|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IDENTIFICAÇÃO  DOCUMENTO DE REQUISITO | | | | |
| **ESCOPO** | | | | |
| Implementar um WebService com arquitetura REST, respeitando os protocolos HTTP, para centralização das listas de tarefas de um cliente. | | | | |
| **PRÉ-CONDIÇÕES** | | | | |
| Não se aplica. | | | | |
| **ESPECIFICAÇÃO** | | | | |
| 1. **PROTÓTIPOS** | | | | |
| Não se aplica. | | | | |
| 1. **REQUISITOS** | | | | |
| **RF – Requisitos Funcionais/ Regras de negócios**   * 1. Deve ser criado um WebService que será consumido por um aplicativo terceiro. Este Web Service deve conter as seguintes funcionalidades:      1. Efetuar Cadastro de usuário      2. Atualizar Cadastro de usuário      3. Autenticar usuário pelo Facebook      4. Autenticar usuário por email      5. Resetar senha de usuário      6. Cadastrar lista de tarefas      7. Atualizar lista de tarefas      8. Listar lista de tarefas      9. Cadastrar tarefa      10. Remover lista de tarefas      11. Atualizar status da tarefa.   2. Para que o Web Service cadastre um usuário, deve ser enviado os campos: Nome, e-mail e Senha, através da rota especificada no RNF 2.13.1.      1. Ao receber os dados pela rota especificada, o Web Service deve verificar se existe cadastro para o email informado.         1. Se existir e-mail cadastrado, o Web Service deve retornar a mensagem: Email já existe na base de dados.         2. Se não existir e-mail cadastrado, o Web Service deve persistir o cadastro do novo usuário conforme RNF 2.14, e retornar para o aplicativo a mensagem: Usuário cadastrado.   3. Para que o Web Service atualize o cadastro, deve ser enviado o Identificador do Usuário, nome, email e Senha, através da rota especificada no RNF 2.13.2.      1. O Web Service deve atualizar os campos: Nome, email e senha, conforme especificado no RNF 2.14 e retornar a mensagem: Usuário Atualizado.      * 1. Para que o Web Service autentique o usuário pelo Facebook, deve ser enviado para o Web Service o facebook\_token, através da rota especificada no RNF 2.13.3.      1. O Web Service deve verificar se o token enviado está expirado.         1. Caso esteja expirado, o Web Service deve retornar a mensagem: Token de acesso já expirado.         2. Caso não esteja expirado, o Web Service deve realizar o cadastro do usuário com o retorno dos dados de uma consulta na API do Facebook, e retornar a mensagem: Usuário Cadastrado.   2. Para que o Web Service autentique o usuário via email e senha, deve ser enviado os campos email e a senha, através da rota especificada no RNF 2.13.4.      1. O Web Service deve verificar se o email informado existe, conforme RNF 2.14.         1. Se o email existir, o Web Service deve verificar se a senha informada está correta.            1. Se a senha estiver correta, o Web Service deve retornar o usuário que foi localizado.            2. Se a senha não estiver correta, o Web Service deve retornar a mensagem de erro: Senha não confere com o email.         2. Se o email não existir, o sistema deve retornar a mensagem: Email não existe na base.   3. Para que o Web Service resete a senha de um usuário, deve ser enviado o campo e-mail, através da rota especificada no RNF 2.13.5.      1. O WebService deve verificar se existe cadastro do email, conforme RNF 2.14.         1. Se existir cadastro do email, o Web Service deve alterar o campo senha, enviar um email para o email cadastrado informando a nova senha e não deve retornar nenhum conteúdo.         