

Trabalho avaliativo – Controle de tarefas

Técnico em Desenvolvimento de Sistemas

Setembro/2025

Blumenau/SC

**17/12/2015**

Florianópolis/SC

1. Contextualização

Para facilitar o controle e gerenciamento das demandas de atendimento do setor de suporte, empresas do setor de TI buscam constantemente soluções de software que tornem esse processo mais simples, prático e dinâmico.

Neste sentido será necessário realizar o desenvolvimento de um sistema de tarefas para receber as demandas do help desk com recursos computacionais.

2. Desafio

Para ajudar nesse problema, você deverá criar um sistema em sua plataforma de desenvolvimento (JAVA) utilizada na UC, que permita a gestão das demandas que a equipe de suporte de uma empresa de TI recebe diuturnamente dos seus clientes a fim de controlarem as tarefas que precisam ser realizadas.

Para que o projeto consiga realizar as ações necessárias para apoiar a solução o sistema deverá implementar os métodos de inserir, atualizar, excluir e consultar os dados das entidades de Usuário e Tarefas.

O sistema deverá implementar o acesso aos recursos através de webservices REST (API REST) não havendo qualquer tipo de autenticação e autorização e também sem nenhuma interface para o usuário, sendo apenas a implementação e consumo via requisições http (postman, vscode + thunderclient ou insomnia).

O projeto deverá seguir as orientações de padrão de projetos definidos em sala, evitando a utilização de objetos Entity nos controladores da API e sempre utilizando os objetos do tipo Dto para receber e enviar conjuntos de dados através das classes.

O projeto deverá estar nesta arquitetura que segue:

📁 **Task-springboot**  
├── 📁 **src/main/java/com/senai/task**  
│ ├── 📁 **controllers**  
│ │ ├── <Criar as classes de controllers aqui>

│ ├── 📁 **dtos**  
│ │ ├── <Criar as classes de dtos aqui>

│ ├── 📁 **models**

│ │ ├── <Criar as classes de models aqui>

│ ├── 📁 **repositories**  
│ │ ├── <Criar as interfaces de repositories aqui>

│ ├── 📁 **services**  
│ │ ├── <Criar as classes de services aqui>

│ ├── 📄 TaskApplication.java  
├── 📄 **pom.xml**

2.1. Requisitos

|  |  |  |
| --- | --- | --- |
| N° | Nome da entrega | Descrição da entrega |
| 01 | API para inserir usuário  Ponto (0,5) | O sistema deverá implementar o método POST para realizar a inserção de um usuário novo. |
| 02 | API para atualizar usuário  Ponto (0,5) | O sistema deverá implementar o método PUT para realizar a atualização de um usuário já existente. |
| 03 | API para excluir usuário  Ponto (0,5) | O sistema deverá implementar um método DELETE para realizar a exclusão de um usuário já existente. |
| 04 | API para consultar usuário  Ponto (0,5) | O sistema deverá implementar um método GET para realizar a consulta de usuários cadastrados. |
| 05 | API para inserir tarefa  Ponto (1,0) | O sistema deverá implementar o método POST para realizar a inserção de uma tarefa nova. |
| 06 | API para atualizar tarefa  Ponto (1,5) | O sistema deverá implementar o método PUT para realizar a atualização de uma tarefa já existente. |
| 07 | API para excluir tarefa  Ponto (1,0) | O sistema deverá implementar um método DELETE para realizar a exclusão de uma tarefa já existente. |
| 08 | API para consultar tarefa  Ponto (1,5) | O sistema deverá implementar um método GET para realizar a consulta de tarefas cadastradas. |
| 09 | Validação de usuário na inserção e atualização da tarefa.  Ponto (1,5) | O sistema deverá implementar um tratamento de validação ao inserir ou atualizar uma tarefa validando se o usuário informado existe. Caso não exista deve retornar um erro. |
| 10 | Consultas das API’s no VSCode  Ponto (0,5) | O aluno deverá entregar os scripts de execução dos métodos implementados. |
| 11 | Validação de data para o usuário e tarefa em conflito.  Ponto (1,0) | O sistema deverá realizar um tratamento de validação ao inserir ou atualizar uma tarefa validando se o usuário informado já possui tarefa agendada para a data especificada. |

* ***Especificações***

***RF001: Classes de modelo:*** O sistema deverá possuir classes para representar a entidades de Usuário e Tarefa. Estas entidades possuem um relacionamento de “1 para N” em relação a tarefa e usuário, isso quer dizer que a atividade deverá possuir um campo para armazenar o usuário gerando o relacionamento.

