12-5-2018

Iván González Mahagamage

Universisdad de oviedo

Entregable 2

Sistemas distribuidos e internet

**Autores**: **Iván González Mahagamage**

Estudiante de Ingeniería Informática del Software en la Universidad de Oviedo

**Fecha**: 12 de mayo de 2018

**Versión:** V1

Contenido

[Implementación 4](#_Toc513920475)

[0. Aspectos comunes 4](#_Toc513920476)

[0.1. Colecciones de la aplicación. 4](#_Toc513920477)

[0.2. Creación del log de actividad. 5](#_Toc513920478)

[0.3. Gestor de base de datos 5](#_Toc513920479)

[0.4. Plantillas 6](#_Toc513920480)

[0.5. Configuración principal de la aplicación 11](#_Toc513920481)

[0.5.3. Rutadores 12](#_Toc513920482)

[1. Aplicación web 13](#_Toc513920483)

[1.1. Público: registrarse como usuario 13](#_Toc513920484)

[1.2. Público: iniciar sesión 19](#_Toc513920485)

[1.3. Usuario registrado: listar todos los usuarios de la aplicación 21](#_Toc513920486)

[1.4. Usuario registrado: buscar entre todos los usuarios de la aplicación 24](#_Toc513920487)

[1.5. Usuario registrado: enviar una invitación de amistad a un usuario 24](#_Toc513920488)

[1.6. Usuario registrado: listar las invitaciones de amistad recibidas 24](#_Toc513920489)

[1.7. Usuario registrado: aceptar una invitación recibida 24](#_Toc513920490)

[1.8. Usuario registrado: listar los usuarios amigos 24](#_Toc513920491)

[2. Servicios web – Implementación de la API de Servicios Web REST 24](#_Toc513920492)

[2.1. Identificarse con usuario – token 24](#_Toc513920493)

[2.2. Usuario identificado: listar todos los amigos 24](#_Toc513920494)

[2.3. Usuario identificado: Crear un mensaje 24](#_Toc513920495)

[2.4. Usuario identificado: Obtener mis mensajes de una “conversación” 24](#_Toc513920496)

[2.5. Usuario identificado: Marcar mensaje como leído 24](#_Toc513920497)

[3. Cliente – Aplicación jQuery 24](#_Toc513920498)

[3.1. Autentificación del usuario 24](#_Toc513920499)

[3.2. Mostrar la lista de amigos 24](#_Toc513920500)

[3.3. Mostrar los mensajes 24](#_Toc513920501)

[3.4. Crear mensaje 24](#_Toc513920502)

[3.5. Marcar mensajes como leídos de forma automática 24](#_Toc513920503)

[3.6. Mostrar el número de mensajes sin leer 25](#_Toc513920504)

[3.7. Ordenar la lista de amigos por último mensaje 25](#_Toc513920505)

[Prueba Unitarias 25](#_Toc513920506)

[0. Configuración de las pruebas y utilidades 25](#_Toc513920507)

[1.1. Registro de Usuario con datos válidos. 25](#_Toc513920508)

[1.2. Registro de Usuario con datos inválidos (repetición de contraseña invalida). 25](#_Toc513920509)

[2.1. Inicio de sesión con datos válidos. 25](#_Toc513920511)

[2.2. Inicio de sesión con datos inválidos. 25](#_Toc513920512)

[3.1. Acceso al listado de usuarios desde un usuario en sesión. 25](#_Toc513920514)

[3.2. Intento de acceso con URL desde un usuario no identificado al listado de usuarios desde un usuario en sesión. 25](#_Toc513920515)

[4.1. Realizar una búsqueda valida en el listado de usuarios desde un usuario en sesión. 25](#_Toc513920517)

[4.2. Intento de acceso con URL a la búsqueda de usuarios desde un usuario no identificado. 25](#_Toc513920518)

[5.1. Enviar una invitación de amistad a un usuario de forma valida. 25](#_Toc513920520)

[5.2. Enviar una invitación de amistad a un usuario al que ya le habíamos invitado la invitación previamente. 25](#_Toc513920521)

[6.1. Listar las invitaciones recibidas por un usuario. 25](#_Toc513920523)

[7.1. Aceptar una invitación recibida. 25](#_Toc513920525)

[8.1. Listar los amigos de un usuario, realizar la comprobación con una lista que al menos tenga un amigo. 26](#_Toc513920527)

[C1.1. Inicio de sesión con datos válidos. 26](#_Toc513920528)

[C1.2. Inicio de sesión con datos inválidos (usuario no existente en la aplicación). 26](#_Toc513920529)

[C2.1. Acceder a la lista de amigos de un usuario, que al menos tenga tres amigos. 26](#_Toc513920530)

[C2.2. Acceder a la lista de amigos de un usuario, y realizar un filtrado para encontrar a un amigo concreto, el nombre a buscar debe coincidir con el de un amigo. 26](#_Toc513920531)

[C3.1. Acceder a la lista de mensajes de un amigo “chat”, la lista debe contener al menos tres mensajes. 26](#_Toc513920532)

[C4.1. Acceder a la lista de mensajes de un amigo “chat” y crear un nuevo mensaje, validar que el mensaje aparece en la lista de mensajes. 26](#_Toc513920533)

[C5.1. Identificarse en la aplicación y enviar un mensaje a un amigo, validar que el mensaje enviado aparece en el chat. Identificarse después con el usuario que recibido el mensaje y validar que tiene un mensaje sin leer, entrar en el chat y comprobar que el mensaje pasa a tener el estado leído. 26](#_Toc513920534)

[C6.1. Identificarse en la aplicación y enviar tres mensajes a un amigo, validar que los mensajes enviados aparecen en el chat. Identificarse después con el usuario que recibido el mensaje y validar que el número de mensajes sin leer aparece en la propia lista de amigos. 26](#_Toc513920535)

[C7.1. Identificarse con un usuario A que al menos tenga 3 amigos, ir al chat del ultimo amigo de la lista y enviarle un mensaje, volver a la lista de amigos y comprobar que el usuario al que se le ha enviado el mensaje este en primera posición. Identificarse con el usuario B y enviarle un mensaje al usuario A. Volver a identificarse con el usuario A y ver que el usuario que acaba de mandarle el mensaje es el primero en su lista de amigos. 27](#_Toc513920536)

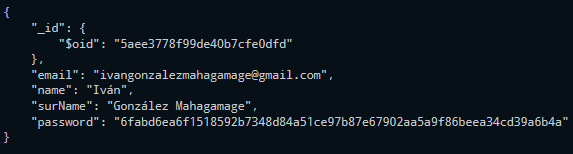
# Implementación

## Aspectos comunes

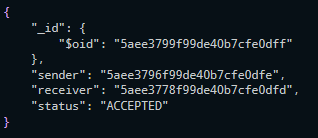
### Colecciones de la aplicación.

