**wEA975 - Laboratório de Engenharia de Software - Turma L - 2019/1s**

**Prof. Dr. Eleri Cardozo**



**Relatório Final taskManager**

Grupo 2

Henrique Roberto da Cunha Junior - 174638

Leonardo Rodrigues Marques - 178610

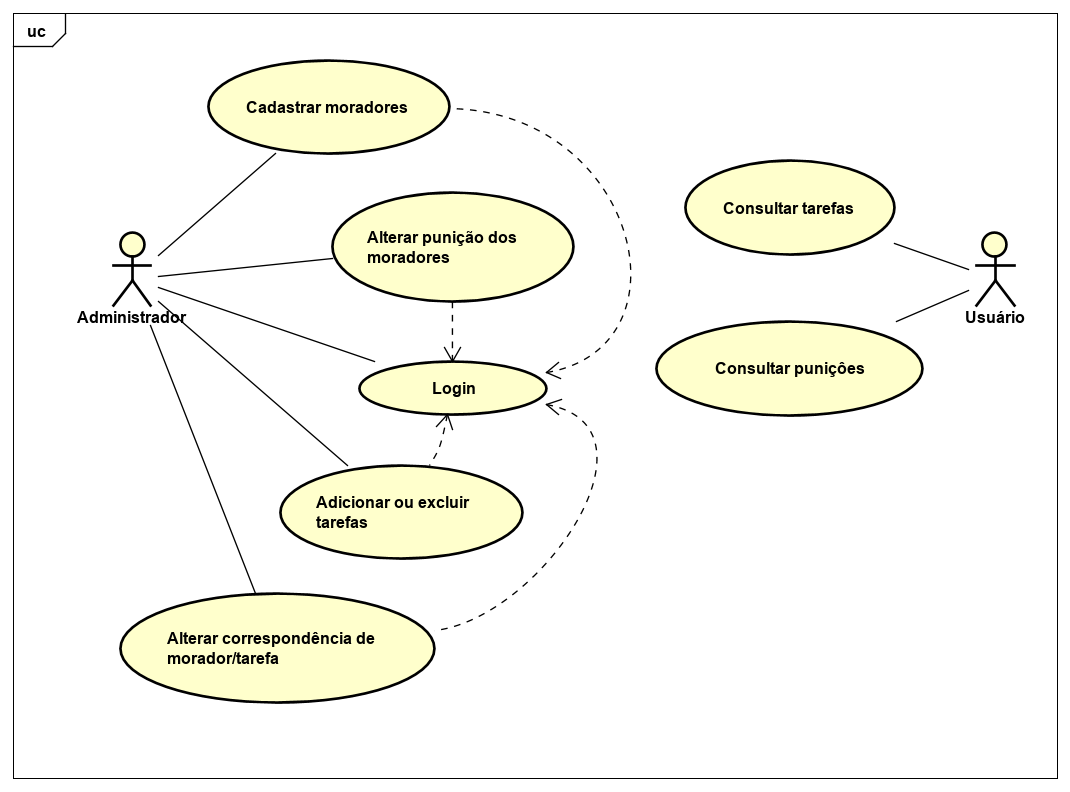
1. **Descrição do Sistema**

Nossa proposta é projetar um sistema que facilite a organização de tarefas em uma república universitária. Conhecendo o ambiente e já sabendo como funciona em boa parte das repúblicas, o objetivo foi automatizar o processo.

O sistema consiste em 2 tipos de usuários: um administrador, responsável por alimentar a base de dados, e os usuários padrões, que podem consultar os dados.

* 1. **Funções do produto**
     + Adicionar os dados dos moradores como nome, ra, apelido, ou o que for necessário ou desejado.
     + Adicionar as tarefas que precisam ser feitas, por exemplo: tirar o lixo, passear com cachorro, ou o que for necessário.
     + Adicionar punições para cada morador, caso deixem de fazer a tarefa.
     + Modificar para semana de punição, onde um morador deverá fazer todas as tarefas da semana, devido a ter deixado muitas tarefas sem fazer.
     + Saber as tarefas destinadas aos moradores durante a semana.
     + Consultar a tabela de punições.
  2. **Restrições gerais**
     + Os usuários só poderão fazer consultas das tarefas destinadas e punições sofridas.
     + O administrador é o único responsável por inserir dados dos usuários e destinar as tarefas para eles.

1. **Requisitos do Sistema**
   1. **Requisitos de Usuário**
      * O usuário deve ser capaz de consultar o seu número de punições e de outros moradores.
      * O usuário deve ser capaz de consultar as tarefas destinadas a ele na semana.
   2. **Requisitos de Sistema**
      * O sistema deve adicionar tarefas a moradores existentes.
      * Armazenar o número de punições de cada morador e tarefas para consulta.
   3. **Requisitos não Funcionais**
      * O sistema deve seguir o framework Mean, MVC.
      * O sistema deve se comunicar com um banco de dados MongoDB.
      * O sistema deve utilizar Angularjs e Node.js.
      * O sistema deve ser acessível em celulares mobile.
   4. **Requisitos Funcionais**
      * O administrador deve ser capaz de cadastrar todos os moradores, inserindo os dados como nome (String), ID(Integer), Curso(String), Punição(Integer).
      * O administrador deve ser ser capaz de adicionar/somar pontos para um determinado morador, dados seu nome ou ID.
      * O administrador deve ser capaz de adicionar todas as tarefas possíveis.
      * O administrador deve ser capaz de alterar a tarefa que corresponde a determinado morador para outro morador.
2. **Casos de Uso**



**Figura 1: Casos de uso para o sistema**

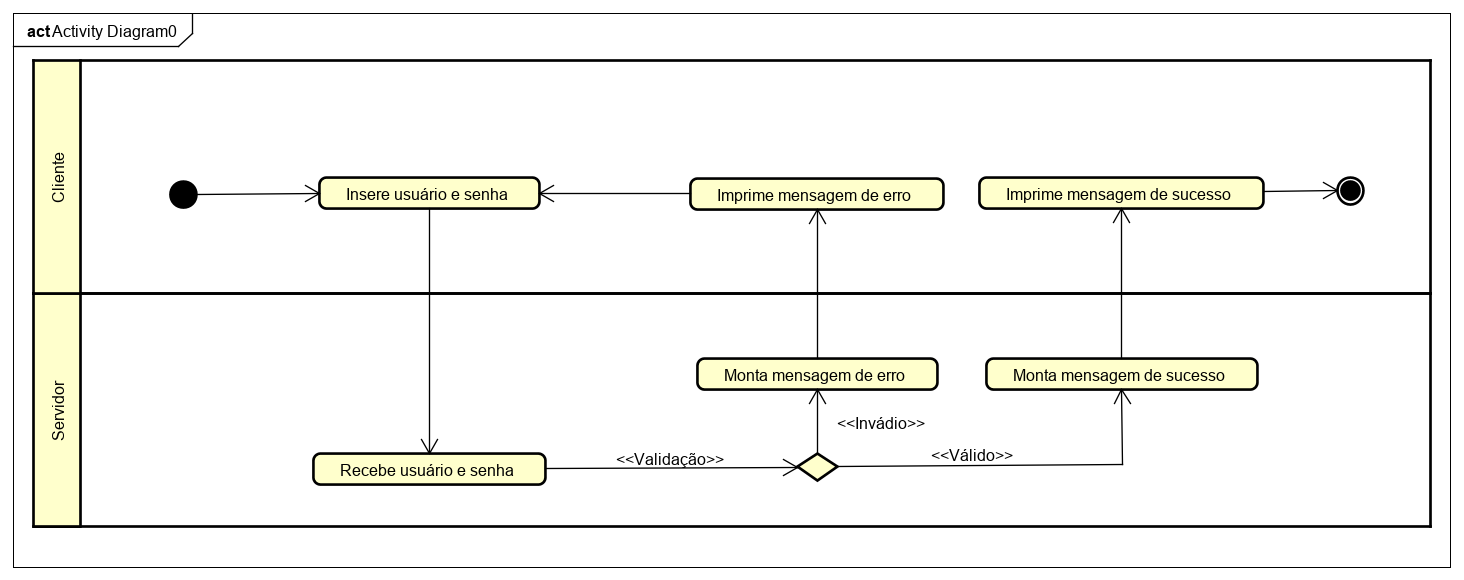
* 1. **Autenticar Administrador:**
* Atores envolvidos:
  + Primário: Administrador
* Fluxo de eventos:
  + Principal: O administrador insere seu usuário e senha.
  + Alternativo: Inserir um usuário que não existe no banco

Inserir uma senha incorreta para um usuário

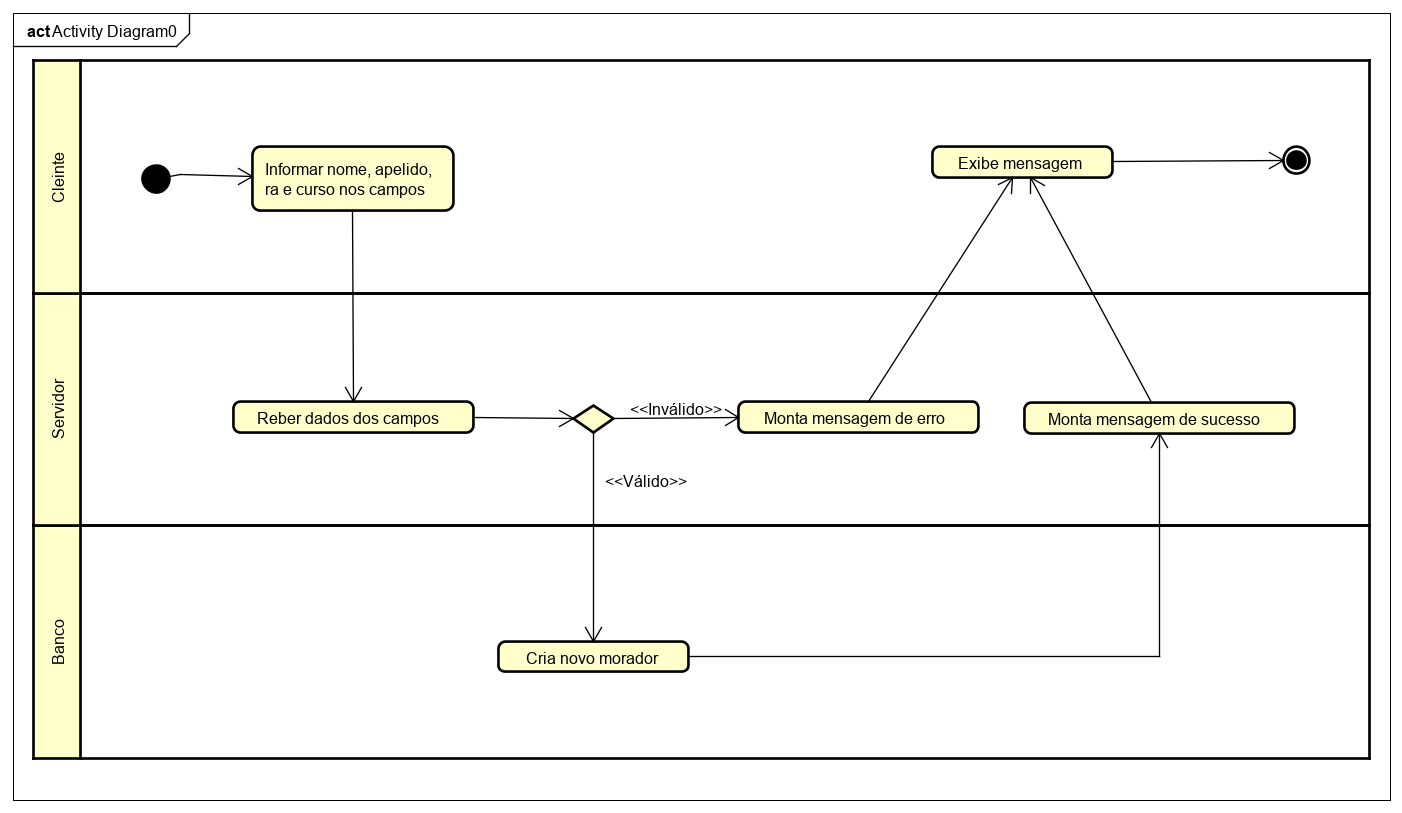
* + Exceção: Impossibilidade de se comunicar com o banco para inserir novo usuário administrador
* Pós-condições: Administrador autenticado com sucesso
  1. **Cadastrar moradores:**
* Atores envolvidos:
  + Primário: Administrador
  + Secundário: Moradores
* Pré-condições:
  + Administrador (Ator primário) deve estar autenticado.
* Fluxo de eventos:
  + Principal: O administrador cadastra um novo usuário com Nome, Ra, Curso que devem ser diferentes de vazio. E coluna punição, inteiro começando em 0.
  + Alternativo: Inserir algum Ra, diferente de números.
  + Exceção: Impossibilidade de se comunicar com o banco para inserir o morador.
* Pós-condições: Usuário cadastrado com sucesso, e sua coluna punição com valor 0.
  1. **Alterar punições dos Moradores:**
* Atores envolvidos:
  + Primário: Administrador
  + Secundário: Moradores
* Pré-condições:
  + Administrador (Ator primário) deve estar autenticado.
  + Devem existir pelo menos um Morador (Ator secundário) cadastrado.
* Fluxo de eventos:
  + Principal: O administrador incrementa a coluna de punições de um determinado morador.
  + Exceção: Impossibilidade de se comunicar com o banco para atualizar a tabela.
  + Pós-condições: Dados do morador atualizado com sucesso.
  1. **Alterar ou excluir tarefas:**
* Atores envolvidos:
  + Primário: Administrador
* Pré-condições:
  + Administrador (Ator primário) deve estar autenticado.
* Fluxo de eventos:
  + Principal: O administrador cadastra um novo tipo de tarefa na tabela Tarefas, com Nome, Id.
* Exceção: Impossibilidade de se comunicar com o banco para inserir a nova linha na tabela
* Pós-condições: Linha inserida na tabela e tabela atualizada com sucesso.
  1. **Alterar correspondência de morador/tarefa:**
* Atores envolvidos:
  + Primário: Administrador
  + Secundário: Moradores
* Pré-condições:
  + Administrador (Ator primário) deve estar autenticado.
  + Deve haver uma relação entre morador e tarefa pré-estabelecida.
* Fluxo de eventos:
  + Principal: O administrador seleciona o morador e altera a tarefa correspondente a ele.
  + Exceção: Impossibilidade de se comunicar com o banco para alterar a correspondência entre morador e tarefa.
* Pós-condições:
  + Alteração de tarefa para determinado morador realizada com sucesso.
  1. **Consultar punições:**
* Atores envolvidos:
  + Primário: Morador (usuário comum)
  + Secundário: Moradores
* Pré-condições:
  + Deve haver pelo menos um morador para consultarmos as punições
* Fluxo de eventos:
  + Principal: O usuário solicita uma tabela informando todos os moradores com suas respectivas punições (número)
  + Alternativo: Sem moradores a página viria em branco.
  + Exceção: Impossibilidade de se comunicar com o banco para consultar as punições.

