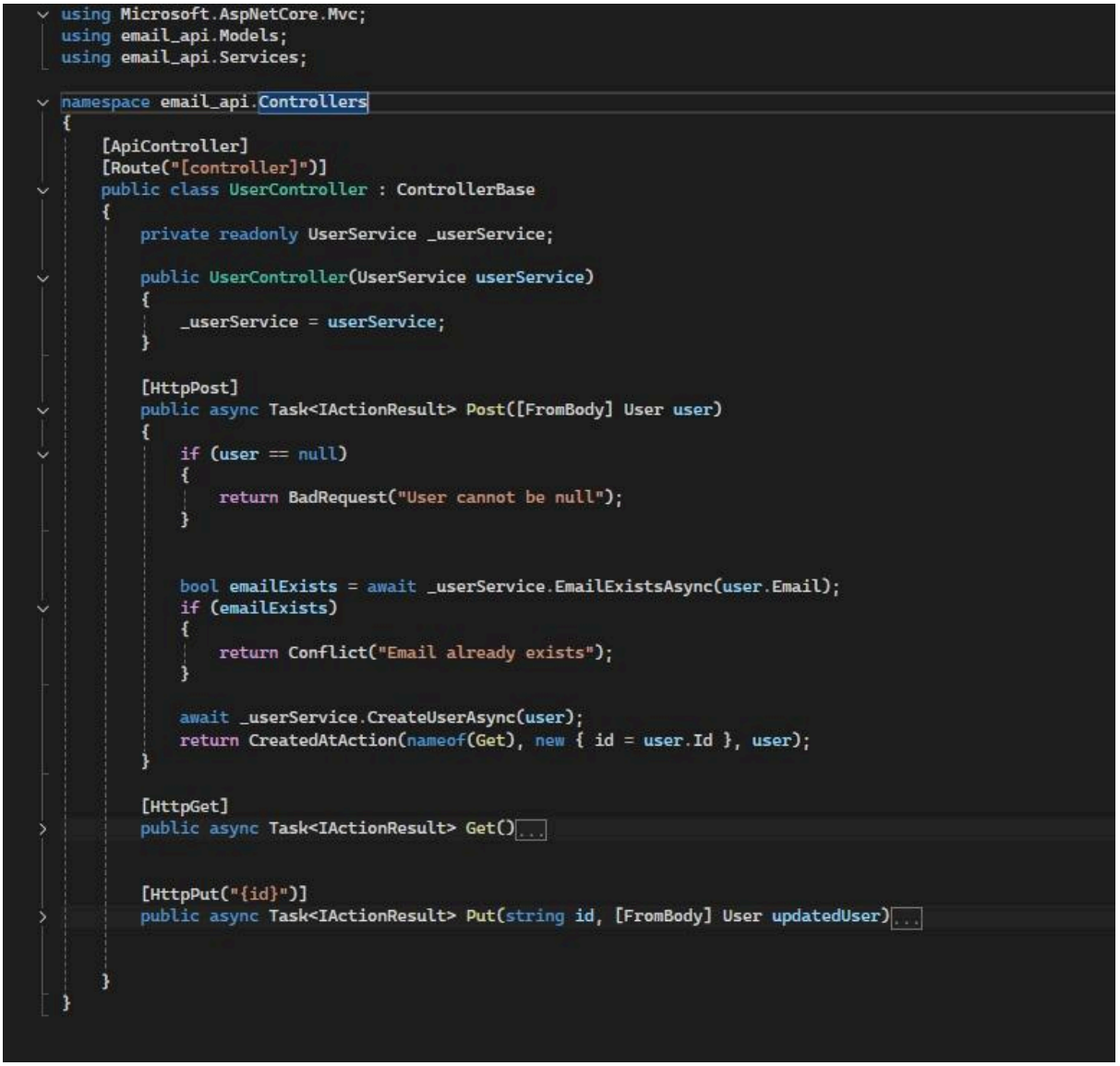
Foi desenvolvido um backend utilizando C# e .NET 8, com a finalidade de gerenciar usuários de forma eficiente. O sistema implementa uma série de endpoints que possibilitam as seguintes funcionalidades: cadastro, edição e listagem de usuários. Ademais, um endpoint GET foi disponibilizado para recuperar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