Para esta aplicación se han creado tres colecciones para su funcionamiento:

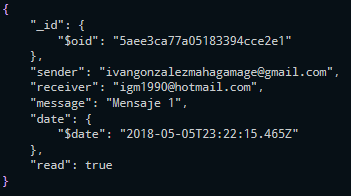
* Users: en esta colección se guarda la información de los usuarios registrados. Se guarda el nombre, apellidos, email y contraseña del usuario. La estructura que siguen los elementos es la siguiente:



* Requests: guarda la información de las peticiones enviadas de los usuarios entre sí. Tiene un campo “status” que indica si dos usuarios son amigos o no. Además, tiene dos campos para guardas los id de los usuarios a los que referencia.



* Messages: guarda los mensajes que se envían entre si los usuarios en la aplicación web. Contiene el emisor y el receptor del mensaje, el contenido de este, la fecha de creación del mensaje y campo “read” que indica si el receptor a leído el mensaje.



### Creación del log de actividad.

Para crear el log de registro de actividad de la aplicación se ha instalado en el proyecto la librería “log4js”. Su repositorio es <https://github.com/log4js-node/log4js-node> y para su instalación se han seguido los siguientes pasos:

* Abrir una consola dentro del proyecto y ejecutar el siguiente comando.

“npm install log4js”

* Creamos el objeto “logger” en app.js
  + Filename: directorio y nombre del fichero log que se creara.
  + Level: nivel a partir el cual vamos a guardar fallos

let log4js = require('log4js');

log4js.configure({

appenders: { sdi: { type: 'file', filename: 'logs/sdi.log' } },

categories: { default: { appenders: ['sdi'], level: 'trace' } }

});

let logger = log4js.getLogger('sdi');

* Añadimos el objeto “logger ” al objeto “app”.

app.set('logger', logger);

* En las clases del paquete “Routes” llamamos al objeto “logger” para ir creando el registro. A continuación, se muestra un ejemplo.

app.get("/signup", function (req, res) {

let answer = swig.renderFile('views/signup.html', {});

res.send(answer);

app.get("logger").info('Usuario se va a registrar');

});

* El contenido del log será similar al siguiente:

[2018-05-06T18:07:46.985] [INFO] sdi - Usuario se intenta registrar

[2018-05-06T18:07:46.986] **[ERROR]** sdi - No coinciden las contraseñas

### Gestor de base de datos

Para simplificar y crear los mínimos métodos para manejar la base de datos se ha creado un gestor genérico llamado “repository.js”. La diferencia con los que se han utilizado en las practicas es que hay que especificar en la colección a la que hacen referencia. Un ejemplo:

getElements: function (element, nameCollection, funcionCallback) {

this.mongo.MongoClient.connect(this.app.get('db'), function (err, db) {

if (err) {

funcionCallback(null);

} else { getElements: function (element, nameCollection, funcionCallback) {

this.mongo.MongoClient.connect(this.app.get('db'), function (err, db) {

if (err) {

funcionCallback(null);

} else {

let collection = db.collection(nameCollection);

collection.find(element).toArray(function (err, elements) {

if (err) {

funcionCallback(null);

} else {

funcionCallback(elements);

}

db.close();

});

}

});

},

let collection = db.collection(nameCollection);

collection.find(element).toArray(function (err, elements) {

if (err) {

funcionCallback(null);

} else {

funcionCallback(elements);

}

db.close();

});

}

});

},

También se ha creado un objeto “ObjectID” para leer los id de los documentos, una variación a lo dado en prácticas.

// Base de datos

let mongo = require('mongodb');

let ObjectId = require('mongodb').ObjectID;

// Controladores

require("./routes/rUsers.js")(app, swig, repository);

require("./routes/rRequests.js")(app, swig, repository, ObjectId);

require("./routes/api.js")(app, repository, ObjectId);

### Plantillas

Se ha utilizado el motor de plantillas Swig para este entregable tal como se ha enseñado en las prácticas de la asignatura:

* Primero se ha creado una plantilla base denominada “template.html” de la cual heredaran el resto. En ella se definen 4 bloques:
  + Title: indica el título de la página.
  + Nav: contiene el menú de navegación de la aplicación.
  + Main: contiene el contenido principal de cada página.
  + Pagination: contiene el menú de paginación si la página lo contiene.

<!DOCTYPE html>

<html lang="es">

<head>

<meta charset="utf-8" />

<meta name="author" content="Iván González Mahagamage" />

<title>{% block title %} SDI - UO239795 {% endblock %}</title>

<link rel="stylesheet" type="text/css" href="/css/style.css" />

<link rel="stylesheet" type="text/css" href="/css/footer.css" />

<link rel="stylesheet" type="text/css" href="/css/sketchy.css" />

<script src="/script/jquery.js"></script>

<script src="/script/popper.min.js"></script>

<script src="/script/bootstrap.min.js"></script>

</head>

<body>

{% block nav %} {% endblock %}

<main class="container">

{% block main %} {% endblock %} {% block pagination %} {% endblock %}

</main>

<footer>

<address>

<p>

<span>Autor</span>: Iván González Mahagamage</p>

<p>

<span>Contacto</span>:

<a href="mailto:uo239795@uniovi.es" style="color: white">uo239795@uniovi.es</a>

</p>

</address>

<div class="footer-copyright py-3 text-center">

<div class="container-fluid">

© 2018 Copyright:

<a href="https://www.linkedin.com/in/iv%C3%A1n-gonz%C3%A1lez-mahagamage-5bb340107/">

Linkedin</a>

</div>

</div>

</footer>

</body>

</html>

* A partir de esta plantilla se han creado otras tres para el correcto funcionamiento de la aplicación:
  + “templateLogin.html”: es la plantilla de las paginas en las cuales el usuario no se ha identificado. Estas no contienen paginación.

{% extends "template.html" %} {% block nav %}

<nav class="navbar navbar-expand-lg navbar-dark bg-primary">

<a class="navbar-brand" href="/">Inicio</a>

<button class="navbar-toggler collapsed" type="button" data-toggle="collapse" data-target="#navbarColor01" aria-controls="navbarColor01"

aria-expanded="false" aria-label="Toggle navigation" style="">

<span class="navbar-toggler-icon"></span>

</button>

<div class="navbar-collapse collapse" id="navbarColor01">

<ul class="nav navbar-nav ml-auto">

<li class="nav-item">

<a class="nav-link" href="/signup">Registrarse</a>

</li>

<li class="nav-item">

<a class="nav-link" href="/login">Identifícate</a>

</li>

</ul>

</div>

</nav>

{% endblock %}

* + “templateLogout.html”: es la plantilla que usan las paginas en las cuales el usuario se ha identificado y no tienen paginación.