[ ] Entidade Usuário (UserModel):

[ ] id (Long, chave primária, auto numerada)

[ ] nome (string)

[ ] email (string, índice único) (só poderá haver um usuário por e-mail)

[ ] Entidade Tarefa (TaskModel):

[ ] id (Long, chave primária, auto numerada)

[ ] titulo (string)

[ ] descrição (string)

[ ] data de agendamento (date)

[ ] status (string ou inteiro utilizando enum (enumeradores)

Exemplo de status: em aberto, pendente, concluído, cancelado etc)

[ ] Usuario (Objeto UsuarioModel, relacionamento Many-to-One)

***RF002: Classe DTO de usuário:*** O sistema deverá possuir objetos DTO para realizar o recebimento de informações assim como as respostas necessárias.

[ ] DTO Usuário (UserDto):

[ ] nome (string)

[ ] não permitir valor em branco (validação)

[ ] email (string)

[ ] validar formato de e-mail (validação)

***RF003: Classe DTO de tarefa:***

[ ] DTO Tarefa (TaskDTO):

[ ] id (Long)

[ ] Utilizado apenas para o método atualizar, na criação pode ser desconsiderado, ou seja, pode vir a ser branco/nulo.

[ ] titulo (String)

[ ] não permitir valor em branco

[ ] descricao (String)

[ ] não permitir valor em branco

[ ] dataAgendamento (Date)

[ ] não permitir valor em branco

[ ] status (string ou inteiro utilizando enum (enumeradores)

Exemplo de status: pendente, concluído, cancelado etc)

[ ] permitir ( 1 – em aberto, 2 – em andamento, 3 – concluído e 4 cancelado)

[ ] emailUsuario ( este campo deverá ser usado para localizar o usuário, caso o usuário não seja localizado deverá retornar um erro, caso o usuário seja localizado deverá ser utilizado esse usuarioModel para atribuir no objeto do tipo TaskModel.

[ ] não permitir valor em branco

***RF004: Classe controladora de usuário:*** O sistema deverá implementar os seguintes webservices REST a fim de disponibilizar os serviços para obter a lista de usuários, cadastrar um usuário novo, atualizar os dados do usuário, excluir os dados do usuário.

[ ] ***Classe Usuário Controller (UserController)***

[ ] @GetMapping("/user")

public ResponseEntity<Object> obterUsuarios()

[ ] sucesso: retornar 200 : Lista com os usuários.

[ ] Erro: sem tarefas: retornar 404 : Lista vazia de usuários.

[ ] @PostMapping("/user")

public ResponseEntity<Object> criarUsuario(@RequestBody @Valid UserDTO user)

[ ] Sucesso: retornar 200 : Texto “Usuário inserido com sucesso”.

[ ] Erro: retornar 409 conflict : Texto “Já existe usuário”.

[ ] @PutMapping("user/{email}")

public ResponseEntity<Object> atualizarUsuario(@PathVariable String email, @RequestBody @Valid UserDTO user)

[ ] Sucesso: retornar 200 : texto “Usuário inserido com sucesso”.

[ ] Erro: sem usuário: retornar 404 : texto “Usuário não encontrado”.

[ ] @DeleteMapping("/user/{email}")

public ResponseEntity<Object> excluirUsuario(@PathVariable String email)

[ ] Sucesso: retornar 200 : texto “usuário excluído com sucesso”.

[ ] Erro: retornar 409: texto “usuário vinculado em tarefas”.

[ ] Erro: retornar 404: texto “usuário não existe”.

***RF005: Classe controladora de tarefa:*** O sistema deverá implementar os seguintes webservices REST a fim de disponibilizar os serviços para obter a lista de tarefas, cadastrar uma tarefa nova, atualizar os dados de uma tarefa e excluir uma tarefa cadastrada.

[ ] ***Classe Tarefa Controller (TaskController)***

[ ] @GetMapping("/task")

public ResponseEntity<Object> obterTarefas()

[ ] Sucesso: retornar 200 : Lista com as tarefas.

[ ] Erro: sem tarefas: retornar 404 : Lista vazia de tarefas.

[ ] @PostMapping("/task")

public ResponseEntity<Object> criarTask(@RequestBody @Valid TaskDTO task)

[ ] Sucesso: retornar 200 : Texto “tarefa inserida com sucesso”.

[ ] Erro: retornar 404: Usuário da tarefa não encontrado

[ ] Erro: retornar 409: Usuário já possui agenda para a data informada.

[ ] Erro: retornar 400 : retornar erro.

[ ] @PutMapping("task/{id}")

public ResponseEntity<Object> atualizarTask(@PathVariable Long id, @RequestBody @Valid TaskDTO task)

[ ] Sucesso: retornar 200 : Texto “tarefa atualizar com sucesso”.

[ ] Erro: sem tarefas: retornar 404 : Texto “tarefa não encontrada”.

[ ] Erro: retornar 404: Usuário da tarefa não encontrado

[ ] @DeleteMapping("/task/{id}")

public ResponseEntity<Object> excluirTask(@PathVariable Long id)

[ ] Sucesso: retornar 200 : texto “tarefa excluída com sucesso.

[ ] Erro: sem tarefa: retornar 404 : texto “tarefa não encontrada”.

