12-5-2018

Iván González Mahagamage

Universisdad de oviedo

Entregable 1

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](#_Toc513411148)

[0. Aspectos comunes 4](#_Toc513411149)

[0.1. Colecciones de la aplicación. 4](#_Toc513411150)

[0.2. Creación del log de actividad. 5](#_Toc513411151)

[0.3. Gestor de base de datos 5](#_Toc513411152)

[0.4. Plantillas 6](#_Toc513411153)

[0.5. Configuración principal de la aplicación 11](#_Toc513411154)

[1. Aplicación web 13](#_Toc513411155)

[1.1. Público: registrarse como usuario 13](#_Toc513411156)

[1.2. Público: iniciar sesión 13](#_Toc513411157)

[1.3. Usuario registrado: listar todos los usuarios de la aplicación 13](#_Toc513411158)

[1.4. Usuario registrado: buscar entre todos los usuarios de la aplicación 13](#_Toc513411159)

[1.5. Usuario registrado: enviar una invitación de amistad a un usuario 13](#_Toc513411160)

[1.6. Usuario registrado: listar las invitaciones de amistad recibidas 13](#_Toc513411161)

[1.7. Usuario registrado: aceptar una invitación recibida 13](#_Toc513411162)

[1.8. Usuario registrado: listar los usuarios amigos 13](#_Toc513411163)

[2. Servicios web – Implementación de la API de Servicios Web REST 13](#_Toc513411164)

[2.1. Identificarse con usuario – token 13](#_Toc513411165)

[2.2. Usuario identificado: listar todos los amigos 13](#_Toc513411166)

[2.3. Usuario identificado: Crear un mensaje 13](#_Toc513411167)

[2.4. Usuario identificado: Obtener mis mensajes de una “conversación” 13](#_Toc513411168)

[2.5. Usuario identificado: Marcar mensaje como leído 13](#_Toc513411169)

[3. Cliente – Aplicación jQuery 13](#_Toc513411170)

[3.1. Autentificación del usuario 13](#_Toc513411171)

[3.2. Mostrar la lista de amigos 13](#_Toc513411172)

[3.3. Mostrar los mensajes 13](#_Toc513411173)

[3.4. Crear mensaje 13](#_Toc513411174)

[3.5. Marcar mensajes como leídos de forma automática 14](#_Toc513411175)

[3.6. Mostrar el número de mensajes sin leer 14](#_Toc513411176)

[3.7. Ordenar la lista de amigos por último mensaje 14](#_Toc513411177)

[Prueba Unitarias 14](#_Toc513411178)

[0. Configuración de las pruebas y utilidades 14](#_Toc513411179)

[1.1. Registro de Usuario con datos válidos. 14](#_Toc513411180)

[1.2. Registro de Usuario con datos inválidos (repetición de contraseña invalida). 14](#_Toc513411181)

[2.1. Inicio de sesión con datos válidos. 14](#_Toc513411183)

[2.2. Inicio de sesión con datos inválidos. 14](#_Toc513411184)

[3.1. Acceso al listado de usuarios desde un usuario en sesión. 14](#_Toc513411186)

[3.2. Intento de acceso con URL desde un usuario no identificado al listado de usuarios desde un usuario en sesión. 14](#_Toc513411187)

[4.1. Realizar una búsqueda valida en el listado de usuarios desde un usuario en sesión. 14](#_Toc513411189)

[4.2. Intento de acceso con URL a la búsqueda de usuarios desde un usuario no identificado. 14](#_Toc513411190)

[5.1. Enviar una invitación de amistad a un usuario de forma valida. 14](#_Toc513411192)

[5.2. Enviar una invitación de amistad a un usuario al que ya le habíamos invitado la invitación previamente. 14](#_Toc513411193)

[6.1. Listar las invitaciones recibidas por un usuario. 14](#_Toc513411195)

[7.1. Aceptar una invitación recibida. 15](#_Toc513411197)

[8.1. Listar los amigos de un usuario, realizar la comprobación con una lista que al menos tenga un amigo. 15](#_Toc513411199)

[C1.1. Inicio de sesión con datos válidos. 15](#_Toc513411200)

[C1.2. Inicio de sesión con datos inválidos (usuario no existente en la aplicación). 15](#_Toc513411201)

[C2.1. Acceder a la lista de amigos de un usuario, que al menos tenga tres amigos. 15](#_Toc513411202)

[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. 15](#_Toc513411203)

[C3.1. Acceder a la lista de mensajes de un amigo “chat”, la lista debe contener al menos tres mensajes. 15](#_Toc513411204)

[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. 15](#_Toc513411205)

[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. 15](#_Toc513411206)

[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. 15](#_Toc513411207)

[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. 16](#_Toc513411208)

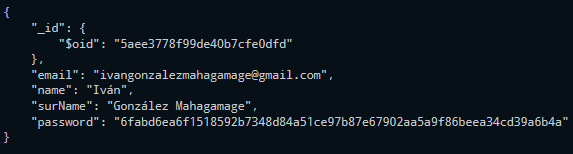
# Implementación

## Aspectos comunes

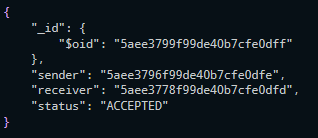
### Colecciones de la aplicación.

