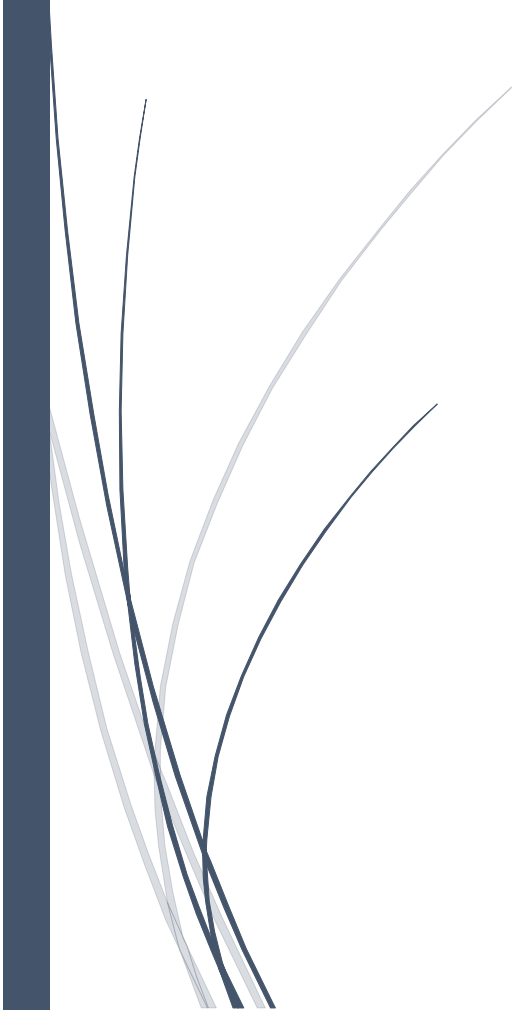
12-5-2018

Entregable 2

Sistemas distribuidos e internet



Iván González Mahagamage
UNIVERSISDAD DE OVIEDO

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

0. Aspectos comunes

0.0. 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:


```
{
    "_id": {
        "$oid": "5aee3778f99de40b7cfe0dfd"
    },
    "email": "ivangonzalezmahagamage@gmail.com",
    "name": "Iván",
    "surName": "González Mahagamage",
    "password": "6fabd6ea6f1518592b7348d84a51ce97b87e67902aa5a9f86beea34cd39a6b4a"
}
```


- 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.


```
{
    "_id": {
        "$oid": "5aee3799f99de40b7cfe0dff"
    },
    "sender": "5aee3796f99de40b7cfe0dfe",
    "receiver": "5aee3778f99de40b7cfe0dfd",
    "status": "ACCEPTED"
}
```


- 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.


```
{
    "_id": {
        "$oid": "5aee3ca77a05183394cce2e1"
    },
    "sender": "ivangonzalezmahagamage@gmail.com",
    "receiver": "igm1990@hotmail.com",
    "message": "Mensaje 1",
    "date": {
        "$date": "2018-05-05T23:22:15.465Z"
    },
    "read": true
}
```

0.1. 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
```

-

0.2. Gestor de base de datos

Para simplificar y crear las mínimas funciones 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);
```

0.3. 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);
    }
}
```

0.4. Configuración principal de la aplicación

0.4.1. 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);
```