Tabela de moradores vazia causa uma exceção, pois não seria possível acessar nenhum.

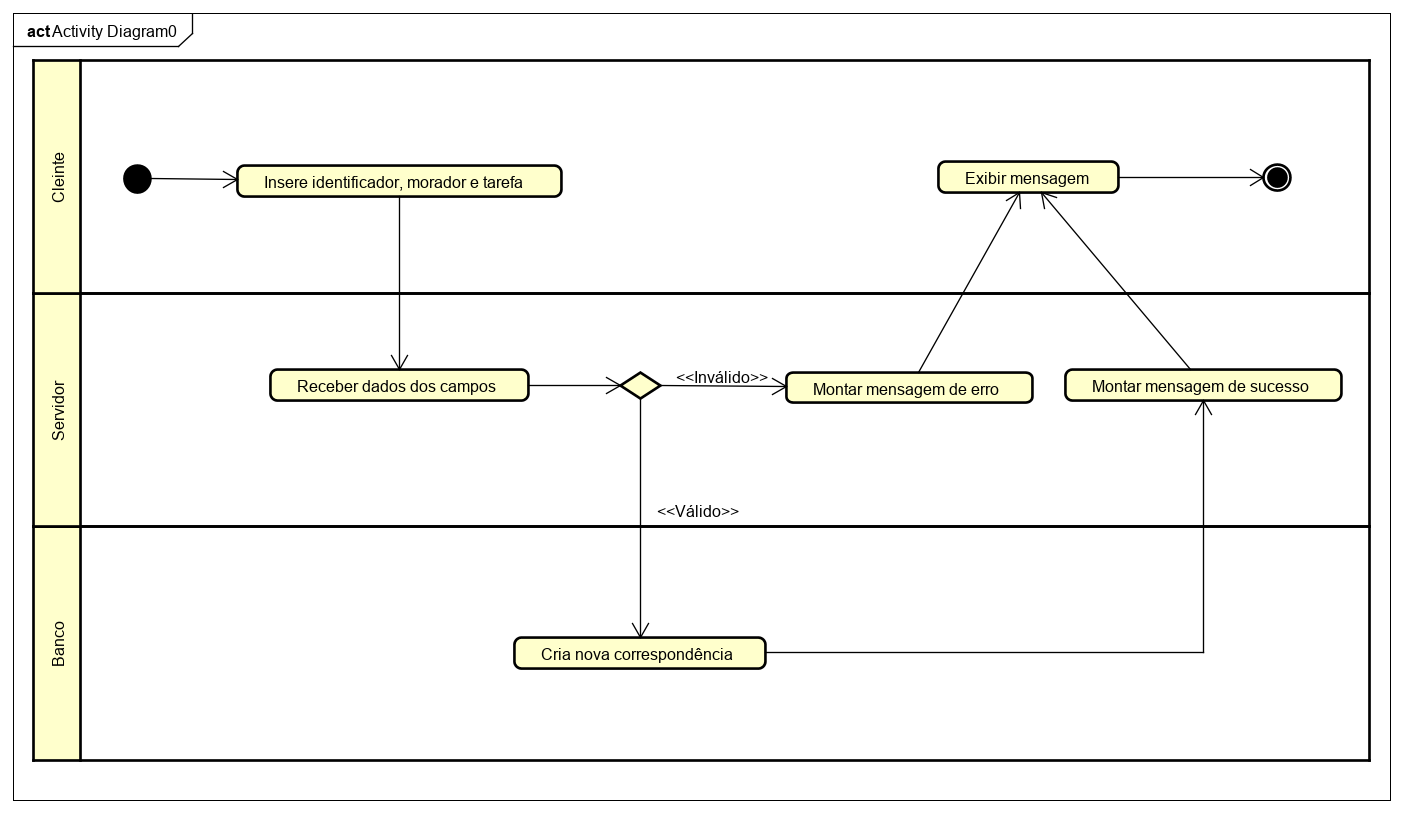
* Pós-condições: A tabela de punições é exibida com sucesso para o usuário.
  1. **Consultar tarefas destinadas:**
* Atores envolvidos:
  + Primário: Morador (usuário comum)
* Pré-condições:
  + Deve haver pelo menos um morador e pelo menos alguma tarefa para esse morador
* Fluxo de eventos:
  + Principal: O usuário solicita uma tabela que informa todos os moradores e suas respectivas tarefas.
  + Alternativo: Se não houver moradores com tarefas respectivas, a tabela viria vazia
  + Exceção: Usuário vazio causaria um null pointer, pois tentaremos acessar as tarefas de um morador que não existe.
* Pós-condições: A tabela de correspondências é exibida com sucesso.

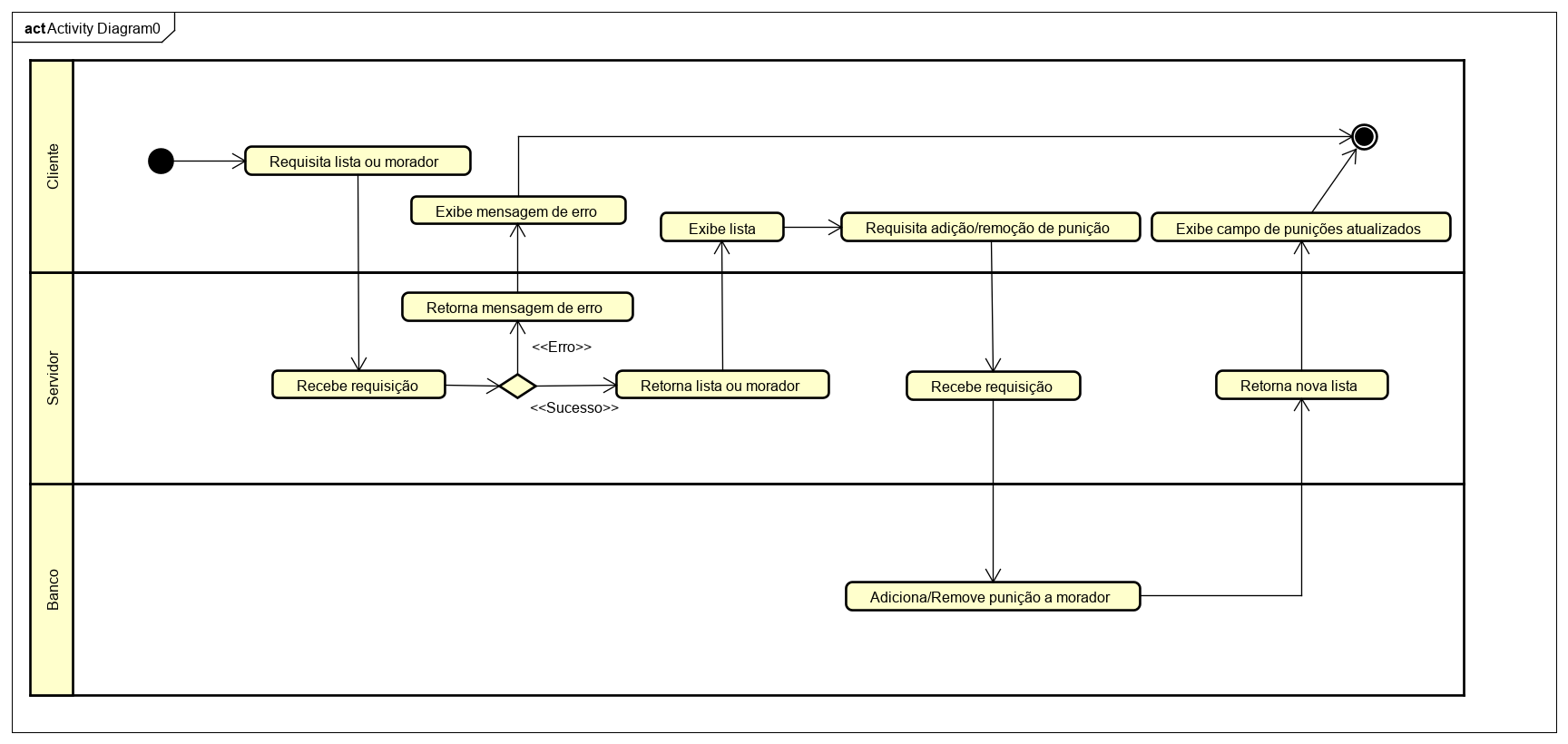
1. **Diagrama de Atividades**
   1. 