Para esta aplicación he creado tres colecciones para su funcionamiento:

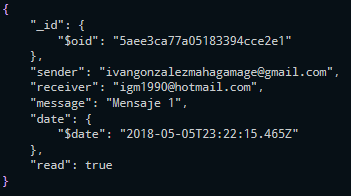
* Users: en esta colección se guarda la información de los usuarios registrados. Se guarda el nombre, apellidos, email y contraseña del usuario. La estructura que siguen los elementos es la siguiente:



* 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”.

{% 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 que los datos introducidos 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) {

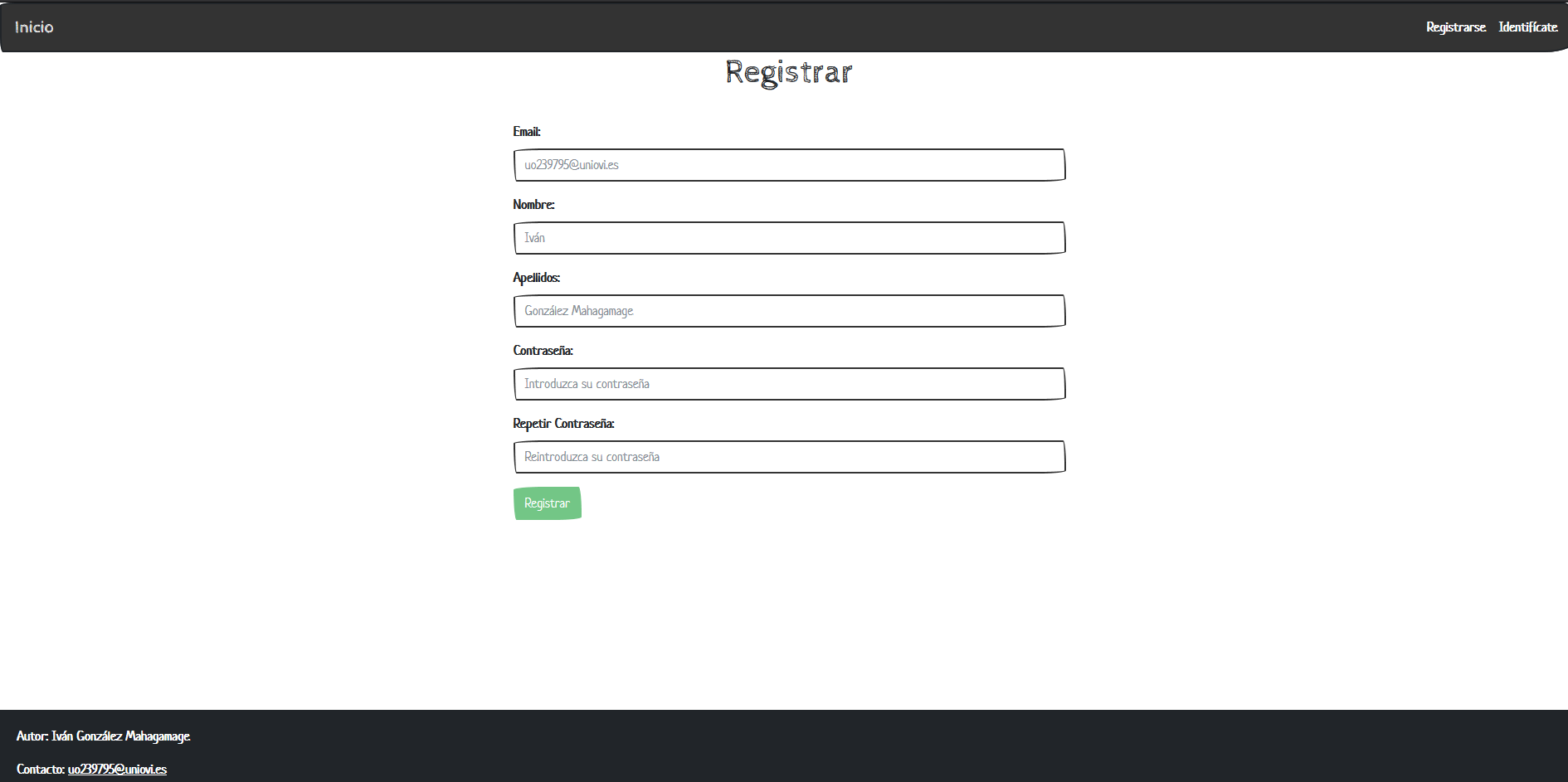
let answer = swig.renderFile('views/signup.html', {});

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 procesas 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”.