2. Se não existir o email cadastrado, o Web Service deve retornar a mensagem: E-mail não existe na base.   4. Para que o Web Service cadastre uma lista de tarefas, deve ser enviado o identificador do usuário, o nome e a cor, através da rota especificada no RNF 2.13.6.      1. O WebService deve criar uma nova lista de tarefas, conforme RNF 2.15 e 2.16, relacioná-la ao usuário e retornar a lista de tarefa criada.   5. Para que o Web Service atualize a lista de tarefa, deve ser enviado a Identificação da lista de tarefas, o nome e a cor, através da rota especificada no RNF 2.13.7.      1. O WebService deve atualizar os campos Nome e Cor para a lista de tarefa identificada, conforme RNF 2.15 e 2.16 e retornar a lista de tarefa atualizada.   6. Para que o Web Service liste uma lista de tarefas, deve ser enviado a identificação do usuário, através da rota especificada no RNF 2.13.8.      1. O WebService deve buscar as listas de tarefas relacionadas ao usuário, conforme RNF 2.15 e 2.16 e retorná-las.      * 1. Para que o Web Service cadastre uma tarefa, deve ser enviado a Identificação da lista de tarefa, a descrição e o status, através da rota especificada no RNF 2.13.9.      1. O WebService deve buscar a lista de tarefas identificada, conforme RNF 2.15 e 2.16, cadastrar e vincular a nova tarefa nessa lista, conforme RNF 2.17 e retornar a tarefa criada.   2. Para que o Web Service remova a lista de tarefas, deve ser enviado o código de identificação da lista de tarefas e utilizado a rota especificada no RNF 2.13.10.      1. Ao receber o dado pela rota especificada, o WebService deve encontrar a lista de tarefa do usuário, conforme RNF 2.15 e 2.16, removê-la e não retornar nenhuma informação.   3. Para que o Web Service atualize o status da tarefa, deve ser enviado a Identificação da tarefa e o status (0: a fazer, 1: feito), através da rota especificada no RNF 2.13.11.      1. O WebService deve encontrar a tarefa identificada, atualizar o status da tarefa, conforme RNF 2.17 e retornar a tarefa atualizada.   **RNF – Requisitos Não Funcionais**   * 1. Rota REST:      1. Cadastrar Usuário: POST em /users/add      2. Atualizar Cadastro: PUT/PATCH em /users/update      3. Autenticar pelo Facebook: POST em /users/authenticate      4. Autenticar via email e senha: POST em /users/authenticate      5. Resetar Senha: POST em /users/forgot\_password      6. Cadastrar Lista de Tarefas: POST em /task\_lists/add      7. Atualizar lista de tarefas: PUT/PATCH em /task\_lists/update      8. Listar lista de tarefas: GET em /task\_lists/list/:user\_id      9. Cadastrar tarefa: POST em /task\_lists/:task\_list\_id/tasks/add      10. Remover lista de tarefas: DELETE em /task\_lists/remove:task\_list\_id      11. Atualizar status da tarefa: PUT/PATCH em /tasks/update   2. Os dados do usuário devem ser inclusos na tabela user com nos campos: name, email, senha e token\_facebook.   3. As cores das tarefas, devem ser inclusas na tabela color, no campo name\_color.   4. Os dados da lista de tarefa, devem ser inclusos na tabela task\_list nos campos: id\_user, name\_task\_list, id\_color.   5. Os dados da tarefa, devem ser inclusos na tabela task nos campos: id\_task\_list, task\_description, status\_task. | | | | |
| **RESTRIÇÕES (FORA DO ESCOPO)** | | | | |
| Não se aplica. | | | | |
| **ANEXOS - DER** | | | | |
| * 1. DER do Banco de Dados   banco.png | | | | |
| **ANEXOS - Scripts MySQL** | | | | |