**Figura 2: Diagrama de atividade Login**

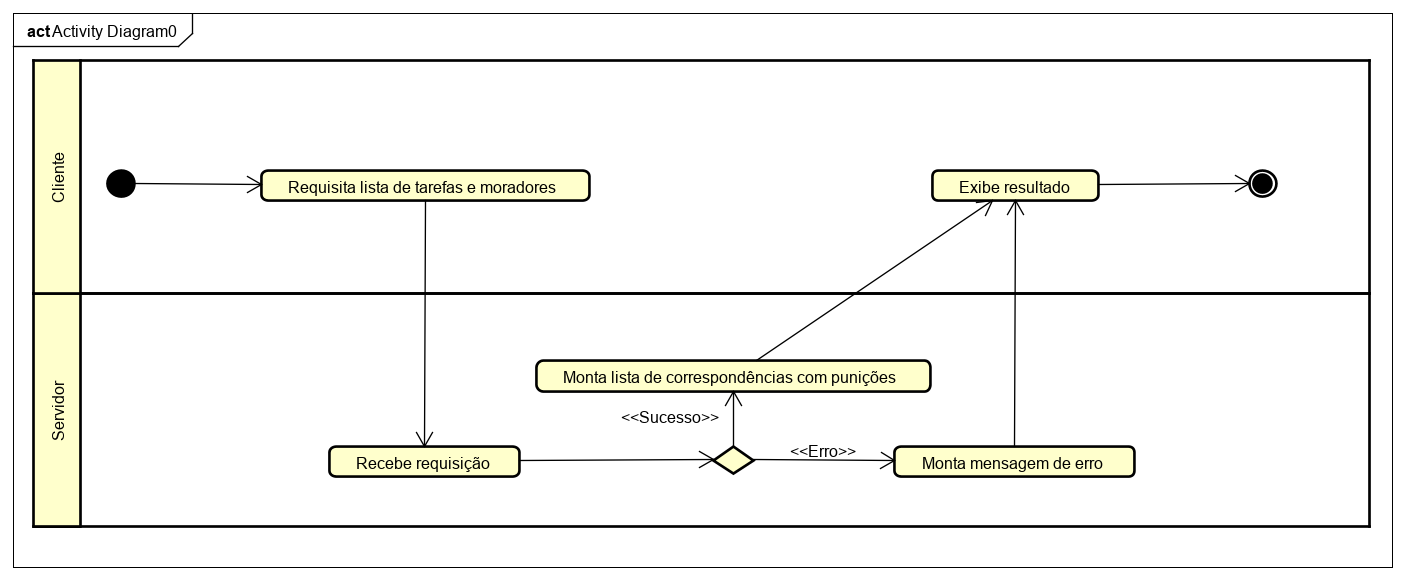
* 1. 

**Figura 3: Diagrama de atividade Cadastro Morador**

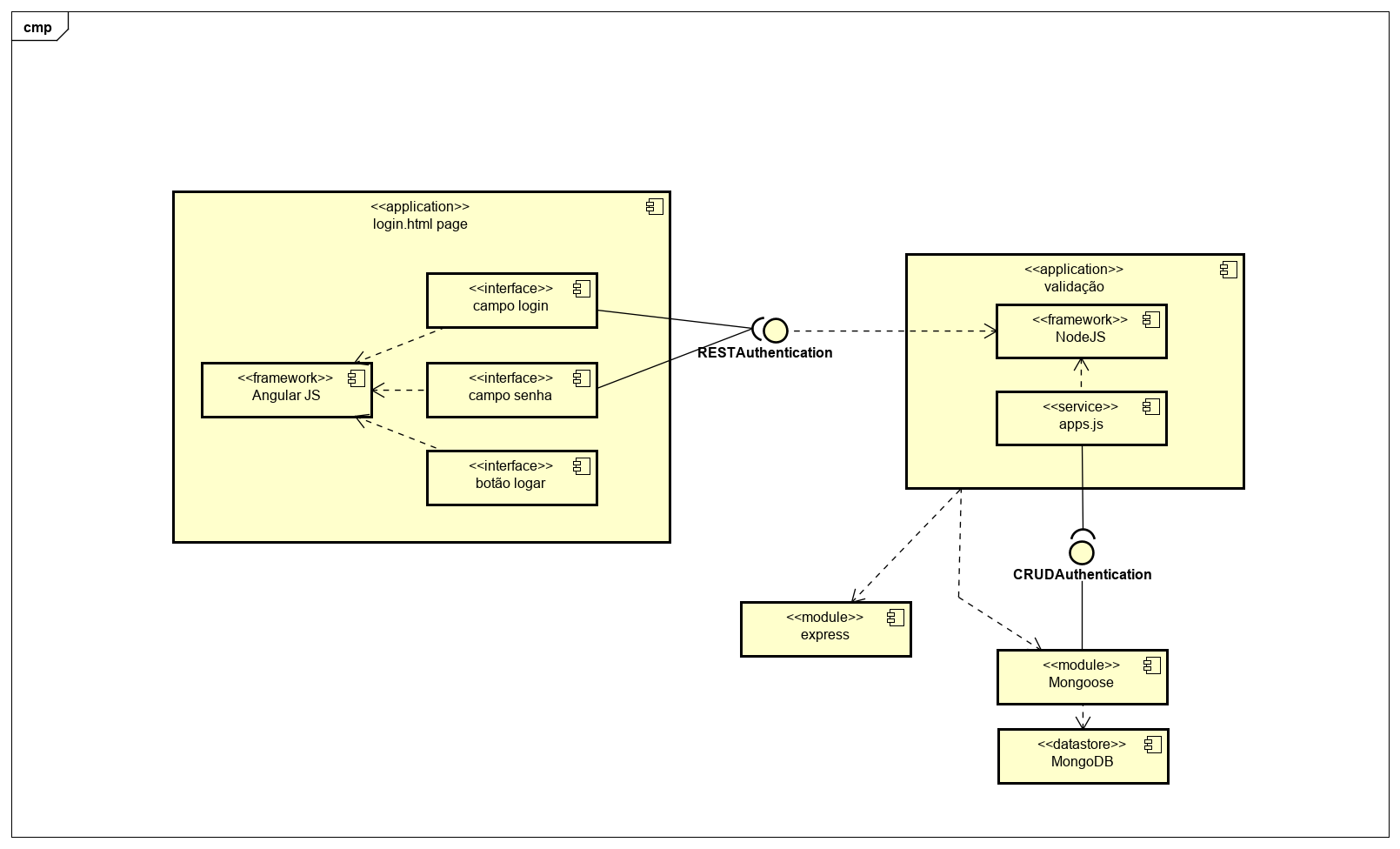
**Figura 4: Diagrama de atividade Controle de tarefas**

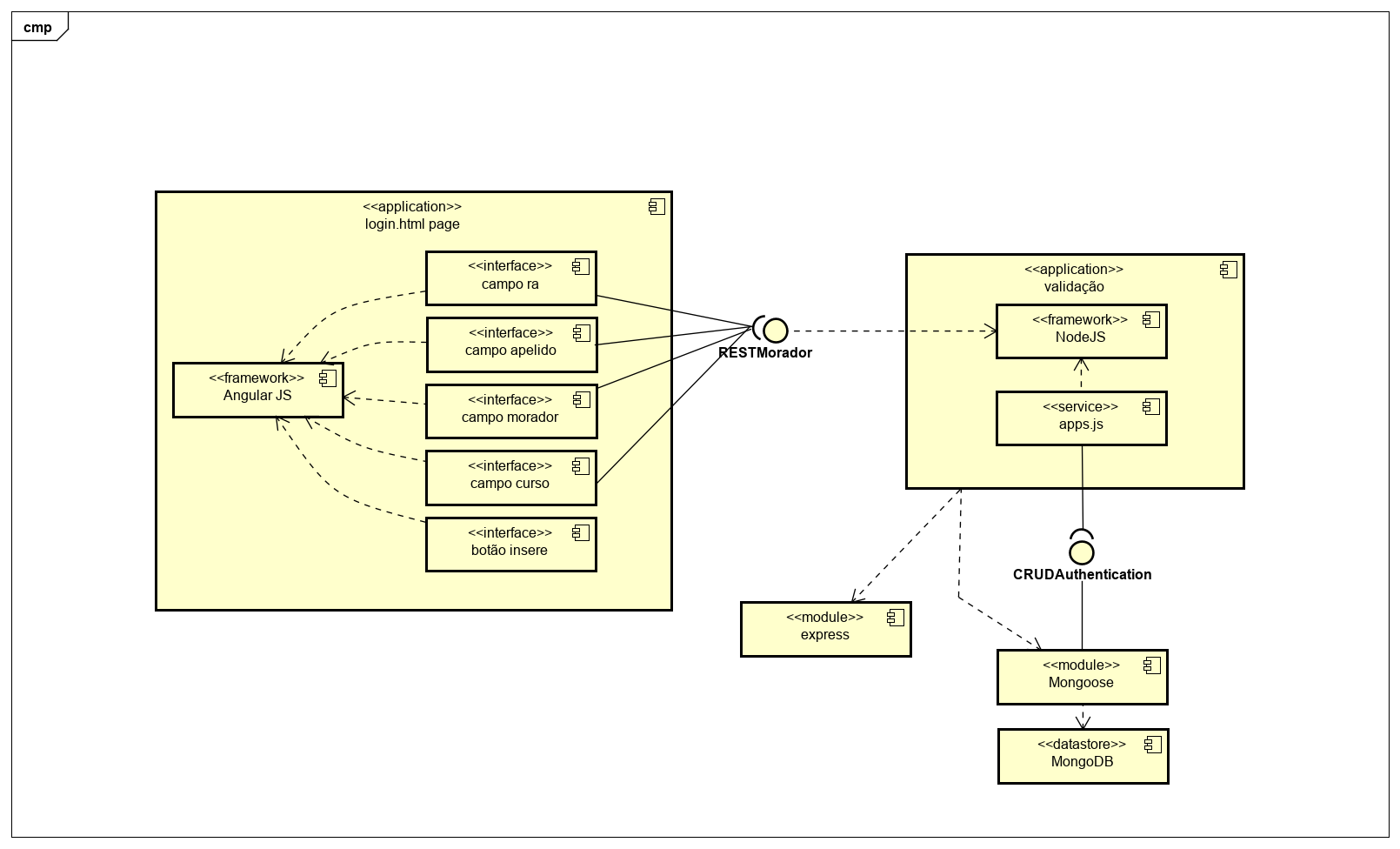
* 1. 