0.4.2. 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"
  }
}
```

0.4.3. 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.

1. Aplicación web

1.0. 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.

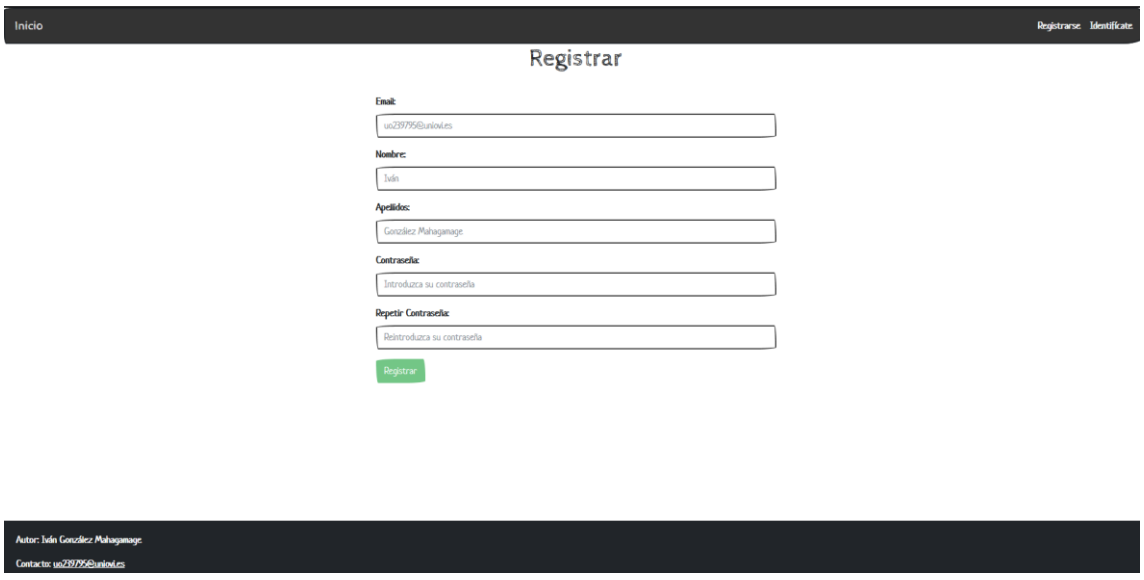
```
"use strict";

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 una función 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 varias funciones en “rUsers.js”.

a. Creamos una función 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);
    }
```

b. Creamos una función 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.");
            }
        });
    });
```

c. Si el registro se ha realzado correctamente, se autologuea al usuario para que empiece a usar la aplicación. Para ello creamos una nueva función 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");
            }
        });
    }
```

1.1. Público: iniciar sesión

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

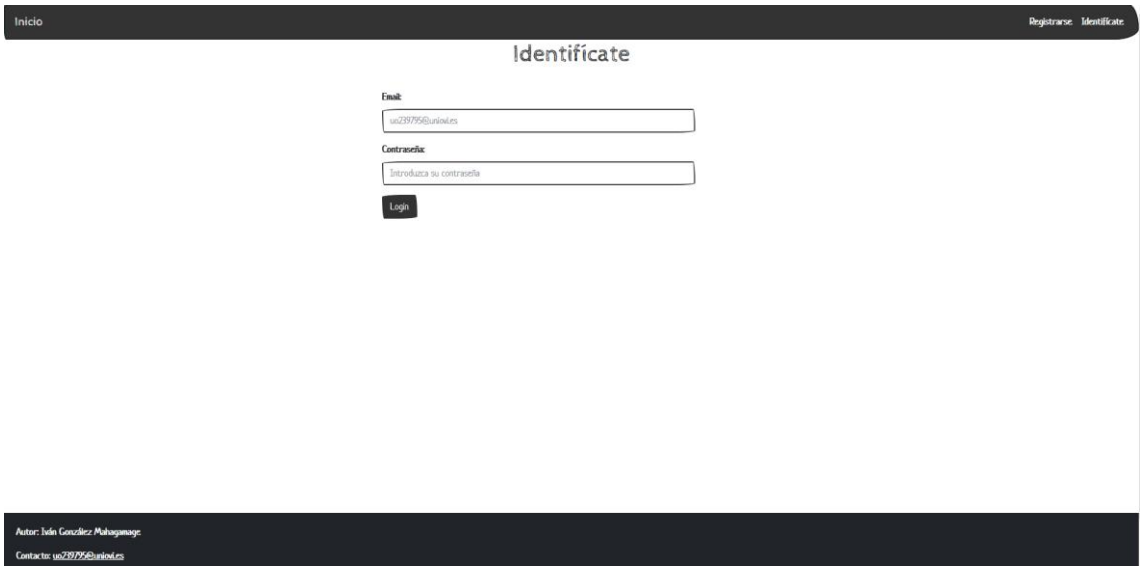
```
{% extends "templates/templateLogin.html" %}
{% block titulo %} Login - UO239795 {% endblock %}
{% block main %}
<h1>Identifícate</h1>
<form class="form-horizontal" method="post" action="/login">
    <fieldset>
        <div class="form-group">
            <label class="control-label col-sm-2"
for="email">Email:</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 función 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);
    });
```

1.2. 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 nueva función en el archivo “rUsers.js” para poder mostrar esta vista. Como esta función 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
});
```

La función 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 función será modificado en casos siguientes para aumentar su funcionalidad y cumplir dichos casos.