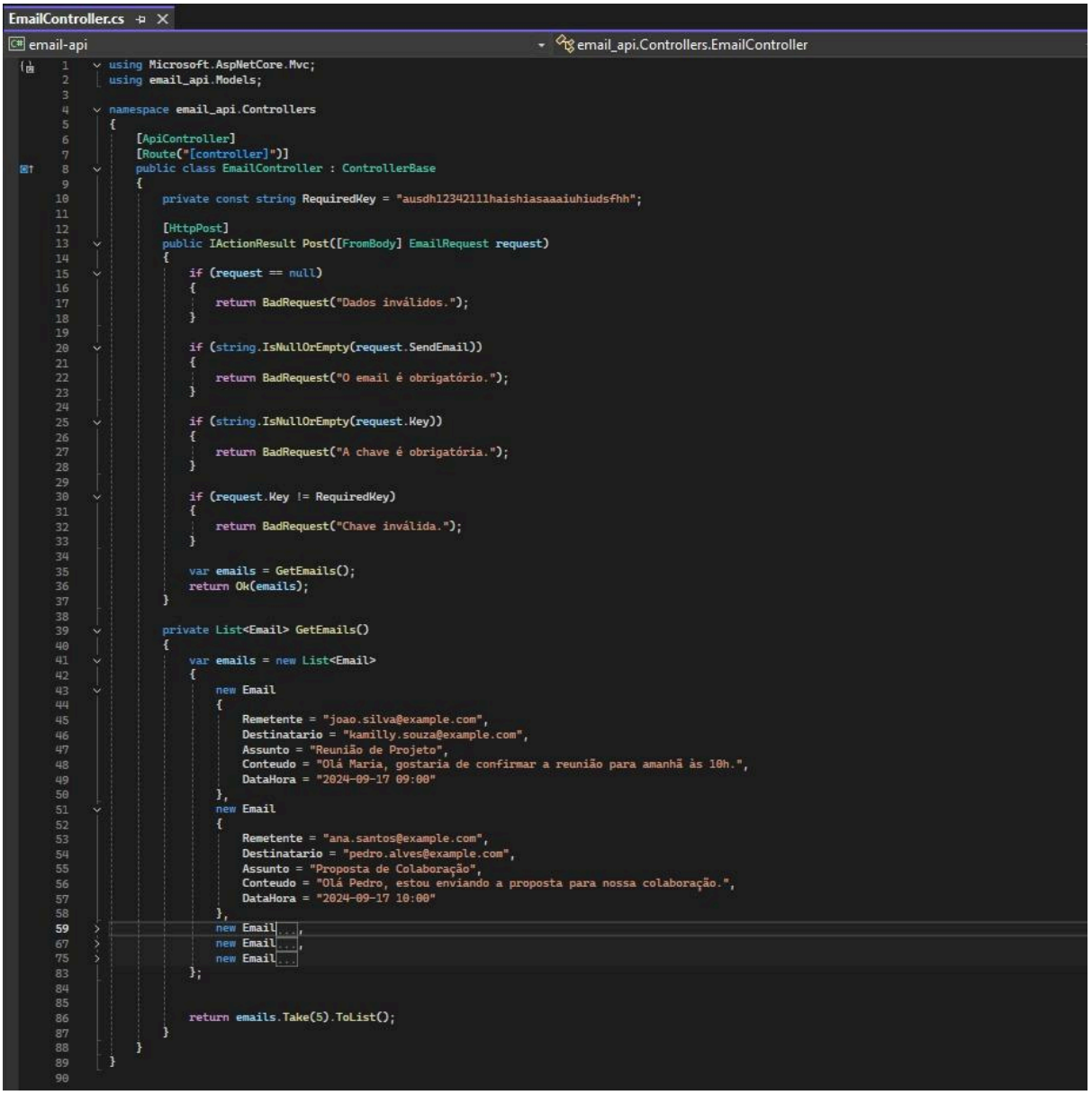
5. **Serviço responsável por executar cada função para cada método deste endpoint /User.**