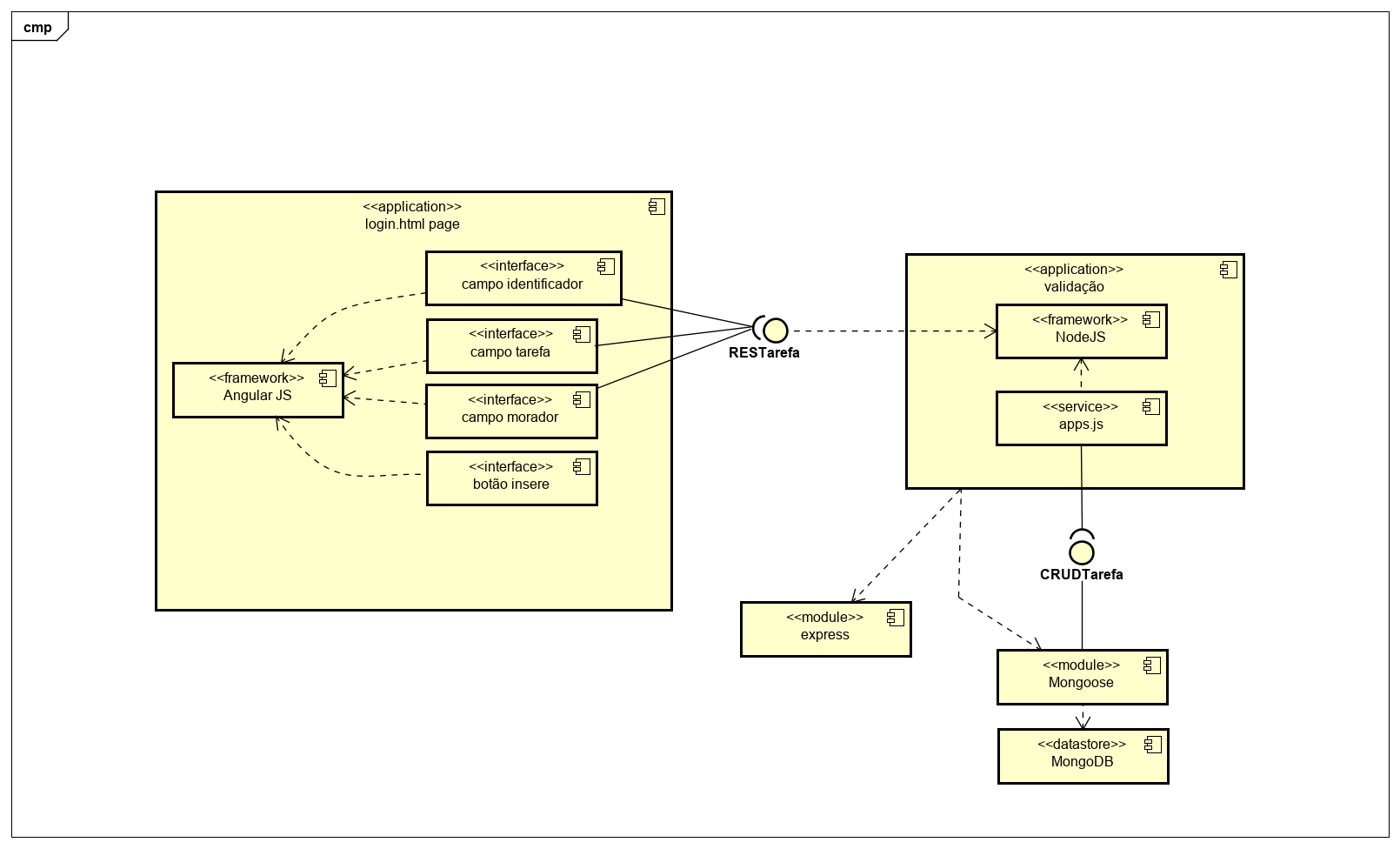
**Figura 5: Diagrama de atividade Administrar punições**

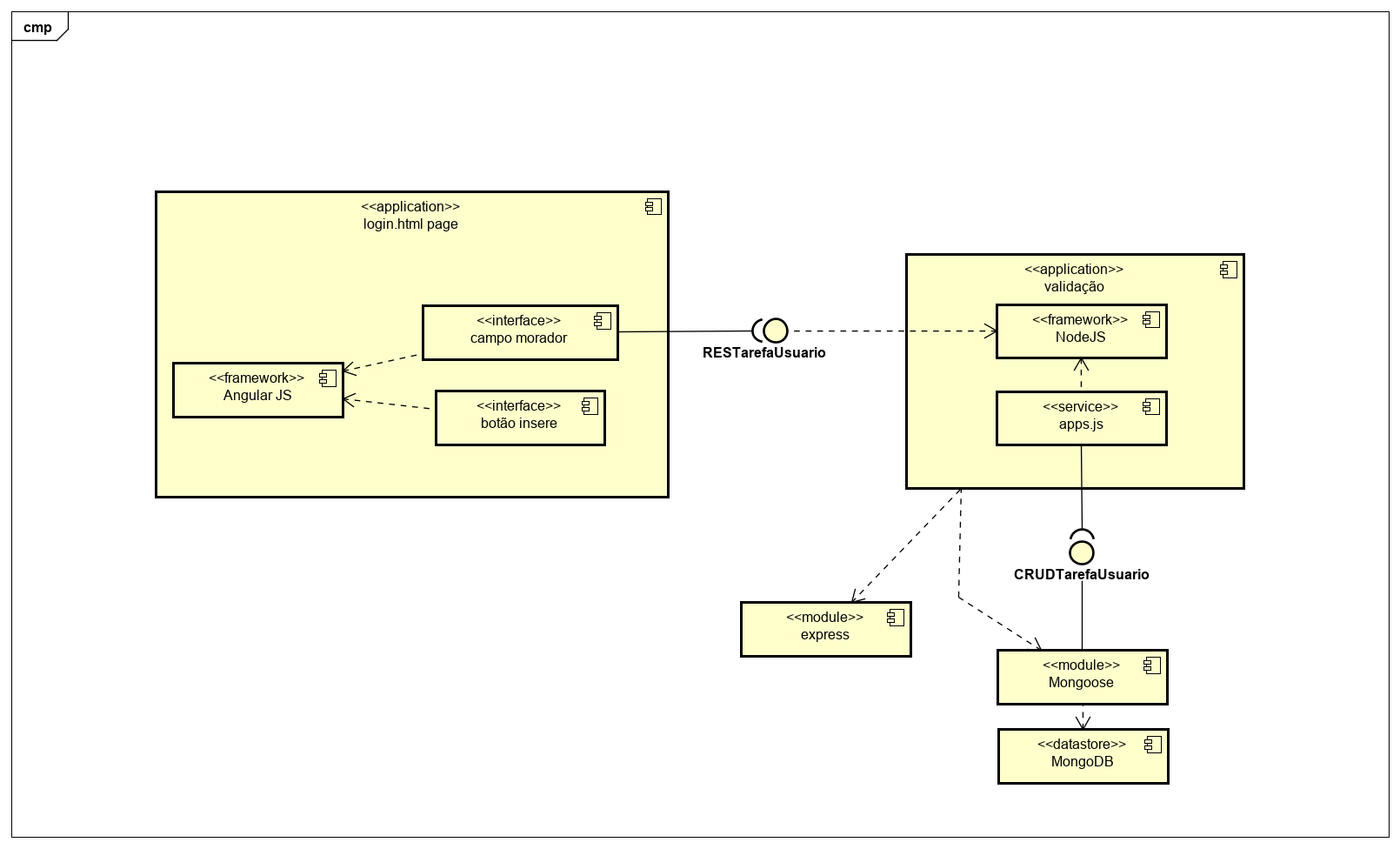
**Figura 6: Diagrama de atividades Consultar tarefas** 

1. **Componentes**

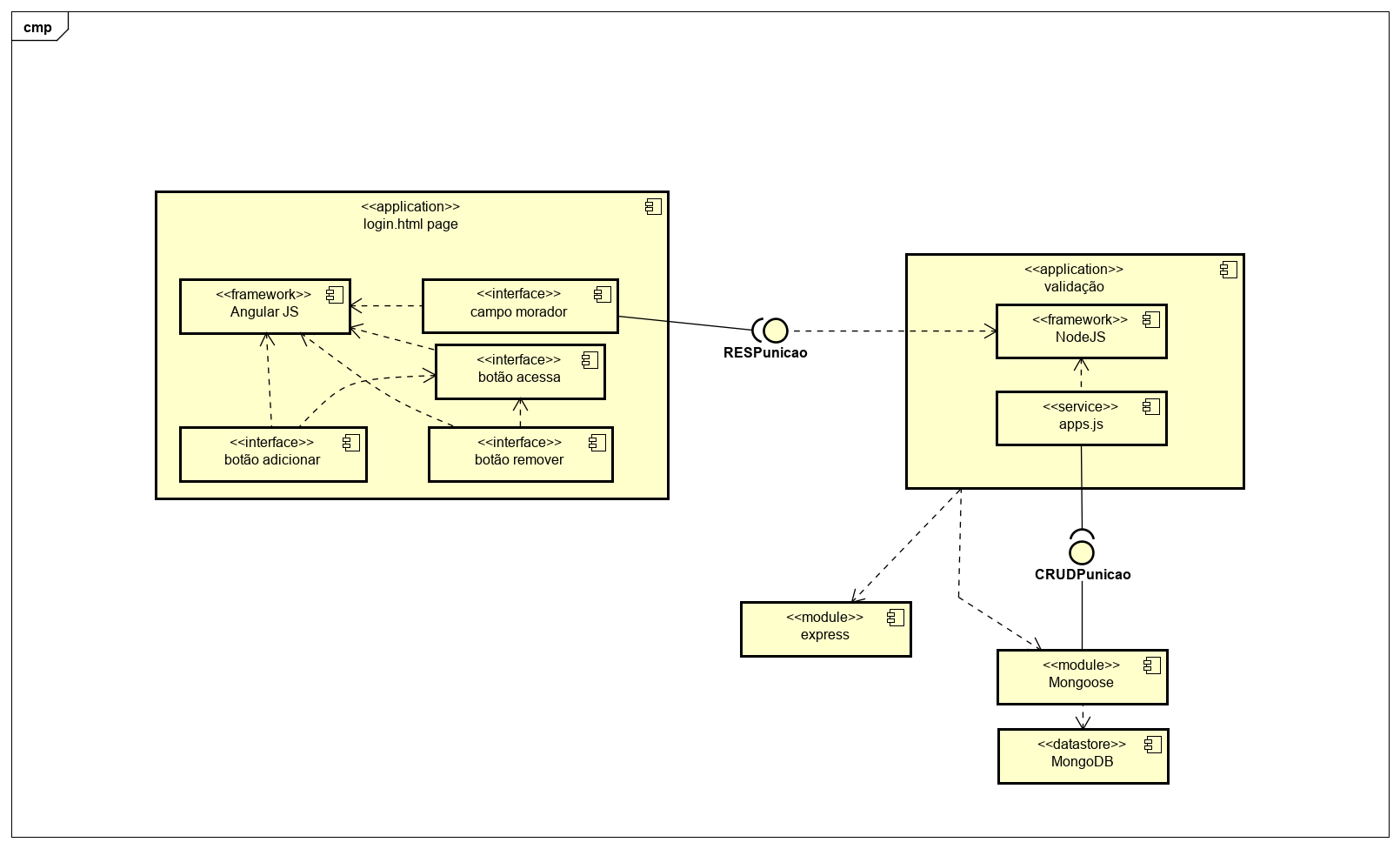
**Figura 7: Diagrama de componentes login**