1.3. 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 de la función usado en el anterior caso de uso.

- Se ha creado una nueva función, 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 la función del fichero “rUsers.js” que lista los usuarios para que use este nueva función.

```
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(repository.createQuery(req), "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);
        }
    });
});
```

1.4. 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.

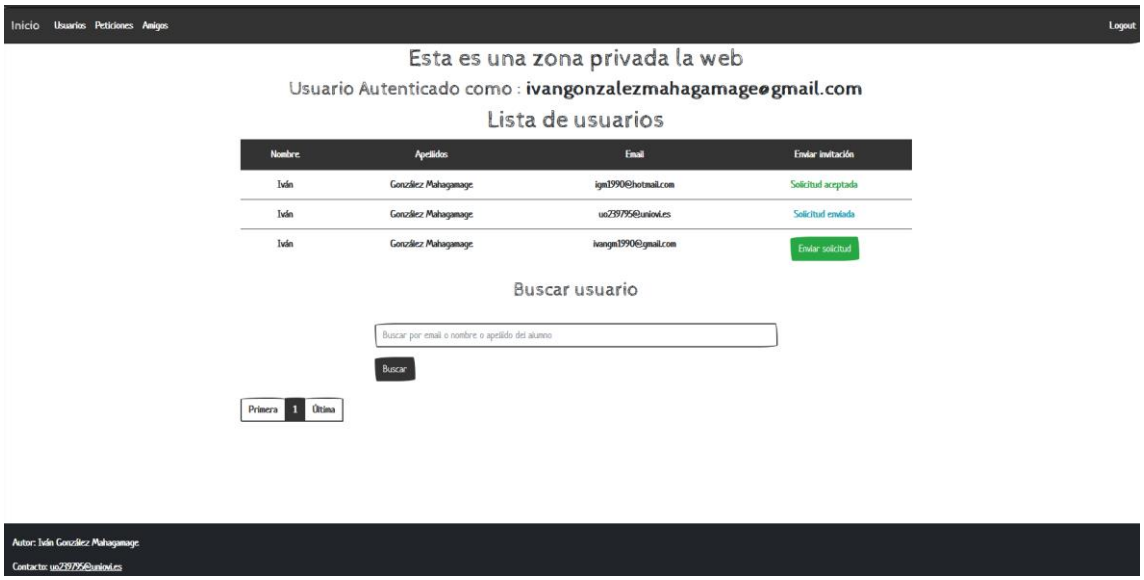
- Una vez que recoja todos los usuarios, la aplicación lista todas las peticiones cuyo emisor sea el usuario en sesión.

```
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”.

Cuando se envia una petición, al pulsar el botón “Enviar petición” se llama a una función creada en el fichero “rRequests” para su creación.

```
app.post('/send/:id', function (req, res) {
    let email = {
        email: req.session.user
    };
    repository.getElements(email, "users", function (users) {
        if (users == null || users.length == 0) {
            res.redirect("/list?error=Usuario no existe");
        } else {
            let request = {
                sender: users[0]._id.toString(),
                receiver: req.params.id,
                status: "SENT"
             };
        repository.getElements(request, "requests", function (requests) {
             if (requests == null || requests.length == 0) {
                 repository.addElement(request, "requests", function () {
                     app.get("logger").info('El usuario ' +
req.session.user + " ha enviado una petición de amistad");
```

```
            res.redirect("/list?success=Petición enviada
correctamente");
              });
             } else {
                app.get("logger").error('Error al enviar una petición
de amistad por parte del usuario ' + req.session.user);
               res.redirect("/list?error=Petición ya enviada");
            }
         });
       }
   });
});
```

1.5. 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

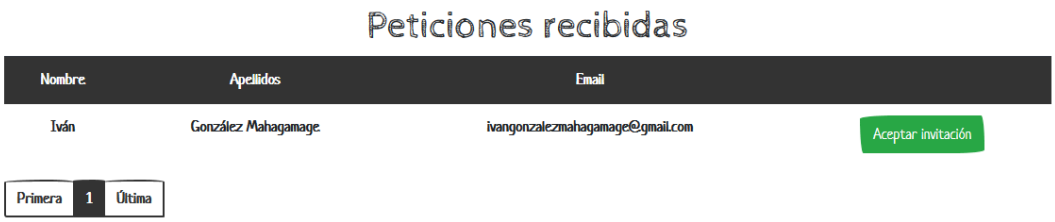
amistades. Dado que el código para estas dos funciones era muy similar se ha optado por esta solución por eficiencia y claridad en el código.