UserController.cs | UserService.cs

Arquivos Diversos — email_api.Services.UserService

```
using MongoDB.Driver;
using email_api.Models;

namespace email_api.Services
{
    public class UserService
    {
        private readonly IMongoCollection<User> _users;

        public UserService(IMongoClient mongoClient)
        {

            var database = mongoClient.GetDatabase("baseEmailCode");
            _users = database.GetCollection<User>("Users");
        }



        public async Task CreateUserAsync(User user)
        {
            await _users.InsertOneAsync(user);
        }



        public async Task<List<User>> GetUsersAsync()
        {
            return await _users.Find(user => true).ToListAsync();
        }



        public async Task<User?> GetUserByIdAsync(string id)
        {
            return await _users.Find(user => user.Id == id).FirstOrDefaultAsync();
        }



        public async Task<bool> EmailExistsAsync(string email)
        {
            var user = await _users.Find(u => u.Email == email).FirstOrDefaultAsync();
            return user != null;
        }

        public async Task UpdateUserAsync(string id, User updatedUser)
        {
            var filter = Builders<User>.Filter.Eq(u => u.Id, id);
            var update = Builders<User>.Update
                .Set(u => u.Name, updatedUser.Name)
                .Set(u => u.Email, updatedUser.Email)
                .Set(u => u.Telefone, updatedUser.Telefone);

            await _users.UpdateOneAsync(filter, update);
        }
    }
}
```