**Figura 7: Diagrama de componentes Cadastrar Morador**

**Figura 7: Diagrama de componentes Adicionar tarefa**



**Figura 8: Diagrama de componente Consultar punições e tarefas**

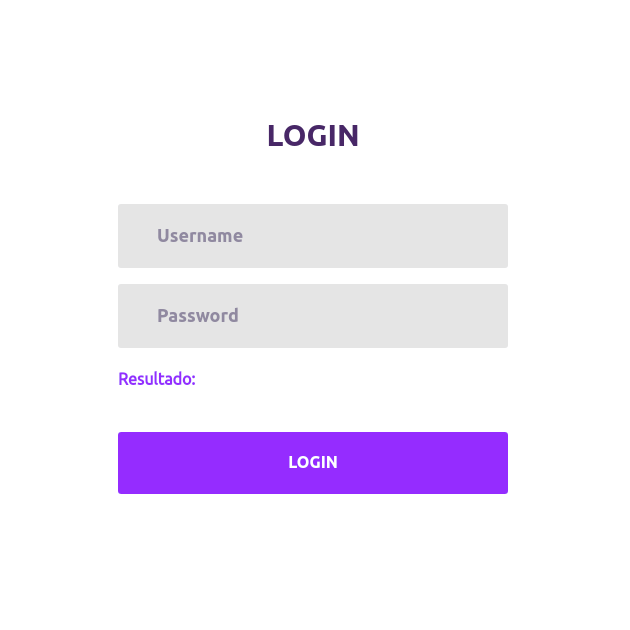


**Figura 9: Diagrama de componente Administrador punições**

1. **Implementação do Sistema**
   1. **Camada do Cliente**

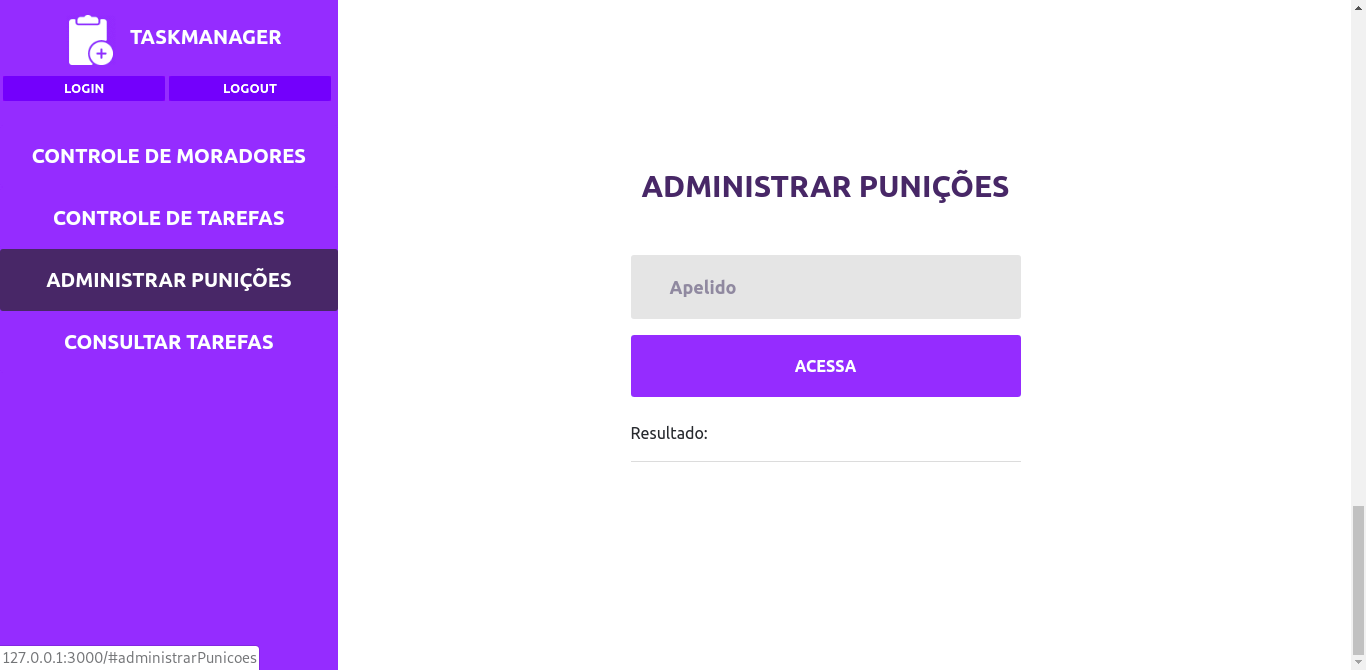
A camada do cliente é onde ocorre a interação com o usuário através de sistemas interativos e visuais onde ocorrem os preenchimentos de valores, informações de erros ou sucessos das requisições HTTP.

Utilizamos o framework AngularJS com um total de 5 páginas diferentes com os casos de uso explicados na documentação. Seguem abaixo as telas:

**Figura 8: Tela de Login**



**Figura 9: Tela de Controle de Tarefas**



**Figura 10 : Tela de Administrar Punições**



**Figura 11 : Tela de Cadastro de Moradores**

**Figura 12: Tela de Consulta de Tarefas e Punições**

* 1. **Camada do Servidor**

No servidor, implementamos os componentes dos serviços HTTP. No servidor temos duas collections no MongoDB : user, de usuários, e task, para as tarefas. Ambas representadas no formato de json. Além disso, utilizou-se o framework Node JS Express para a implementação dos componentes de serviço. Como MongoDB não é um banco relacional, criamos um relacionamento através da tabela task, onde temos um campo "apelido" que referencia o apelido de um morador.

Seguem a documentação com os métodos disponíveis:

Tabela 1: Tabela com os recursos e métodos HTTP disponíveis

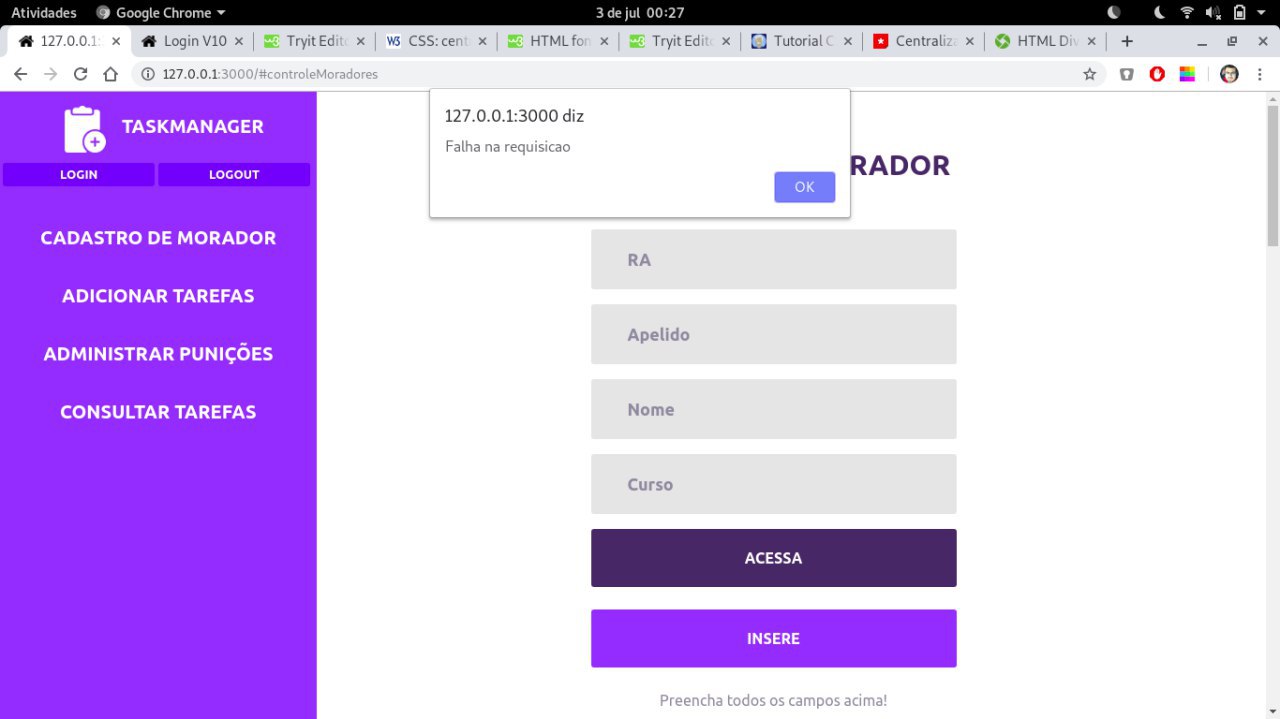
|  |  |
| --- | --- |
| Recurso | Métodos Disponíveis |
| User | POST, PUT,DELETE,GET |
| Task | POST,PUT,DELETE,GET |

* 1. **Camada de retaguarda**

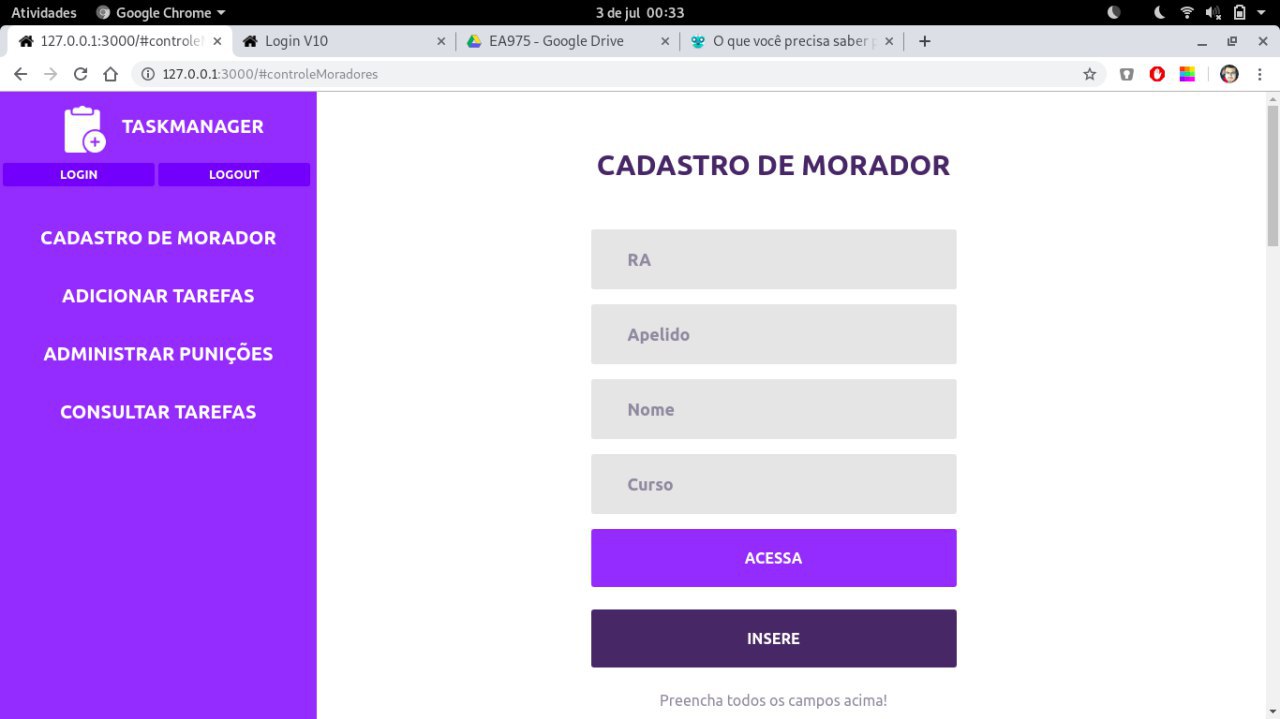
Na camada de retaguarda é onde estão guardadas as estruturas que recebem as requests disponíveis no formato de Json a qual identifica em qual recurso as operações devem ser realizadas. Utilizamos um framework com MongoDB e mongoose para essa camada de persistência dos dados.

1. **Testes**

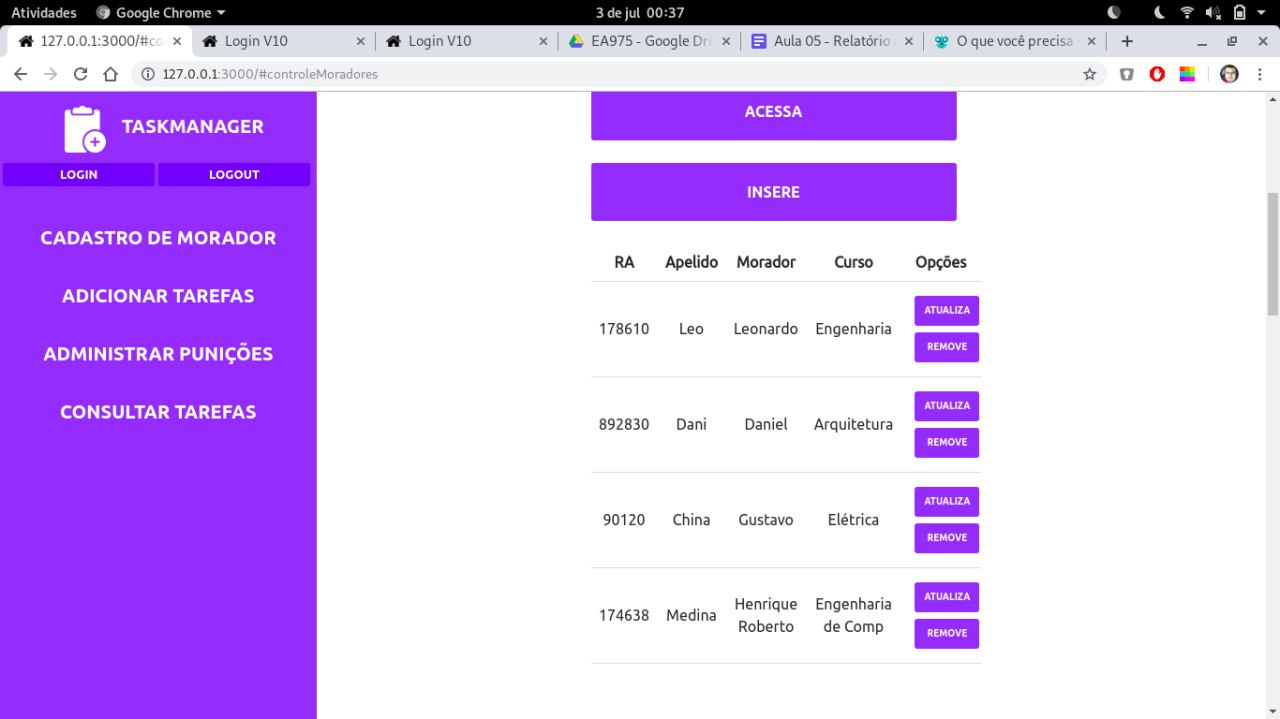
**Teste 1:** Tentamos fazer uma operação que necessite de autenticação sem autenticação. Recebemos a mensagem de falha.