| CREATE TABLE `user` (  `id\_user` INT NOT NULL,  `name\_user` varchar(256) NOT NULL,  `email\_user` varchar(128) UNIQUE,  `senha\_user` varchar(256),  `token\_facebook` varchar(256),  PRIMARY KEY (`id\_user`)  );  CREATE TABLE `task\_list` (  `id\_task\_list` INT NOT NULL UNIQUE,  `id\_user` INT NOT NULL,  `name\_task\_list` varchar(128) NOT NULL,  `id\_color` INT NOT NULL,  PRIMARY KEY (`id\_task\_list`)  );  CREATE TABLE `task` (  `id\_task` INT NOT NULL UNIQUE,  `id\_task\_list` INT NOT NULL,  `task\_description` varchar(128) NOT NULL,  `status\_task` BOOLEAN NOT NULL DEFAULT '0',  PRIMARY KEY (`id\_task`)  );  CREATE TABLE `color` (  `id\_color` INT NOT NULL UNIQUE,  `name\_color` varchar(128) NOT NULL UNIQUE,  PRIMARY KEY (`id\_color`)  );  ALTER TABLE `task\_list` DROP FOREIGN KEY `task\_list\_fk0`;  ALTER TABLE `task\_list` DROP FOREIGN KEY `task\_list\_fk1`;  ALTER TABLE `task` DROP FOREIGN KEY `task\_fk0`;  DROP TABLE IF EXISTS `user`;  DROP TABLE IF EXISTS `task\_list`;  DROP TABLE IF EXISTS `task`;  DROP TABLE IF EXISTS `color`;  ALTER TABLE `task\_list` ADD CONSTRAINT `task\_list\_fk0` FOREIGN KEY (`id\_user`) REFERENCES `user`(`id\_user`);  ALTER TABLE `task\_list` ADD CONSTRAINT `task\_list\_fk1` FOREIGN KEY (`id\_color`) REFERENCES `color`(`id\_color`);  ALTER TABLE `task` ADD CONSTRAINT `task\_fk0` FOREIGN KEY (`id\_task\_list`) REFERENCES `task\_list`(`id\_task\_list`); | | | | |
| **ANEXOS - Scripts Postgres** | | | | |
| CREATE TABLE "user" (  "id\_user" serial NOT NULL,  "name\_user" varchar(256) NOT NULL,  "email\_user" varchar(128) UNIQUE,  "senha\_user" varchar(256),  "token\_facebook" varchar(256),  CONSTRAINT user\_pk PRIMARY KEY ("id\_user")  ) WITH (  OIDS=FALSE  );  CREATE TABLE "task\_list" (  "id\_task\_list" serial NOT NULL UNIQUE,  "id\_user" integer NOT NULL,  "name\_task\_list" varchar(128) NOT NULL,  "id\_color" integer NOT NULL,  CONSTRAINT task\_list\_pk PRIMARY KEY ("id\_task\_list")  ) WITH (  OIDS=FALSE  );  CREATE TABLE "task" (  "id\_task" serial NOT NULL UNIQUE,  "id\_task\_list" integer NOT NULL,  "task\_description" varchar(128) NOT NULL,  "status\_task" BOOLEAN NOT NULL DEFAULT '0',  CONSTRAINT task\_pk PRIMARY KEY ("id\_task")  ) WITH (  OIDS=FALSE  );  CREATE TABLE "color" (  "id\_color" serial NOT NULL UNIQUE,  "name\_color" varchar(128) NOT NULL UNIQUE,  CONSTRAINT color\_pk PRIMARY KEY ("id\_color")  ) WITH (  OIDS=FALSE  );  ALTER TABLE "task\_list" DROP CONSTRAINT IF EXISTS "task\_list\_fk0";  ALTER TABLE "task\_list" DROP CONSTRAINT IF EXISTS "task\_list\_fk1";  ALTER TABLE "task" DROP CONSTRAINT IF EXISTS "task\_fk0";  DROP TABLE IF EXISTS "user";  DROP TABLE IF EXISTS "task\_list";  DROP TABLE IF EXISTS "task";  DROP TABLE IF EXISTS "color";  ALTER TABLE "task\_list" ADD CONSTRAINT "task\_list\_fk0" FOREIGN KEY ("id\_user") REFERENCES "user"("id\_user");  ALTER TABLE "task\_list" ADD CONSTRAINT "task\_list\_fk1" FOREIGN KEY ("id\_color") REFERENCES "color"("id\_color");  ALTER TABLE "task" ADD CONSTRAINT "task\_fk0" FOREIGN KEY ("id\_task\_list") REFERENCES "task\_list"("id\_task\_list"); | | | | |
| **ESPECIFICAÇÃO**  HISTÓRICO DE REVISÕES | | | | |
| **Data** | **Versão** | **Tópico** | **Revisão** | **Autor** |
| 15/09/2016 | 1.0 |  | Prototipação | Keila.Soares |
| 16/09/2016 | 1.1 |  | Inclusão dos rf 2.1 ao 2.6 | Keila.Soares |
| 19/09/2016 | 1.2 |  | Alteração dos rf2.1 ao 2.6, exclusão protótipos | Keila.Soares |
| 20/09/2016 | 1.3 |  | Inclusão dos rf 2.7 ao 2.12 e dos rnf 2.13 e 2.14 | Keila.Soares |
| 21/09/2016 | 1.4 |  | Inclusão do DER e dos scripts | Keila.Soares |
|  |  |  |  |  |