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



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.**

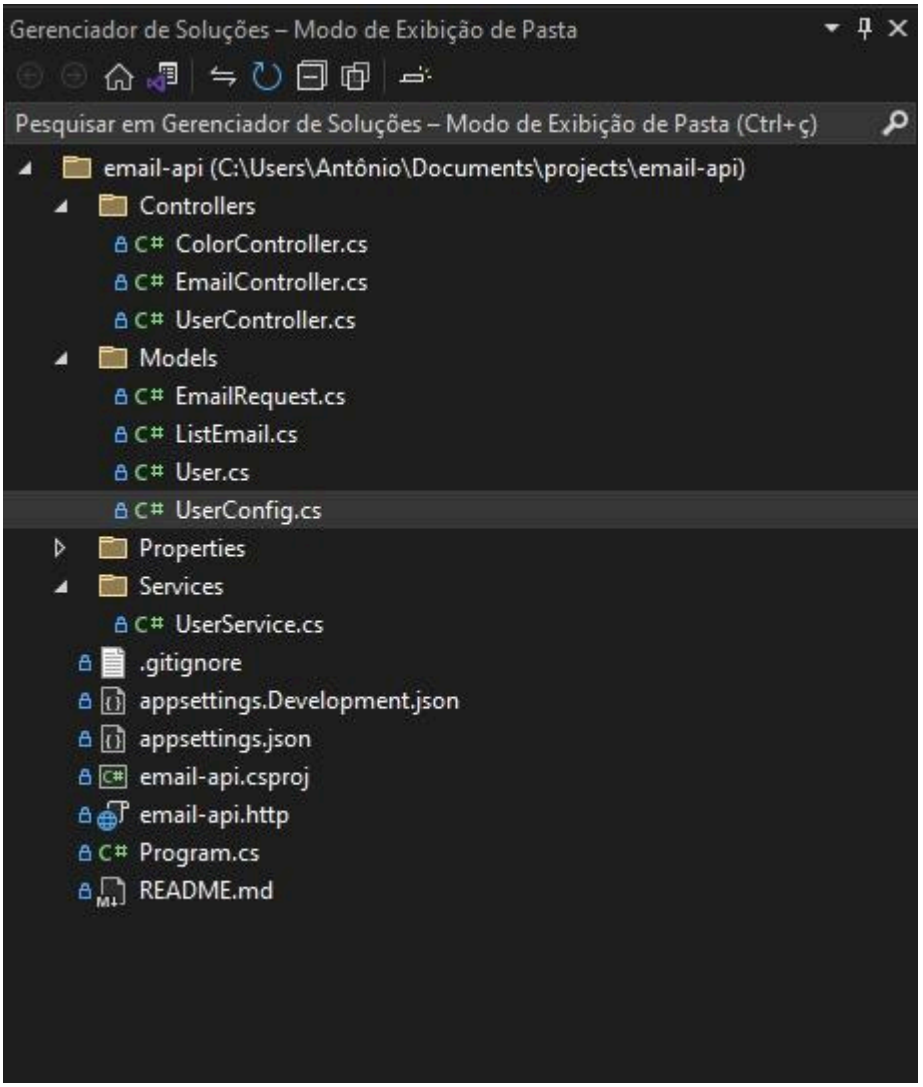
ColorController.cs

email-api

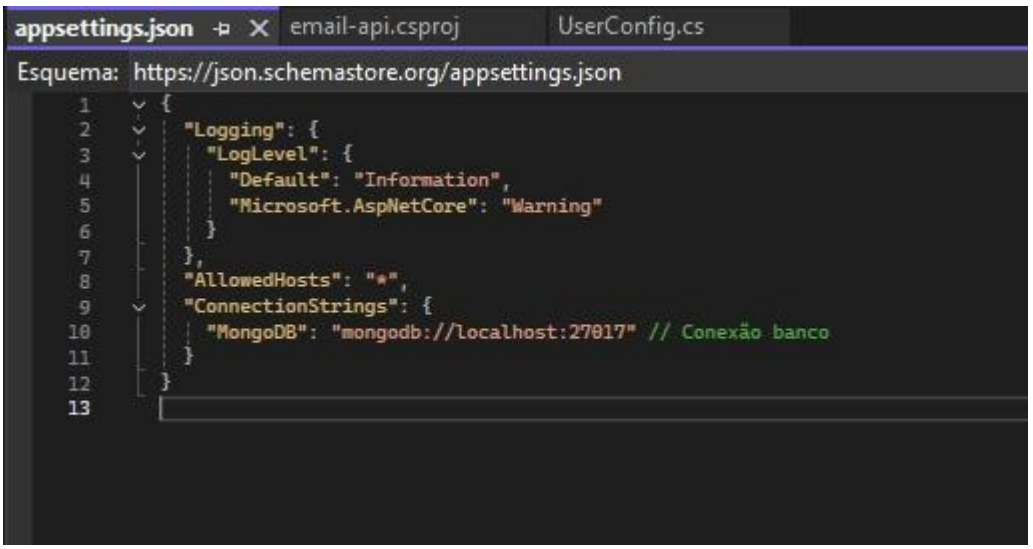
SeuNamespace.Controllers.ColorController

```
using Microsoft.AspNetCore.Mvc;

namespace SeuNamespace.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class ColorController : ControllerBase
    {
        [HttpPost]
        public IActionResult Post([FromBody] ColorRequest request)
        {

            if (request == null)
            {
                return BadRequest("Dados inválidos.");
            }

            if (string.IsNullOrEmpty(request.ColorName))
            {
                return BadRequest("O nome da cor é obrigatório.");
            }


            string responseMessage = $"A cor {request.ColorName} tem intensidade {request.Intensity}.";

            return Ok(responseMessage);
        }
    }
}
```

8. Estrutura de pastas.



9. Configuração do projeto e conexão do banco MongoDB



10. Banco com a tabela de usuários criadas.