{% extends "template.html" %} {% block nav %}

<nav class="navbar navbar-expand-lg navbar-dark bg-primary">

<a class="navbar-brand" href="/home">Inicio</a>

<button class="navbar-toggler collapsed" type="button" data-toggle="collapse" data-target="#navbarColor01" aria-controls="navbarColor01"

aria-expanded="false" aria-label="Toggle navigation" style="">

<span class="navbar-toggler-icon"></span>

</button>

<div class="navbar-collapse collapse" id="navbarColor01">

<ul class="navbar-nav mr-auto">

<li class="nav-item active">

<a class="nav-link" href="/list">Usuarios</a>

</li>

<li class="nav-item active">

<a class="nav-link" href="/requests">Peticiones</a>

</li>

<li class="nav-item active">

<a class="nav-link" href="/friends">Amigos</a>

</li>

</ul>

<ul class="nav navbar-nav ml-auto">

<li class="nav-item">

<a class="nav-link" href="/logout">Logout</a>

</li>

</ul>

</div>

</nav>

{% endblock %}

* + “templatePagination.html”: hereda de “templateLogout.html” para paginar las páginas que lo necesiten.

{% extends "templateLogout.html" %}

{% block pagination %}

<div class="text-center">

<ul class="pagination pagination-centered">

<li class="page-item">

<a id="first" class="page-link" href="?pg=1">Primera</a>

</li>

{% if pgCurrent-1 >= 1 %}

<li class="page-item">

<a id="before" class="page-link" href="?pg={{ pgCurrent -1 }}">{{ pgCurrent -1 }}</a>

</li>

{% endif %}

<li class="page-item active">

<a id="now" class="page-link" href="?pg={{ pgCurrent }}">{{ pgCurrent }}</a>

</li>

{% if pgCurrent+1 <= pgLast %}

<li class="page-item">

<a id="next" class="page-link" href="?pg={{ pgCurrent+1 }}">{{ pgCurrent+1 }}</a>

</li>

{% endif %}

<li class="page-item">

<a id="last" class="page-link" href="?pg={{ pgLast }}">Última</a>

</li>

</ul>

</div>

<script src="/script/pagination.js"></script>

<script>

var pag = new Pagination({{ pgCurrent }});

</script>

{% endblock %}

Para que estas páginas guarden la búsqueda se les añade un script “pagination.js”

"use strict";

class Pagination {

    constructor(limit) {

        var page = this.getURLParameter('pg');

        if (page == undefined) {

            page = 0;

        }

        var searchText = this.getURLParameter('searchText');

        if (searchText == undefined)

            searchText = "";

        else

            searchText = "&searchText=" + searchText;

        this.changeHref(limit, page, searchText);

    }

    getURLParameter(sParam) {

        var sPageURL = window.location.search.substring(1);

        var sURLVariables = sPageURL.split('&');

        for (var i = 0; i < sURLVariables.length; i++) {

            var sParameterName = sURLVariables[i].split('=');

            if (sParameterName[0] == sParam) {

                return sParameterName[1];

            }

        }

    }

    changeHref(limit, page, searchText) {

        $("#first").attr("href", "?pg=0" + searchText);

        $("#before").attr("href", "?pg=" + (parseInt(page) - 1) + searchText);

        $("#now").attr("href", "?pg=" + (parseInt(page)) + searchText);

        $("#next").attr("href", "?pg=" + (parseInt(page) + 1) + searchText);

        $("#last").attr("href", "?pg=" + limit + searchText);

    }

}

### Configuración principal de la aplicación

#### App.js

La configuración de esta clase es prácticamente la misma que en prácticas a excepción de log comentado anteriormente y los interceptores de aplicación y la API.

// routerUserSession

let routerUserSession = express.Router();

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

if (req.session.user) { // dejamos correr la petición

next();

} else {

res.redirect("/login");

}

});

//Aplicar routerUsuarioSession

app.use("/home", routerUserSession);

app.use("/list", routerUserSession);

app.use("/send", routerUserSession);

app.use("/requests", routerUserSession);

app.use("/accepted", routerUserSession);

app.use("/friends", routerUserSession);

// routerUsuarioToken

let routerUsuarioToken = express.Router();

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

// obtener el token, puede ser un parámetro GET , POST o HEADER

let token = req.body.token || req.query.token || req.headers['token'];

if (token != null) {

// verificar el token

jwt.verify(token, 'secreto', function (err, infoToken) {

if (err || (Date.now() / 1000 - infoToken.tiempo) > 24000) {

res.status(403); // Forbidden

res.json({

acceso: false,

error: 'Token invalido o caducado'

});

return;

} else {

res.user = infoToken.user;

next();

}

});

} else {

res.status(403); // Forbidden

res.json({

acceso: false,

mensaje: 'No hay Token'

});

}

});

// Aplicar routerUsuarioToken

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

app.use('/api/messages', routerUsuarioToken);

#### Package.json

A continuación, se muestran todos los paquetes instalados para el funcionamiento de la aplicación.

{

"name": "sdi2-UO239795",

"version": "0.0.1",

"description": "Entregable 2 de la asignatura SDI",

"author": "Iván Gonzalez Mahagamage",

"license": "UNLICENSED",

"repository": "https://github.com/igm1990/SDI",

"dependencies": {

"body-parser": "^1.18.2",

"eslint": "^4.19.1",

"express": "^4.16.3",

"express-session": "^1.15.6",

"install": "^0.11.0",

"jsonwebtoken": "^8.2.1",

"log4js": "^2.5.3",

"mongodb": "^2.2.33",

"swig": "^1.4.2"

}

}

### Rutadores

Para la realización de la práctica se han creados tres rutadores en el fichero “app.js”.

* rUsers: maneja las url que hacen referencia a los usuarios de la aplicación.
* rRequests: maneja las url que hacen referencia a las peticiones de la aplicación.
* api: maneja las url de la api de la aplicación.

// Controladores

require("./routes/rUsers.js")(app, swig, repository);

require("./routes/rRequests.js")(app, swig, repository, ObjectId);

require("./routes/api.js")(app, repository, ObjectId);

Su funcionamiento será detallado en los respectivos casos de uso.

## Aplicación web

### Público: registrarse como usuario

Primero se ha creado una vista para este propósito denominada “signup.html” en el directorio “views”.