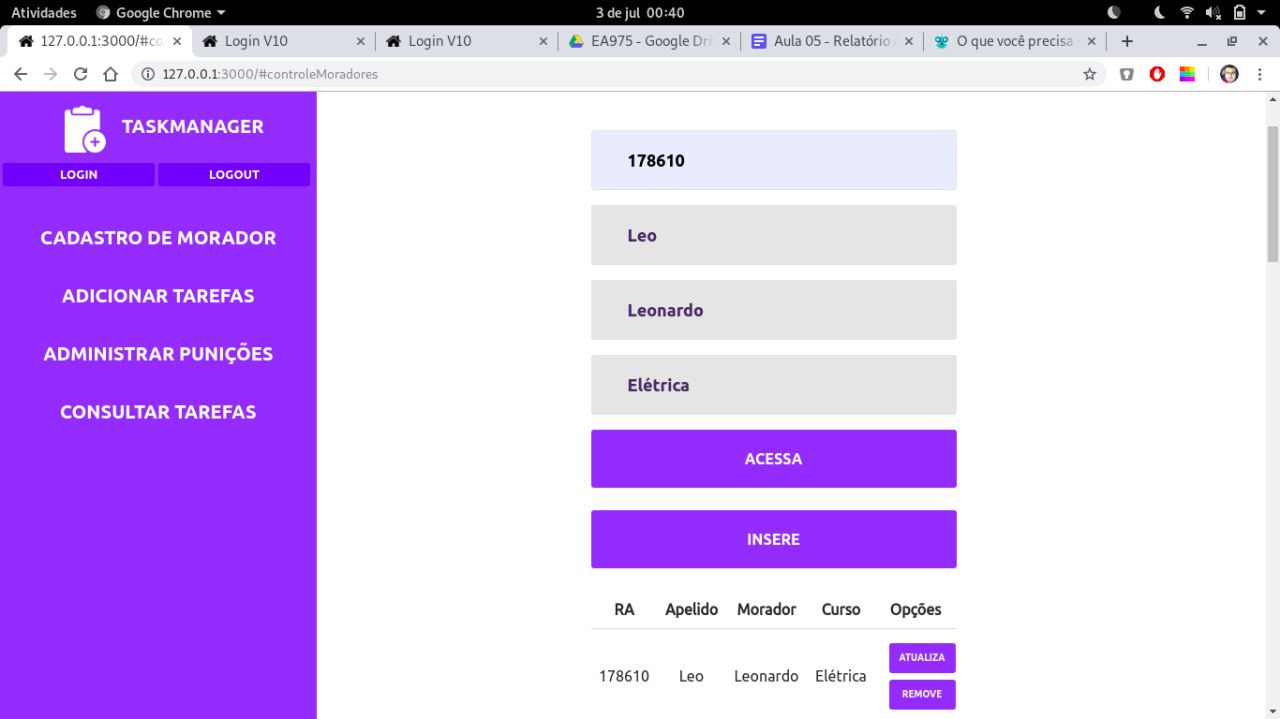
**Teste 2:** Tentar inserir sem estar logado. Falha na requisição.



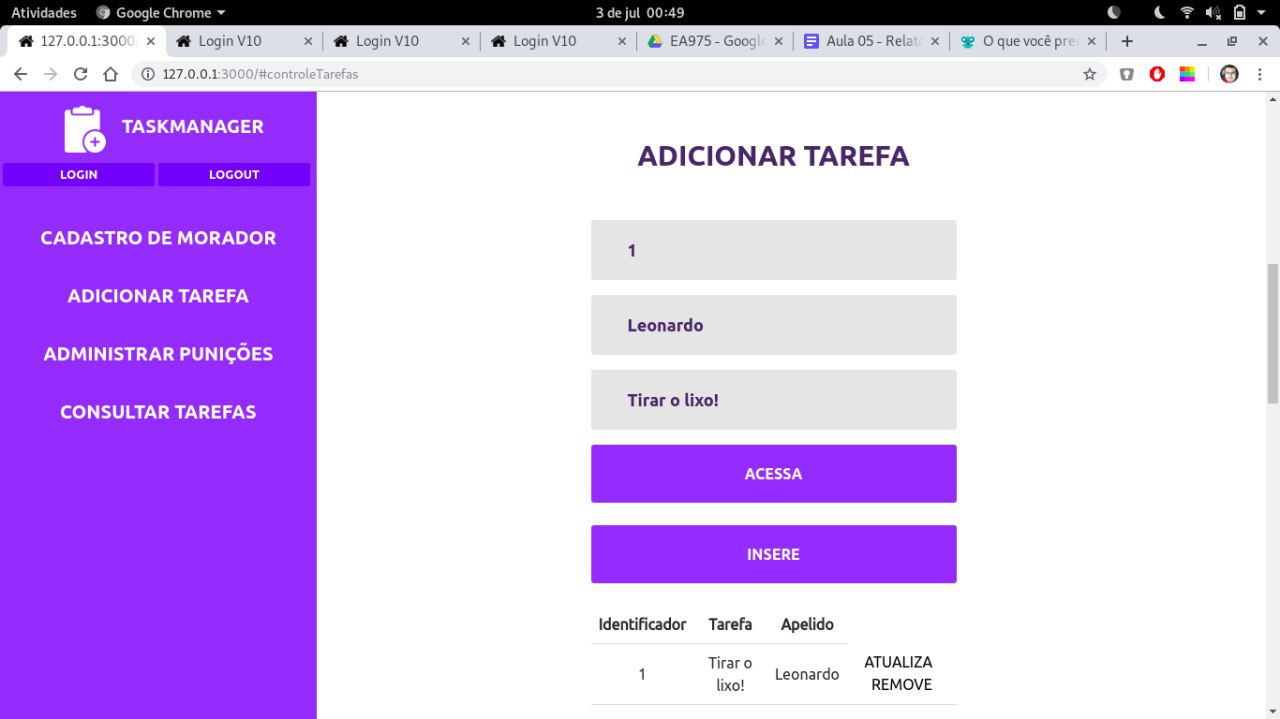
**Teste 3:** Acessar moradores sem especificar nenhum ra. Recebemos uma lista de todos.



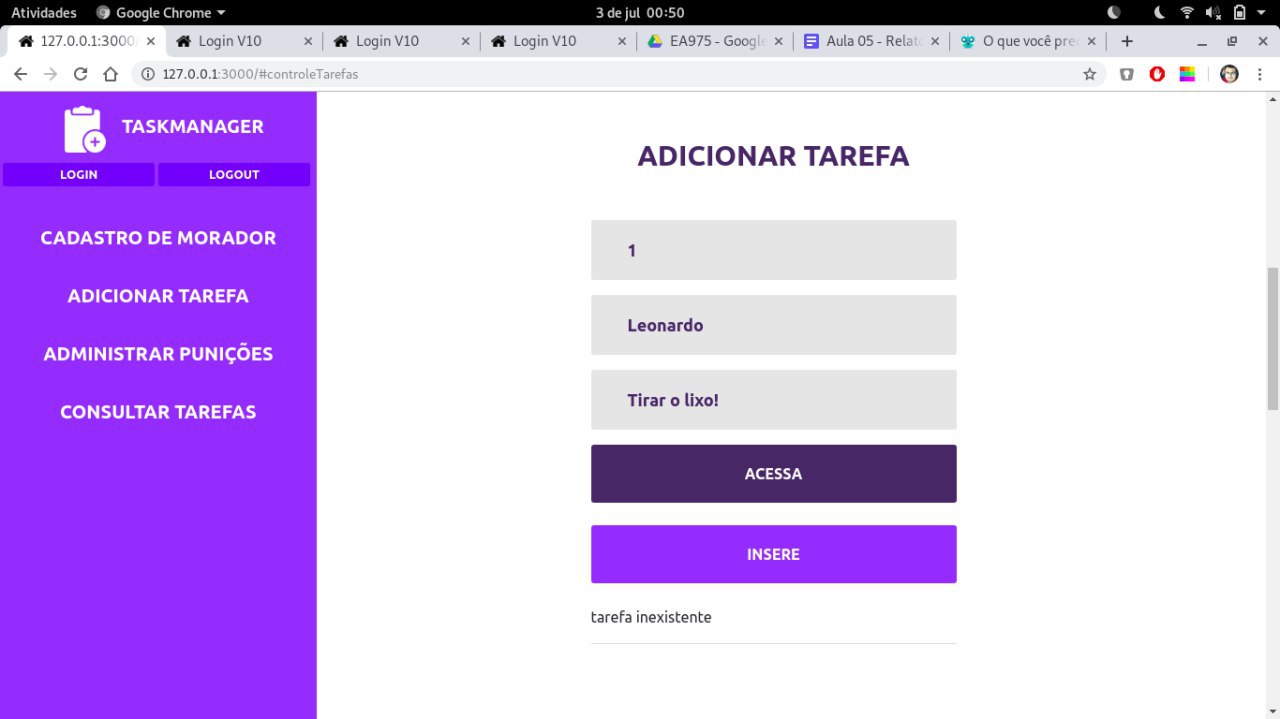
**Teste 4:** Atualizar o curso do primeiro da lista:



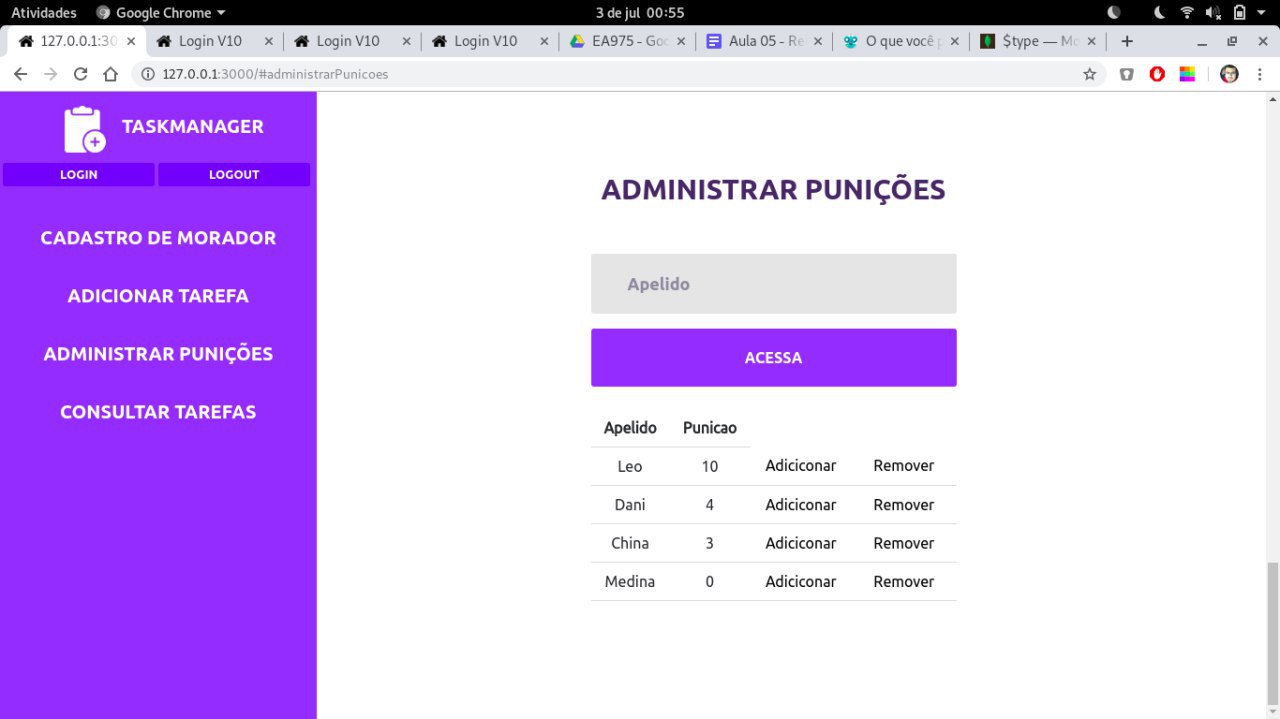
**Teste 5:** Adicionar uma tarefa



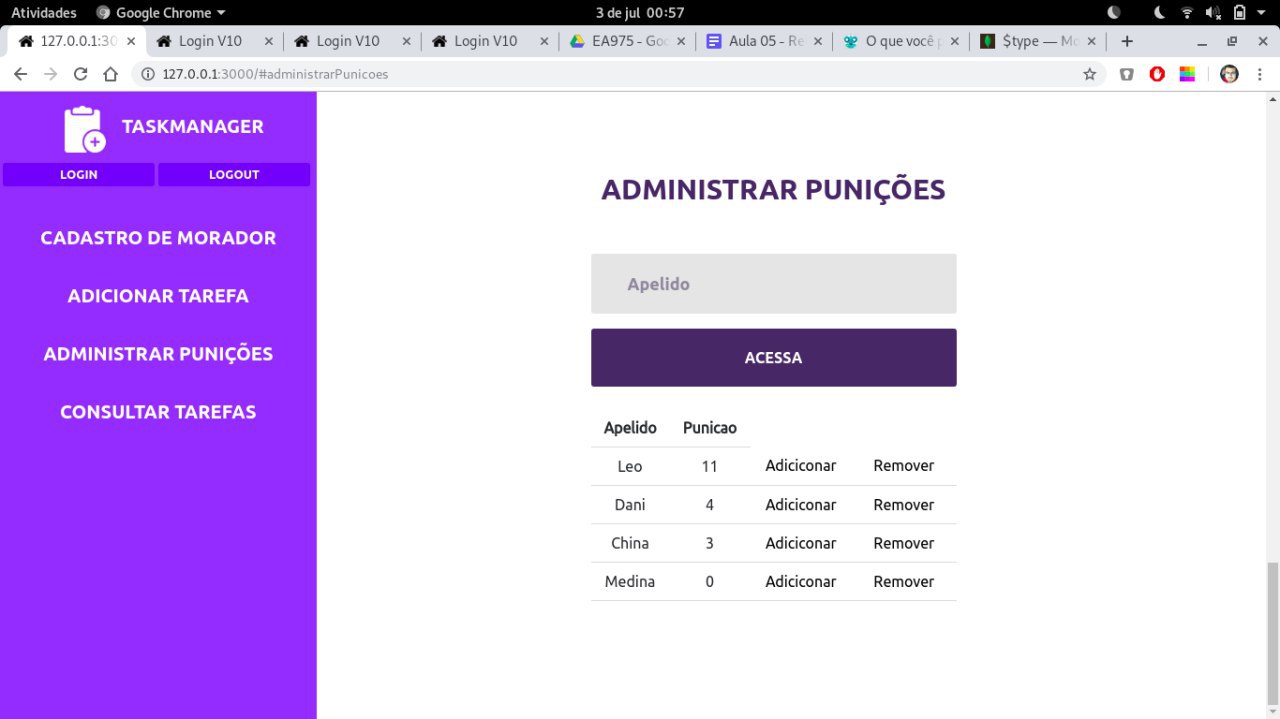
**Teste 6:** Remover tarefa. Tentando acessar novamente.

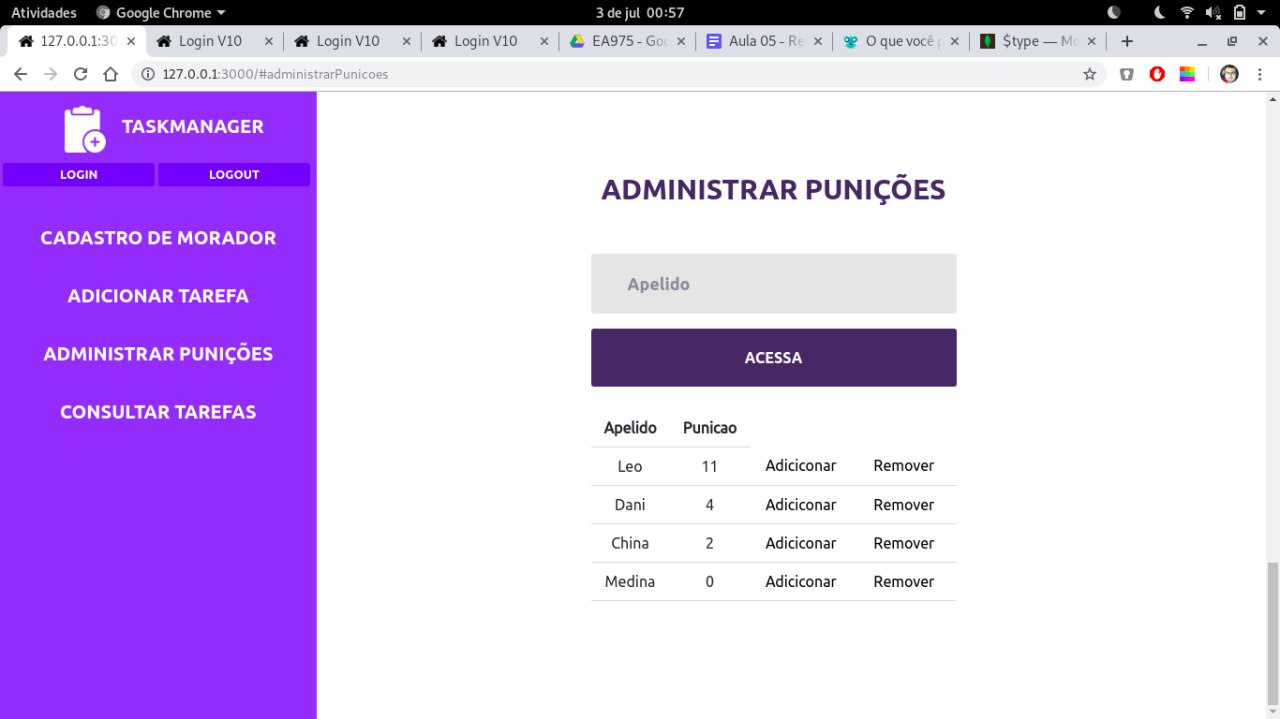


**Teste 7:** Administrar punições sem especificar nenhum apelido : lista todos.

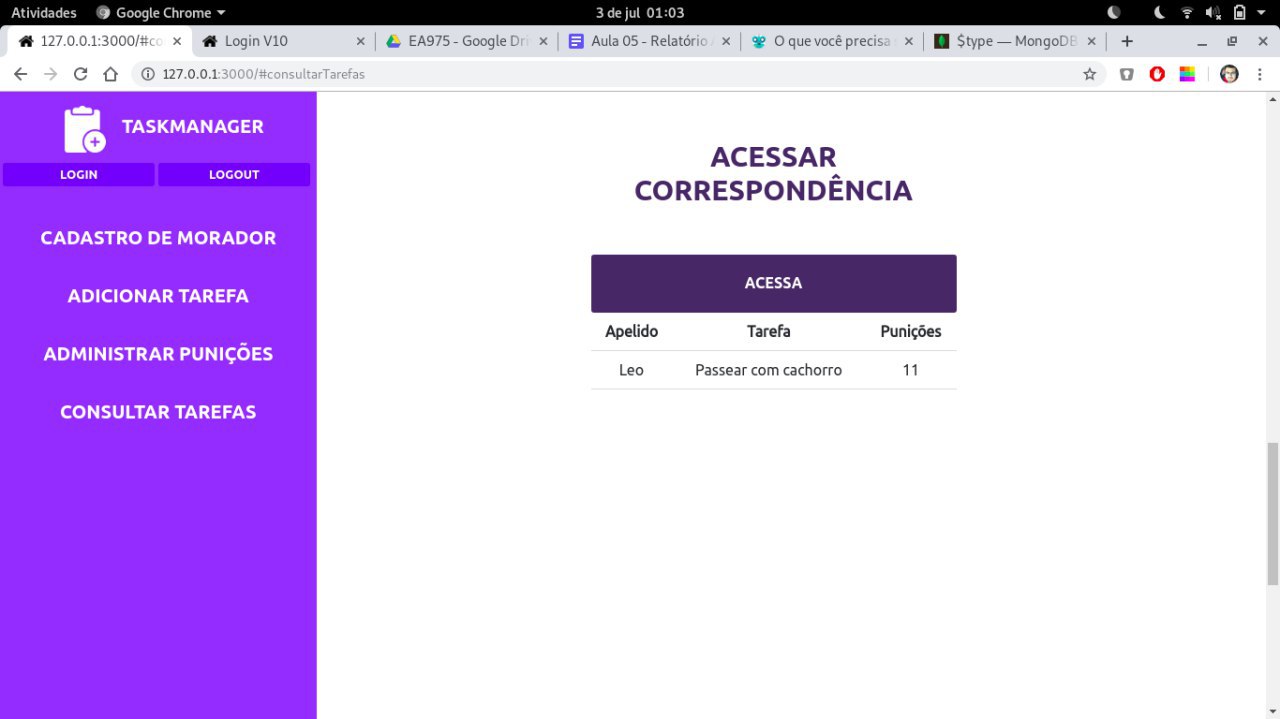


**Teste 8:** Adicionar uma punição para "Leo".



**Teste 9:** Remover uma punição de "China"  
  


**Teste 10:** Consultar todas as tarefas. Não necessitando de login. Apenas uma tarefa tem um morador correspondente.



1. Conclusão

Com esse projeto, foi possível aplicar as teorias que vimos em em Engenharia de Software. Sendo elas: Estudar os requisitos, implementá-los, além de estudar os casos de uso e fazer toda a documentação do software. Em específico também desenvolvemos o projeto com a arquitetura de software MVC. MVC é um dos mais difundidos e utilizados pelos desenvolvedores principalmente pela funcionalidade e objetividade, o que faz o trabalho agregar muito em conhecimento. Foi possível dividir as tarefas e desenvolver os conhecimentos.