***RF006: Classe de serviço de usuário:*** O sistema deverá implementar os métodos na classe de serviços a fim de realizar as operações de obter lista de usuários cadastrados, inserir usuário, atualizar usuário e excluir usuário.

[ ] ***Classe Usuário Serviço (UserService)***

*Métodos:*

[ ] public List<UserDto> obterUsuarios()

[ ] Obter uma lista de usuários do banco de dados

[ ] converter lista de UserModel para UserDTO. percorrendo List<UserModel> adicionando em List<UserDTO>.

[ ] Retornar List<UserDTO>

[ ] public boolean inserirUsuario(UserDTO user)

[ ] validar se o usuário já existe, se já existe retornar erro

[ ] realizar a inserção de um usuário no banco de dados

[ ] public boolean atualizarUsuario(UserDTO user)

[ ] validar se o usuário já existe, se não existir retornar erro

[ ] realizar a atualização do usuário no banco de dados

[ ] public UsuarioModel obterUsuario(String email)

[ ] Obter o usuário a partir de um e-mail.

[ ] public boolean excluirUsuario(String email)

[ ] validar se o usuário já existe, se não existir retornar erro

[ ] realizar a exclusão do usuário no banco de dados

***RF007: Classe de serviço de tarefa:***

[ ] ***Classe Tarefa Serviço (TaskService)***

Métodos:

[ ] public List<TaskDTO> obterTarefas()

[ ] Obter uma lista de tarefas do banco de dados.

[ ] converter lista de TaskModel para TaskDTO. percorrendo List<TaskModel> adicionando em List<TaskDTO>.

[ ] Retornar List<TaskDTO>

[ ] public boolean inserirTarefa(TaskDTO task)

[ ] validar se o usuário existe, se não existe retornar erro

[ ] se o usuário existe, utilizar para gravar na tabela tarefa o usuário.

[ ] realizar a inserção da tarefa no banco de dados

[ ] public boolean atualizarTarefa(TaskrDTO task)

[ ] validar se o usuário existe, se não existe retornar erro

[ ] se o usuário existe, utilizar para atualizar na tabela tarefa o usuário Id.

[ ] realizar a atualização da tarefa no banco de dados

[ ] public boolean excluirTarefa(Long tarefaId)

[ ] realizar a exclusão da tarefa no banco de dados

2.2. Entregáveis

Para realizar a devida entrega do trabalho será considerado como entrega válida mediante aos seguintes artefatos:

**Projeto**: Arquivo **.zip do projeto Java** desenvolvido, este arquivo deverá conter a pasta completa do projeto, ou seja, deverá ser possível descompactar o arquivo, abrir na IDE de desenvolvimento, como por exemplo o Intellij, e deverá estar sem erros de compilação.

**Banco de dados**: Deverá ser entregue o arquivo com o **script de criação do banco de dados** juntamente com os dados inseridos de teste, ou seja, sua aplicação deverá ao menos ser executada, deverá conter **5 usuários cadastrados e 7 tarefas cadastradas**, dessa forma o script do banco de dados **deverá conter os comandos de insert** destes dados nas tabelas de usuário e tarefa.

**Requisições http**: Deverá ser entregue um arquivo contendo a exportação de uma coleção de **requisições para cada webservice** implementado com as corretas configurações:

**Relatório de entrega**: Deverá ser **realizado o preenchimento** fiel dos requisitos atendidos, atendidos parcialmente e não atendidos pelo aluno conforme modelo em anexo ao trabalho.

**NOTA: Não será aceito entrega de trabalho que não contenha todos os artefatos solicitados.**

**SENAI – Serviço de Aprendizagem Industrial**

**CURSO: Técnico de Desenvolvimento de Sistemas**

**UC: Desenvolvimento de sistemas**

**Período: 4ª Semestre**

**NOME: Nome do Aluno**

**Data: 31/03/2025**

**Relatório de entrega:**

|  |  |  |
| --- | --- | --- |
| N° | Nome da entrega | Entrega |
| 01 | API para inserir usuário  Ponto (0,5) | Entregue |
| 02 | API para atualizar usuário  Ponto (0,5) | Entregue |
| 03 | API para excluir usuário  Ponto (0,5) | Entregue |
| 04 | API para consultar usuário  Ponto (0,5) | Entregue |
| 05 | API para inserir tarefa  Ponto (1,0) | Entregue |
| 06 | API para atualizar tarefa  Ponto (1,5) | Entregue |
| 07 | API para excluir tarefa  Ponto (1,0) | Entregue |
| 08 | API para consultar tarefa  Ponto (1,5) | Entregue |
| 09 | Validação de usuário na inserção e atualização da tarefa.  Ponto (1,5) | Entregue |
| 10 | Consultas das API’s no VSCode  Ponto (0,5) | Entregue |
| 11 | Validação de data para o usuário e tarefa em conflito.  Ponto (1,0) | Entregue |
|  | Nota | 10 |