{% extends "templates/templateLogin.html" %}

{% block titulo %} Registro - UO239795 {% endblock %}

{% block main %}

<script src="/script/checkForm.js"></script>

<h1>Registrar</h1>

<form method="post" action="/signup">

<fieldset>

<div id="email" class="form-group">

<label>Email:</label>

<input class="form-control" type="email" name="email" placeholder="uo239795@uniovi.es"

required="required" onkeyup="check.checkInputs()"

title="Introduzca un email valido"/>

</div>

<div id="name" class="form-group">

<label>Nombre:</label>

<input class="form-control" type="text" name="name" placeholder="Iván" required="required"

onkeyup="check.checkInputs()" title="Introduzca su nombre"

pattern="[a-zA-ZñÑáéíóúÁÉÍÓÚ]{2,}"/>

</div>

<div id="surName" class="form-group">

<label>Apellidos:</label>

<input class="form-control" type="text" name="surName" placeholder="González Mahagamage"

required="required" onkeyup="check.checkInputs()"

title="Introduzca su apellido"

pattern="[a-zA-ZñÑáéíóúÁÉÍÓÚ ]{2,}"/>

</div>

<div id="password" class="form-group">

<label>Contraseña:</label>

<input class="form-control" type="password" name="password"

placeholder="Introduzca su contraseña" title="Introduzca su contraseña"

pattern=".{5,}" required="required" onkeyup="check.checkInputs()"/>

</div>

<div id="passwordConfirm" class="form-group">

<label>Repetir Contraseña:</label>

<input class="form-control" type="password" name="passwordConfirm" required="required"

placeholder="Reintroduzca su contraseña" title="Reintroduzca su contraseña"

pattern=".{5,}" onkeyup="check.checkInputs()"/>

</div>

<div class="form-group">

<input class="btn btn-success" type="submit" value="Registrar"

onclick="check.checkPasswords()" disabled="disabled"/>

</div>

</fieldset>

</form>

<script src="/script/showMessage.js"></script>

<script>

message.showErrors("email");

message.showErrors("name");

message.showErrors("surName");

message.showErrors("password");

message.showErrors("passwordConfirm");

message.showErrors("coincidence");

</script>

{% endblock %}

Este html usa dos archivos JavaScript, situados en la carpeta “public/script” para comprobar el formulario que debe introducir el usuario.

* “Checkform.js”: realiza comprobaciones en el lado del cliente de los datos introducidos si son correctos.

"use strict";

class Check {

constructor() {}

checkPasswords() {

var password = $('input[name=password]').val();

var repassword = $('input[name=passwordConfirm]').val();

if (password !== repassword) {

alert("Error: las contraseñas no coinciden");

return false;

}

return true;

}

checkInputs() {

$('input[type=submit]').prop('disabled', false);

var inputs = $('input');

for (var i = 0; i < inputs.length; i++) {

var value = inputs[i].value;

if (inputs[i].value === "") {

$('input[type=submit]').prop('disabled', true);

break;

}

}

}

}

var check = new Check();

* “showMessage.js”: si el usuario ha cometido un fallo, este archivo modifica la vista para mostrar un mensaje de error al usuario.

"use strict";

class ShowMessage {

constructor() {}

getURLParameter(sParam) {

var sPageURL = window.location.search.substring(1);

var sURLVariables = sPageURL.split('&');

for (var i = 0; i < sURLVariables.length; i++) {

var sParameterName = sURLVariables[i].split('=');

if (sParameterName[0] == sParam) {

return sParameterName[1];

}

}

}

showMessage(name, type) {

var message = this.getURLParameter(name);

if (message !== undefined) {

message = this.convertCharactersUTF8(message);

var error = "<div class=\"alert alert-dismissible " + type + "\">\n" +

" <button type=\"button\" class=\"close\" data-dismiss=\"alert\">&times;</button>\n" +

" <strong>" + message + "</strong>\n" +

" </div>";

if (name === "coincidence")

name = "passwordConfirm";

$("#" + name).append(error);

}

}

showErrors(name) {

this.showMessage(name, "alert-danger")

}

showInfo(name) {

this.showMessage(name, "alert-success")

}

convertCharactersUTF8(message) {

message = message.split("%20").join(' ');

message = message.split("%C3%A1").join('á');

message = message.split("%C3%A9").join('é');

message = message.split("%C3%AD").join('í');

message = message.split("%C3%B3").join('ó');

message = message.split("%C3%BA").join('ú');

message = message.split("%C3%B1").join('ñ');

message = message.split("%C3%81").join('Á');

message = message.split("%C3%89").join('É');

message = message.split("%C3%8D").join('Í');

message = message.split("%C3%93").join('Ó');

message = message.split("%C3%9A").join('Ú');

message = message.split("%C3%91").join('Ñ');

return message;

}

}

var message = new ShowMessage();

Hay que definir un método en el fichero “rUsers.js” para mostrar esta vista.

app.get("/signup", function (req, res) {

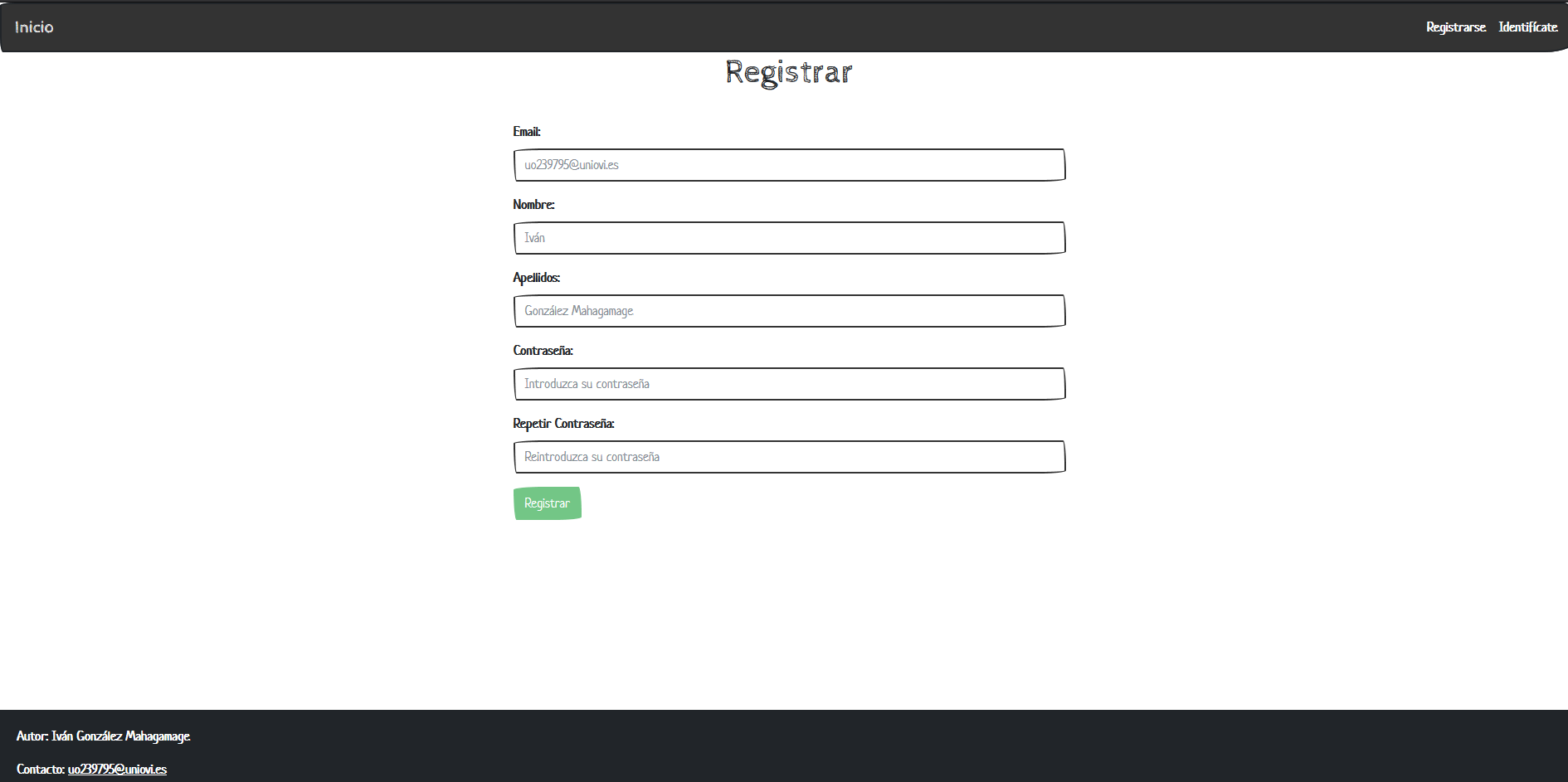
let answer = swig.renderFile('views/signup.html', {});