1. Apêndice

**app.js**

// Servidor da aplicação

var express = require('express');

var path = require('path');

var favicon = require('serve-favicon');

var logger = require('morgan');

var cookieParser = require('cookie-parser');

var bodyParser = require('body-parser');

// adicione "ponteiro" para o MongoDB

var mongoStudent = require('./models/mongoStudent');

var mongoTask = require('./models/mongoTask');

var mongoUser = require('./models/mongoUser');

// comente as duas linhas abaixo

// var index = require('./routes/index');

// var users = require('./routes/users');

var app = express();

// view engine setup

// app.set('views', path.join(\_\_dirname, 'views'));

app.set('view engine', 'jade');

// serve static files

app.use('/', express.static(\_\_dirname + '/'));

// uncomment after placing your favicon in /public

// app.use(favicon(path.join(\_\_dirname, 'public', 'favicon.ico')));

app.use(logger('dev'));

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({"extended" : false }));

app.use(cookieParser());

app.use(express.static(path.join(\_\_dirname, 'public')));

// adicione as duas linhas abaixo

var router = express.Router();

app.use('/', router); // deve vir depois de app.use(bodyParser...

// comente as duas linhas abaixo

// app.use('/', index);

// app.use('/users', users);

// catch 404 and forward to error handler

app.use(function(req, res, next) {

var err = new Error('Not Found');

err.status = 404;

next(err);

});

// error handler

app.use(function(err, req, res, next) {

// set locals, only providing error in development

res.locals.message = err.message;

res.locals.error = req.app.get('env') === 'development' ? err : {};

// render the error page

res.status(err.status || 500);

res.render('error');

});

module.exports = app;

// codigo abaixo adicionado para o processamento das requisições

// HTTP GET, POST, PUT, DELETE

function checkAuth(req, res) {

cookies = req.cookies;

if(! cookies || ! cookies.userAuth) return 'unauthorized';

cauth = cookies.userAuth;

var content = JSON.parse(cauth);

var key = content.key;

var role = content.role;

if(key == 'secret') return role

return 'unauthorized';

}

// index.html

router.route('/')

.get(function(req, res) { // GET

var path = 'index.html';

res.header('Cache-Control', 'no-cache');

res.sendFile(path, {"root": "./"});

}

);

router.route('/userPunishment')

.get(function(req, res) {

if(checkAuth(req, res) == 'unauthorized') {

res.status(401).send('Unauthorized');

return;

}

mongoStudent.find({}, function(erro, data) {

if(erro){

response = {"resultado" : "falha de acesso ao BD"};

} else {

response = {"userPunishment": data};

}

res.json(response);

})

});

router.route('/userPunishment/:apelido')

.get(function(req, res) {

if(checkAuth(req, res) == 'unauthorized') {

res.status(401).send('Unauthorized');

return;

}

var response = {};

var query = {"apelido": req.params.apelido};

mongoStudent.findOne(query, function(erro, data) {

if(erro) {

response = {"resultado": "falha de acesso ao BD"};

} else if(data == null) {

response = {"resultado": "aluno inexistente"};

} else {

response = {"userPunishment": [data]};

}

res.json(response);

})

})

.put(function(req, res){

if(checkAuth(req, res) == 'unauthorized') {

res.status(401).send('Unauthorized');

return;

}

var response = {};

var query = {"apelido": req.params.apelido};

mongoStudent.findOne(query, function(erro, data){

if(erro){

response = {"resultado": "falha de acesso ao BD"};

} else {

var punishment = data.punicao + 1;

var update = {"punicao": punishment};

mongoStudent.findOneAndUpdate(query, update, function(erro, data){

if(erro){

response = {"resultado": "falha no acesso ao BD"};

} else {

mongoStudent.find({}, function(erro, new\_data){

if(erro){

response = {"resultado": "falha no acesso BD"}

} else {

response = {"resultado": "punição acrescentada", "userPunishment": new\_data};

}

res.json(response);

})

}

})

}

})

})

.delete(function(req, res){

if(checkAuth(req, res) == 'unauthorized') {

res.status(401).send('Unauthorized');

return;

}

var response = {};

var query = {"apelido": req.params.apelido};

mongoStudent.findOne(query, function(erro, data){

if(erro){

response = {"resultado": "falha de acesso ao BD"};

} else {

var punishment = data.punicao - 1;

var update = {"punicao": punishment};

mongoStudent.findOneAndUpdate(query, update, function(erro, data){

if(erro){

response = {"resultado": "falha no acesso ao BD"};

} else {

mongoStudent.find({}, function(erro, new\_data){

if(erro){

response = {"resultado": "falha no acesso BD"}

} else {

response = {"resultado": "punição acrescentada", "userPunishment": new\_data};

}

res.json(response);

})

}

})

}

})

});

router.route('/userTask') // operacoes sobre todos as tarefas

.get(function(req, res) { // GET

let response = {};

mongoStudent.find({}, function(erro, dataStudent) {

if(erro)

response = {"resultado": "falha de acesso ao BD"};

else{

let flag = 0;

mongoTask.find({"apelido": dataStudent.map((data) => data.apelido)}, function(erro, dataTask) {

if(erro)

response = {"resultado": "falha de acesso ao BD"};

else {

response = {

userTask: dataTask.map((task) => {

let student = dataStudent.find((s) => s.apelido === task.apelido)

return {

tarefa: task.nome,

apelido: student.apelido,

punicao: student.punicao

}

})

};

}

res.json(response);

})

}

})

});

router.route('/task') // operacoes sobre todos as tarefas

.get(function(req, res) { // GET

if(checkAuth(req, res) == 'unauthorized') {

res.status(401).send('Unauthorized');

return;

}

var response = {};

mongoTask.find({}, function(erro, data) {

if(erro)

response = {"resultado": "falha de acesso ao BD"};

else

response = {"task": data};

res.json(response);

}

)

}

)

.post(function(req, res) { // POST (cria)

//Verifica login

if(checkAuth(req, res) != 'admin') {

res.status(401).send('Unauthorized');

return;

}

var query = {"identificador": req.body.identificador};

var query2 = {"nome": req.body.apelido};

console.log(query2);

var response = {};

mongoTask.findOne(query, function(erro, data) {

if (data == null) {

mongoStudent.findOne(query2, function(dataMorador, erro){

if(dataMorador == null){

response = {"resultado" : "Não existe esse morador!"}

res.json(response);

} else if(erro){

response = {"resultado" : "Falha de acesso ao BD!"}

res.json(response);

} else {

var db = new mongoTask();

db.identificador = req.body.identificador;

db.nome = req.body.nome;

db.apelido = req.body.apelido;

db.save(function(erro) {

if(erro) {

response = {"resultado": "Falha de acesso ao BD!"};

res.json(response);

} else {

response = {"resultado": "tarefa inserida"};

res.json(response);

}

})

}

})

} else {

response = {"resultado": "tarefa ja existente"};

res.json(response);

}

})

});

router.route('/task/:identificador') // operacoes sobre uma tarefa

.get(function(req, res) { // GET

if(checkAuth(req, res) == 'unauthorized') {

res.status(401).send('Unauthorized');

return;

}

var response = {};

var query = {"identificador": req.params.identificador};

mongoTask.findOne(query, function(erro, data) {

if(erro) response = {"resultado": "falha de acesso ao BD"};

else if (data == null) response = {"resultado": "tarefa inexistente"};

else response = {"task": [data]};

res.json(response);

}

)

}

)

.put(function(req, res) { // PUT (altera)

if(checkAuth(req, res) != 'admin') {

res.status(401).send('Unauthorized');

return;

}

var response = {};

var query = {"identificador": req.params.identificador};

var data = {"nome": req.body.nome, "apelido": req.body.apelido};

mongoTask.findOneAndUpdate(query, data, function(erro, data) {

if(erro) response = {"resultado": "falha de acesso ao DB"};

else if (data == null) response = {"resultado": "tarefa inexistente"};

else response = {"resultado": "tarefa atualizado"};

res.json(response);

}

)

}

)

.delete(function(req, res) { // DELETE (remove)

if(checkAuth(req, res) != 'admin') {

res.status(401).send('Unauthorized');

return;

}

var response = {};

var query = {"identificador": req.params.identificador};

mongoTask.findOneAndRemove(query, function(erro, data) {

if(erro) response = {"resultado": "falha de acesso ao DB"};

else if (data == null) response = {"resultado": "tarefa inexistente"};

else response = {"resultado": "tarefa removido"};

res.json(response)

}

)

}

);

router.route('/alunos') // operacoes sobre todos os alunos

.get(function(req, res) { // GET

if(checkAuth(req, res) == 'unauthorized') {

res.status(401).send('Unauthorized');

return;

}

var response = {};

mongoStudent.find({}, function(erro, data) {

if(erro)

response = {"resultado": "falha de acesso ao BD"};

else

response = {"alunos": data};

res.json(response);

}

)

}

)

.post(function(req, res) { // POST (cria)

if(checkAuth(req, res) != 'admin') {

res.status(401).send('Unauthorized');

return;

}

var query = {"ra": req.body.ra};

var response = {};

mongoStudent.findOne(query, function(erro, data) {

if (data == null) {

var db = new mongoStudent();

db.ra = req.body.ra;

db.apelido = req.body.apelido;

db.nome = req.body.nome;

db.curso = req.body.curso;

db.punicao = 0;

db.save(function(erro) {

if(erro) {

response = {"resultado": "falha de acesso ao BD"};

res.json(response);

} else {

response = {"resultado": "aluno inserido"};

res.json(response);

}

}

)

} else {

response = {"resultado": "aluno ja existente"};

res.json(response);

}

}

)

}

);

router.route('/alunos/:ra') // operacoes sobre um aluno (RA)

.get(function(req, res) { // GET

if(checkAuth(req, res) == 'unauthorized') {

res.status(401).send('Unauthorized');

return;

}

var response = {};

var query = {"ra": req.params.ra};

mongoStudent.findOne(query, function(erro, data) {

if(erro) response = {"resultado": "falha de acesso ao BD"};

else if (data == null) response = {"resultado": "aluno inexistente"};

else response = {"alunos": [data]};

res.json(response);

}

)

}

)

.put(function(req, res) { // PUT (altera)

if(checkAuth(req, res) != 'admin') {

res.status(401).send('Unauthorized');

return;

}

var response = {};

var query = {"ra": req.params.ra};

var data = {"nome": req.body.nome, "curso": req.body.curso};

mongoStudent.findOneAndUpdate(query, data, function(erro, data) {

if(erro) response = {"resultado": "falha de acesso ao DB"};

else if (data == null) response = {"resultado": "aluno inexistente"};

else response = {"resultado": "aluno atualizado"};

res.json(response);

}

)

}

)

.delete(function(req, res) { // DELETE (remove)

if(checkAuth(req, res) != 'admin') {

res.status(401).send('Unauthorized');

return;

}

var response = {};

var query = {"ra": req.params.ra};

mongoStudent.findOneAndRemove(query, function(erro, data) {

if(erro) response = {"resultado": "falha de acesso ao DB"};

else if (data == null) response = {"resultado": "aluno inexistente"};

else response = {"resultado": "aluno removido"};

res.json(response)

}

)

}

);

router.route('/authentication') // autenticação

.get(function(req, res) { // GET

var path = 'login.html';

res.header('Cache-Control', 'no-cache');

res.sendFile(path, {"root": "./"});

}

)

.post(function(req, res) {

var query = {"username": req.body.user, "pass": req.body.pass};

//Verifica usuario e senha no banco de dados

mongoUser.findOne(query, function(erro, data){

if(data == null){

res.status(401).send("Falha na autenticação!");

} else if(erro){

res.status(401).send("Erro na autenticação!");

} else {

var content = {"key":"secret", "role":"admin"};

res.cookie('userAuth', JSON.stringify(content), {'maxAge': 3600000\*24\*5});

res.status(200).send('Sucesso'); // OK

}

})

}

)

.delete(function(req, res) {

if(checkAuth(req, res) != 'unauthorized') {

res.clearCookie('userAuth'); // remove cookie no cliente

res.status(200).send('Sucesso');

} else {

res.status(401).send('Unauthorized');

return;

}

}

);