{% 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”

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

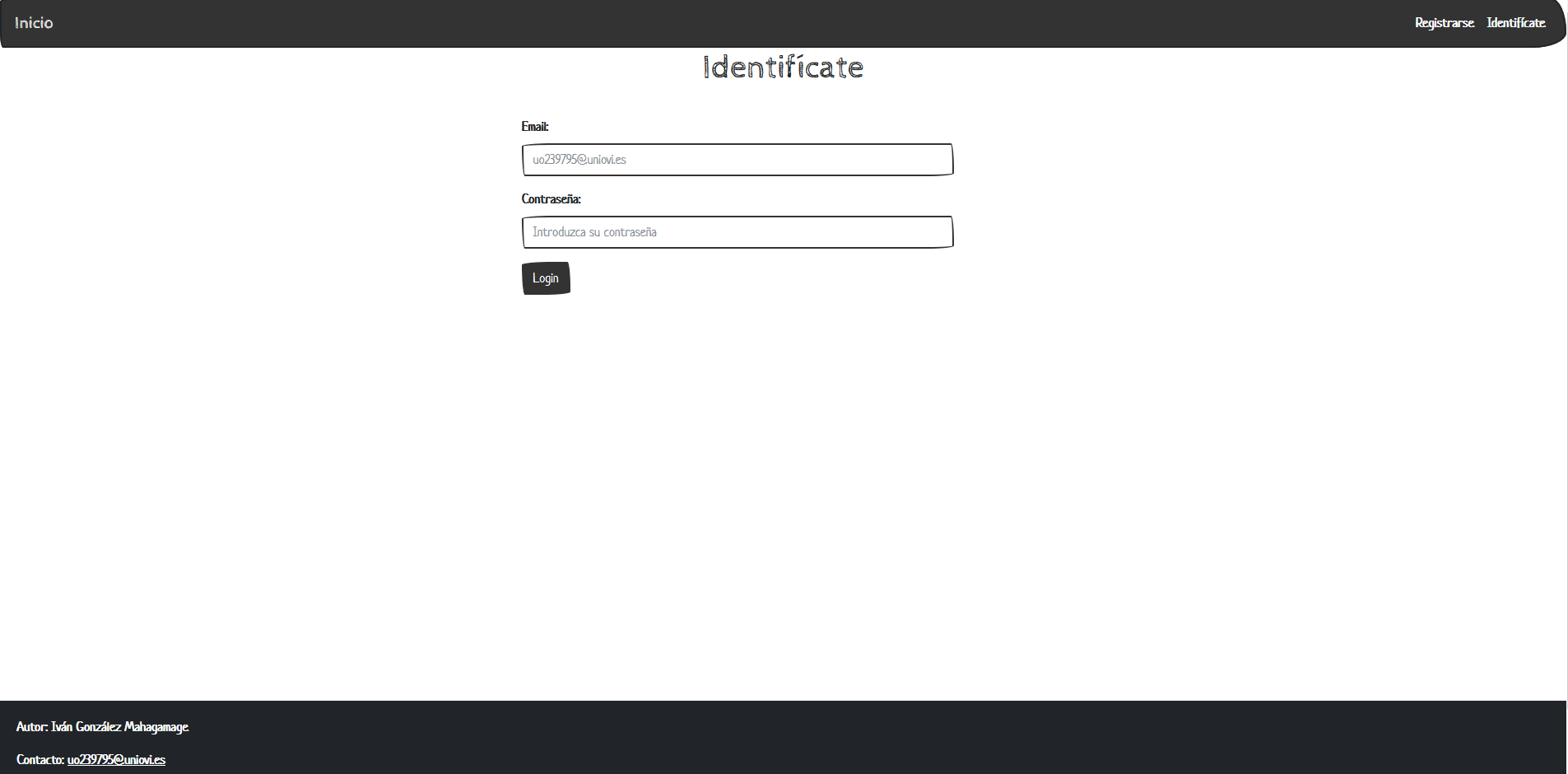
let answer = swig.renderFile('views/login.html', {});

res.send(answer);

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

});

Y el resultado es el siguiente:



Para procesar el formulario 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

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

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

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

### 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.