res.send(answer);

app.get("logger").info('Usuario se va a registrar');

});

La vista resultante de estos ficheros es la siguiente:



Una vez rellenado el formulario hay que procesar los datos, para ello creamos varios métodos en “rUsers.js”.

1. Creamos un método para comprobar los datos del formulario en el lado del servidor.

function checkErrorSignUp(req) {

let errors = [];

if (req.body.email == "") {

app.get("logger").error('Email vacío al registrar el usuario');

errors.push("email=Este campo no puede ser vacío");

}

if (req.body.name.length < 2) {

app.get("logger").error('Longitud del nombre invalida');

errors.push("name=El nombre debe tener entre 5 y 24 caracteres.");

}

if (req.body.surName.length < 2) {

app.get("logger").error('Longitud del apellido invalida');

errors.push("surName=El apellido debe tener entre 5 y 24 caracteres.");

}

if (req.body.password.length < 5) {

app.get("logger").error('Longitud de la contraseña invalida');

errors.push("password=La contraseña debe tener entre 5 y 24 caracteres.");

}

if (req.body.passwordConfirm.length < 5) {

app.get("logger").error('Longitud de la recontraseña invalida');

errors.push("passwordConfirm=La contraseña debe tener entre 5 y 24 caracteres.");

}

if (req.body.password != req.body.passwordConfirm) {

app.get("logger").error('No coinciden las contraseñas');

errors.push("coincidence=Las contraseñas no coinciden.");

}

let url = "";

for (let i = 0; i < errors.length; i++) {

url += "&" + errors[i];

}

return url.substr(1, url.length);

}

1. Creamos un método para procesar la petición POST generada por el formulario.

app.post('/signup', function (req, res) {

app.get("logger").info('Usuario se intenta registrar');

let errors = checkErrorSignUp(req);

if (errors.length > 0) {

res.redirect("/signup?" + errors);

return;

}

let password = app.get("crypto").createHmac('sha256', app.get('key')).update(req.body.password).digest('hex');

let user = {

email: req.body.email,

name: req.body.name,

surName: req.body.surName,

password: password

};

let findByEmail = {

email: req.body.email

};

repository.getElements(findByEmail, "users", function (users) {

if (users == null || users.length == 0) {

repository.addElement(user, "users", function (id) {

if (id == null) {

res.redirect("/signup?email=Error al añadir al usuario. Intentelo más tarde");

} else {

let textSearch = {

email: req.body.email,

password: password

};

app.get("logger").info('Usuario registrado como ' + req.body.email);

autoLogin(textSearch, req, res);

}

});

} else {

app.get("logger").error('Error al registrar al usuarios');

res.redirect("/signup?email=Este correo ya esta registrado.");

}

});

});

1. Si el registro se ha realzado correctamente, se autologuea al usuario para que empiece a usar la aplicación. Para ello creamos un nuevo método para esta función que también será usada cuando un usuario hace login.

function autoLogin(textSearch, req, res) {

repository.getElements(textSearch, "users", function (users) {

if (users == null || users.length == 0) {

req.session.user = null;

res.redirect("/login?error=Email o password incorrecto");

app.get("logger").error('Error al loguear al usuario');

} else {

req.session.user = users[0].email;

res.redirect("/list");

app.get("logger").info('El usuario ' + req.session.user + " se ha logueado correctamente");

}

});

}

### Público: iniciar sesión

Para este caso de uso se ha creado la vista con el fichero “login.html” situado en el directorio “views”.

{% extends "templates/templateLogin.html" %}

{% block titulo %} Login - UO239795 {% endblock %}

{% block main %}

<h1>Identifícate</h1>

<form class="form-horizontal" method="post" action="/login">

<fieldset>

<div class="form-group">

<label class="control-label col-sm-2" for="email">Email:</label>

<div class="col-sm-10">

<input type="email" class="form-control" id="email" name="email" placeholder="uo239795@uniovi.es"

required="required" title="Introduzca un email valido"/>

</div>

</div>

<div class="form-group">

<label class="control-label col-sm-2" for="password">Contraseña:</label>

<div class="col-sm-10">

<input type="password" class="form-control" id="password" name="password"

placeholder="Introduzca su contraseña" title="Introduzca su contraseña"

pattern=".{2,}" required="required"/>

</div>

</div>

<div class="form-group">

<div class="col-sm-offset-2 col-sm-10">

<button id="boton-login" type="submit" class="btn btn-primary">Login</button>

</div>

</div>

<input type="hidden" name="${\_csrf.parameterName}" value="${\_csrf.token}"/>

</fieldset>

<div id="error"></div>

<div id="logout"></div>

</form>

<script src="/script/showMessage.js"></script>

<script>

message.showErrors("error");

message.showInfo("logout");

</script>

{% endblock %}

Para mostrar esta vista se ha modificado el archivo “rUsers.js” añadiendo una nueva función.

app.get("/login", function (req, res) {

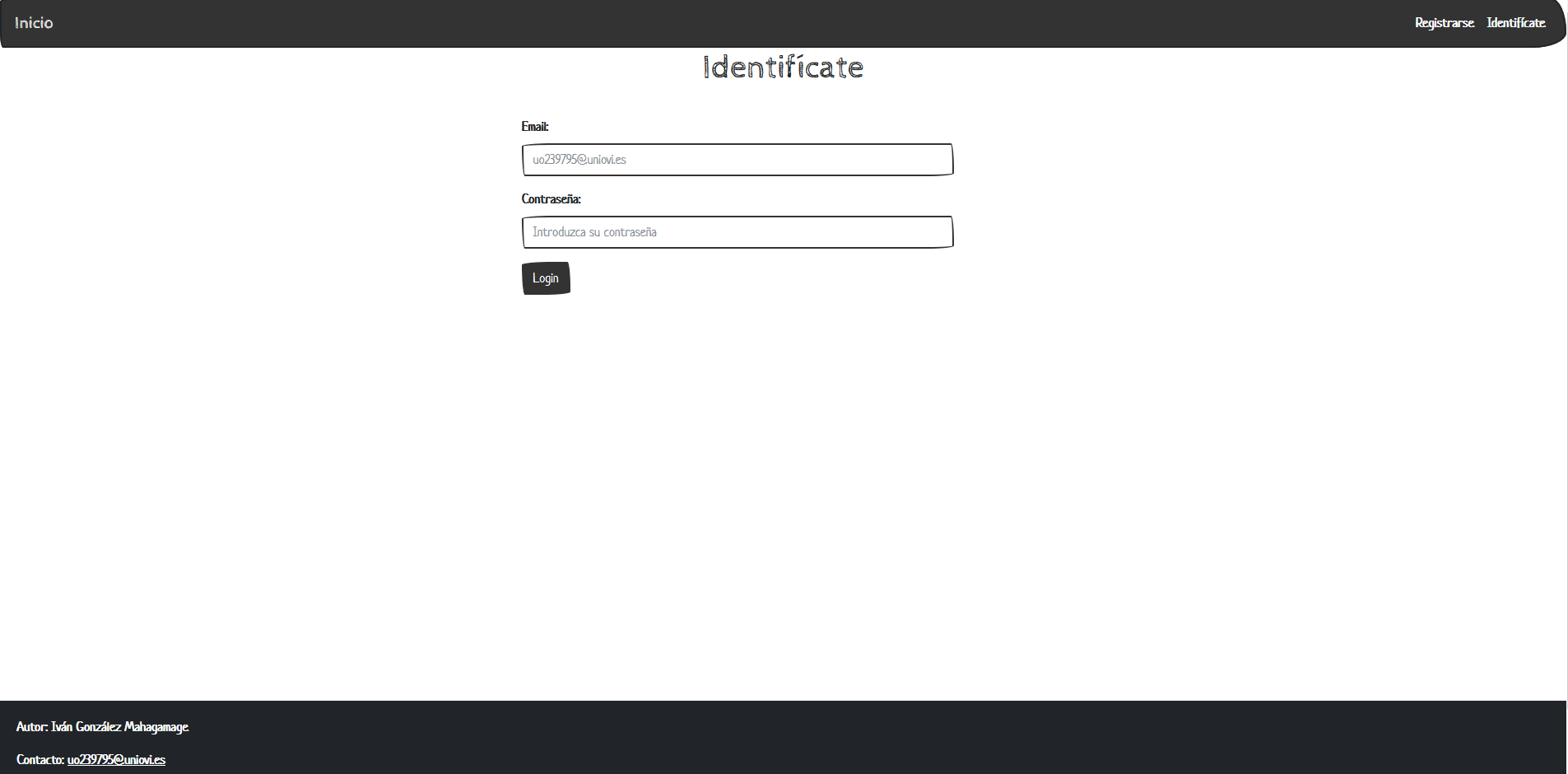
let answer = swig.renderFile('views/login.html', {});

res.send(answer);

app.get("logger").info('Usuario se intenta loguear');

});

Y el resultado es el siguiente:



Para procesar el formulario enviado creamos otro método en el fichero “rUsers.js”.

app.post("/login", function (req, res) {

let password = app.get("crypto").createHmac('sha256', app.get('key'))

.update(req.body.password).digest('hex');

let textSearch = {

email: req.body.email,

password: password

};

autoLogin(textSearch, req, res);

});

### Usuario registrado: listar todos los usuarios de la aplicación

Lo primero es crear una nueva vista denominada “list” en el directorio “views/users”. Esta vista es el resultado final de la aplicación por lo que contiene casos de usos posteriores.

{% extends "../templates/templatePagination.html" %}

{% block titulo %} Lista de Usuarios - UO239795 {% endblock %}

{% block main %}

<h1>Esta es una zona privada la web</h1>

<h2 align="center">

<span> Usuario Autenticado como :</span>

<b>{{ user }}</b>

</h2>

<h1>Lista de usuarios</h1>

<table class="table table-hover">

<thead class="table-dark">

<tr>

<th scope="col">Nombre</th>

<th scope="col">Apellidos</th>

<th scope="col">Email</th>

<th scope="col">Enviar invitación</th>

</tr>

</thead>

<tbody>

{% for user in users %}

<tr class="table-light">

<td>{{ user.name }}</td>

<td>{{ user.surName }}</td>

<td>{{ user.email }}</td>

<td>

{% if user.request == undefined %}

<form method="post" action="/send/{{ user.\_id.toString() }}">

<input type="submit" class="btn btn-success" value="Enviar solicitud"/>

</form>

{% endif %}

{% if user.request !== undefined && user.request.status == "ACCEPTED"%}

<span class="text-success">Solicitud aceptada</span>

{% endif %}

{% if user.request !== undefined && user.request.status == "SENT"%}

<span type="button" class="text-info" disabled="disable">Solicitud enviada</span>

{% endif %}

</td>

</tr>

{% endfor %}

</tbody>

</table>

<div id="error"></div>

<div id="success"></div>

<h2>Buscar usuario</h2>

<form class="navbar-form" action="/list">

<div class="form-group">

<input id="searchText" name="searchText" type="text" class="form-control" size="50"

placeholder="Buscar por email o nombre o apellido del alumno"/>

</div>

<button type="submit" class="btn btn-primary">Buscar</button>

</form>

<script src="/script/showMessage.js"></script>

<script>

message.showErrors("error");

message.showInfo("success");

</script>

{% endblock %}

Lo siguiente es añadir un nuevo método en el archivo “rUsers.js” para poder mostrar esta vista. Como este método es algo complicado vamos a explicarlo por partes.

* Primero guardamos la página actual en la que se encuentra el usuario y cargamos todos los usuarios de la aplicación.

let pg = parseInt(req.query.pg);

if (req.query.pg == null) {

pg = 1;

}

* Después calculamos el número de páginas totales que tendrá la lista de usuarios.

function calculatePgLast(value) {

let pgLast = value / 5;

if (value % 5 > 0) {

return pgLast + 1;

} else if (value == 0) {

return 0;

}

}

* Por último, filtramos los usuarios en grupos de 5 usando la función de JavaScript “slice”, esto nos ayudara en el siguiente caso de uso.

let i = (pg - 1) \* 5;

let answer = swig.renderFile('views/users/list.html', {

users: users.slice(i, i + 5),

pgCurrent: pg,

pgLast: calculatePgLast(users.length),

user: req.session.user

});

El método resultante sería el siguiente.

app.get("/list", function (req, res) {

app.get("logger").info('El usuario ' + req.session.user + " lista los usuarios de la aplicación");

let pg = parseInt(req.query.pg);

if (req.query.pg == null) {

pg = 1;

}

repository.getElements({}, "users", function (users) {

if (users == null) {

res.send("Error al listar");

} else {

let i = (pg - 1) \* 5;

let answer = swig.renderFile('views/users/list.html', {

users: users.slice(i, i + 5),

pgCurrent: pg,

pgLast: calculatePgLast(users.length),

user: req.session.user

});

res.send(answer);

}

});

});

Este método será modificado en casos siguientes para aumentar su funcionalidad y cumplir dichos casos.

### Usuario registrado: buscar entre todos los usuarios de la aplicación

Para este caso de uso también se usará la vista “list.html” por lo que se omite su código. Para poder realizar este filtrado, se va aumentar la funcionalidad del método usado en el anterior caso de uso.

* Se ha creado un nuevo método, denominado “createQuery”, con la consulta ya preparada en el fichero “repository.js”. Se ha realizado en esta clase para intentar respetar la separación entre la capa de lógica y la capa de persistencia de la aplicación.

createQuery(req) {

let textSearch = {

email: {

$ne: req.session.user

}

};

if (req.query.searchText != null) {

let searchText = req.query.searchText;

textSearch = {

$and: [{

email: {

$ne: req.session.user

}

},

{

$or: [{

email: {

$regex: ".\*" + searchText + ".\*"

}

},

{

name: {

$regex: ".\*" + searchText + ".\*"

}

},

{

surName: {

$regex: ".\*" + searchText + ".\*"

}

}

]

}

]

};

}

return textSearch;

}

* Modificamos el método del fichero “rUsers.js” que lista los usuarios para que use este nuevo método.



### Usuario registrado: enviar una invitación de amistad a un usuario

Para este caso de uso volvemos a utilizar la vista “list.html”. También vamos a modificar la función de “rUsers.js” que lista los usuarios.

* Una vez que recoja todos los usuarios, la aplicación lista todas las peticiones cuyo emisor sea el usuario en sesión.

let email = {

email: req.session.user

};

repository.getElements(email, "users", function (user) {

let request = {

sender: user[0].\_id.toString()

};

repository.getElements(request, "requests", function (requests) {

…

}

* Cuando la aplicación tiene las dos colecciones las recorre de la siguiente forma, por cada uno de los usuarios, recorre la lista de petición comprobando si el receptor es dicho usuario. En caso afirmativo modifica su JSON en tiempo de ejecución para añadirle el campo “request” que contendrá la petición.

let i = 0;

for (; i < users.length; i++) {

for (let j = 0; j < requests.length; j++) {

if (requests[j].receiver == users[i].\_id.toString()) {

users[i].request = requests[j];

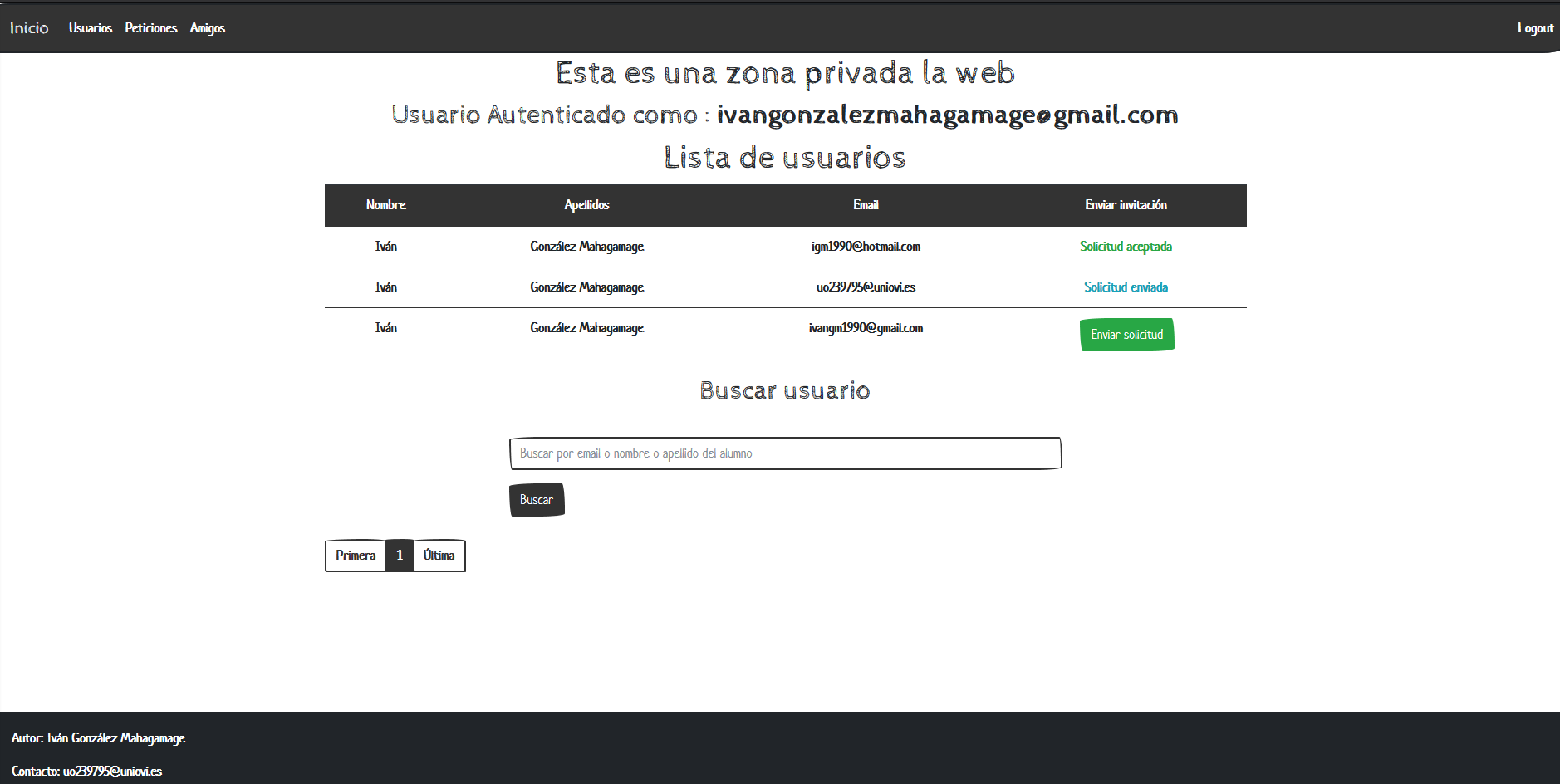
break;

}

}

}

La vista resultante de todos estos pasos seria la siguiente:



En la imagen se ven las tres situaciones que se pueden dar para este caso de uso:

* El usuario de la lista no tiene ninguna petición con el usuario que está en sesión por lo que se le muestra un botón para enviar y crear esa petición.
* El usuario de la lista tiene una petición con el usuario en sesión, pero este todavía no la ha aceptado por eso se le muestra el mensaje de “Solicitud enviada”.
* El usuario de la lista tiene una petición con el usuario en sesión y esta ha sido aceptada por lo que se muestra el mensaje de “Solicitud aceptada”.

### Usuario registrado: listar las invitaciones de amistad recibidas

En este caso de usos se ha creado la vista “receiver.html” en el directorio “views/requests”.

{% extends "../templates/templatePagination.html" %}

{% block titulo %} Peticiones Recibidas - UO239795 {% endblock %}

{% block main %}

<h1>Peticiones recibidas</h1>

<table class="table table-hover">

<thead class="table-dark">

<tr>

<th scope="col">Nombre</th>

<th scope="col">Apellidos</th>

<th scope="col">Email</th>

<th scope="col"></th>

</tr>

</thead>

<tbody>

{% for user in users %}

<tr class="table-light">

<td>{{ user.name }}</td>

<td>{{ user.surName }}</td>

<td>{{ user.email }}</td>

<td>

<form method="post" action="/accepted/{{ user.request }}">

<input type="submit" class="btn btn-success" value="Aceptar invitación"/>

</form>

</td>

</tr>

{% endfor %}

</tbody>

</table>

{% endblock %}

Dentro del fichero “rUsers.js” se ha creado una función genérica denominada “searchPeople” con el código necesario para tanto para listar las peticiones de amistad recibidas como las amistades. Dado que el código para estas dos funciones era muy similar se ha optado por esta solución por eficiencia y claridad en el código.

function searchPeople(req, res, status, view) {

let pg = parseInt(req.query.pg);

if (req.query.pg == null) {

pg = 1;

}

repository.getElements({}, "users", function (users) {

if (users == null) {

res.send("Error al listar");

} else {

let email = {

email: req.session.user

};

repository.getElements(email, "users", function (user) {

let request = {

receiver: user[0].\_id.toString(),

status: status

};

repository.getElements(request, "requests", function (requests) {

let collection = users.filter(function (user) {

for (let i = 0; i < requests.length; i++) {

if (user.\_id.toString() == requests[i].sender) {

user.request = requests[i].\_id.toString();

return true;

}

}

return false;

});

let limit = (pg - 1) \* 5;

let answer = swig.renderFile(view, {

users: collection.slice(limit, limit + 5),

pgCurrent: pg,

pgLast: calculatePgLast(collection.length)

});

res.send(answer);

});

});

}

});

}

Para este caso en particular, creamos un método para llamar a este método indicando el estado de la petición debe ser “SENT” (enviada) y la vista “receiver”).

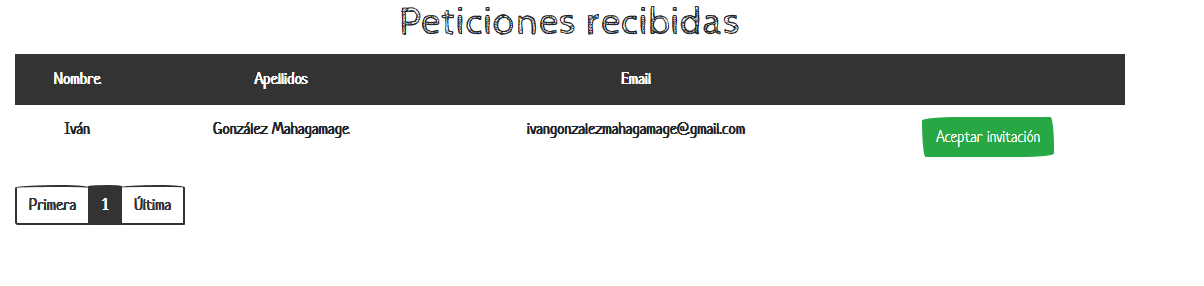
app.get("/requests", function (req, res) {

app.get("logger").info('El usuario ' + req.session.user + " lista su peticiones de amistades recibidas");

searchPeople(req, res, "SENT", 'views/requests/receiver.html')

});

El resultado sería el siguiente:



### Usuario registrado: aceptar una invitación recibida

### Usuario registrado: listar los usuarios amigos

## Servicios web – Implementación de la API de Servicios Web REST

### Identificarse con usuario – token

### Usuario identificado: listar todos los amigos

### Usuario identificado: Crear un mensaje

### Usuario identificado: Obtener mis mensajes de una “conversación”

### Usuario identificado: Marcar mensaje como leído

## Cliente – Aplicación jQuery

### Autentificación del usuario

### Mostrar la lista de amigos

### Mostrar los mensajes

### Crear mensaje

### Marcar mensajes como leídos de forma automática

### Mostrar el número de mensajes sin leer

### Ordenar la lista de amigos por último mensaje

# Prueba Unitarias

## Configuración de las pruebas y utilidades

## Registro de Usuario con datos válidos.

## Registro de Usuario con datos inválidos (repetición de contraseña invalida).



## Inicio de sesión con datos válidos.

## Inicio de sesión con datos inválidos.



## Acceso al listado de usuarios desde un usuario en sesión.

## Intento de acceso con URL desde un usuario no identificado al listado de usuarios desde un usuario en sesión.



## Realizar una búsqueda valida en el listado de usuarios desde un usuario en sesión.

## Intento de acceso con URL a la búsqueda de usuarios desde un usuario no identificado.



## Enviar una invitación de amistad a un usuario de forma valida.

## Enviar una invitación de amistad a un usuario al que ya le habíamos invitado la invitación previamente.



## Listar las invitaciones recibidas por un usuario.



## Aceptar una invitación recibida.



## Listar los amigos de un usuario, realizar la comprobación con una lista que al menos tenga un amigo.

## Inicio de sesión con datos válidos.

## Inicio de sesión con datos inválidos (usuario no existente en la aplicación).

## Acceder a la lista de amigos de un usuario, que al menos tenga tres amigos.

## Acceder a la lista de amigos de un usuario, y realizar un filtrado para encontrar a un amigo concreto, el nombre a buscar debe coincidir con el de un amigo.

## Acceder a la lista de mensajes de un amigo “chat”, la lista debe contener al menos tres mensajes.

## Acceder a la lista de mensajes de un amigo “chat” y crear un nuevo mensaje, validar que el mensaje aparece en la lista de mensajes.

## Identificarse en la aplicación y enviar un mensaje a un amigo, validar que el mensaje enviado aparece en el chat. Identificarse después con el usuario que recibido el mensaje y validar que tiene un mensaje sin leer, entrar en el chat y comprobar que el mensaje pasa a tener el estado leído.

## Identificarse en la aplicación y enviar tres mensajes a un amigo, validar que los mensajes enviados aparecen en el chat. Identificarse después con el usuario que recibido el mensaje y validar que el número de mensajes sin leer aparece en la propia lista de amigos.

## Identificarse con un usuario A que al menos tenga 3 amigos, ir al chat del ultimo amigo de la lista y enviarle un mensaje, volver a la lista de amigos y comprobar que el usuario al que se le ha enviado el mensaje este en primera posición. Identificarse con el usuario B y enviarle un mensaje al usuario A. Volver a identificarse con el usuario A y ver que el usuario que acaba de mandarle el mensaje es el primero en su lista de amigos.