```
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 una función para llamar a este función 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:



1.6. Usuario registrado: aceptar una invitación recibida

Para este caso de uso se utiliza tambien la vista “receiver.html” en el directorio “views/requests” mostrada en el caso anterior.

Para aceptar la petición el usuario tiene que pulsar el botón “Aceptar invitación”. Al accionarlo se llama a una función creado en el fichero “rRequests.js”.

```
app.post('/accepted/:id', function (req, res) {
    let request = {
        "_id": new ObjectId(req.params.id)
    };
    repository.getElements(request, "requests", function (requests) {
        if (requests != null || requests.length > 0) {
            acceptedRequest(requests[0].sender, requests[0].receiver);
            acceptedRequest(requests[0].receiver, requests[0].sender);
            res.redirect("/list?success=Petición aceptada
correctamente");
            app.get("logger").error('La peticion con id ' + req.params.id
+ " ha sido aceptada");
        } else {
            app.get("logger").info('La peticion con id ' + req.params.id
+ " no se ha podido aceptar");
            res.redirect("/list?error=No se ha podido aceptar la
peticion");
        }
    });
});
```

Su funcionamiento se hace en dos pasos:

- Actualiza el estado de la petición recibida al usuario en sesión.
- Busca si existe una petición en sentido inverso, es decir si el usuario en sesión ha enviado una petición al otro usuario.

- Si existe también la actualiza su estado.
- Si no existe, crea una nueva petición y pone su estado en aceptada.

Para hacer esto se ha creado una función denominada “acceptedRequest”.

```
function acceptedRequest(senderParam, receiverParam) {
    let request = {
        sender: senderParam,
        receiver: receiverParam
    };
    repository.getElements(request, "requests", function (requests) {
        let updateRequest = {
            sender: senderParam,
            receiver: receiverParam,
            status: "ACCEPTED"
        };
        if (requests == null || requests.length == 0) {
            repository.addElement(updateRequest, "requests", function
(result) {
                if (result == null) {
                    res.redirect("/list?error=No se ha podido aceptar la
peticion");
                    app.get("logger").error('No se ha podido aceptar la
peticion');
                }
            });
        } else {
            repository.updateElement(requests[0], updateRequest,
"requests", function (result) {
                if (result == null) {
                    res.redirect("/list?error=No se ha podido aceptar la
peticion");
                    app.get("logger").error('No se ha podido aceptar la
peticion');
                }
            });
        }
    });
}
```

1.7. Usuario registrado: listar los usuarios amigos

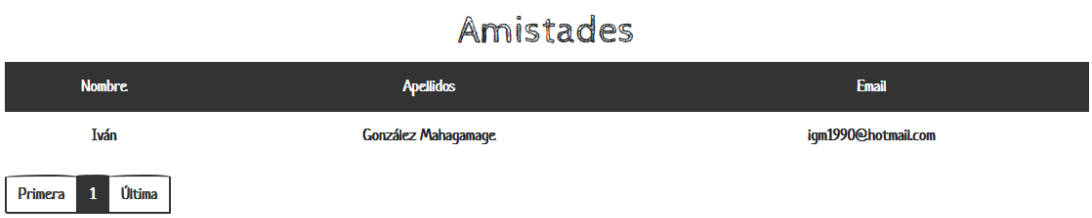
Para este caso de uso se ha implementado la vista “friends.html” en el directorio “views/users”.

```
{% extends "../templates/templatePagination.html" %}
{% block titulo %} Amistades - UO239795 {% endblock %}
{% block main %}
<h1>Amistades</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>
    </tr>
    </thead>
    <tbody>
    {% for user in users %}
    <tr class="table-light">
        <td>{{ user.name }}</td>
        <td>{{ user.surName }}</td>
        <td>{{ user.email }}</td>
    </tr>
    {% endfor %}
    </tbody>
</table>
{% endblock %}
```

Para mostrar esta vista añadimos una nueva función al fichero “rUsers.js”.

```
app.get("/friends", function (req, res) {
    app.get("logger").info('El usuario ' + req.session.user + " lista sus
amigos");
    searchPeople(req, res, "ACCEPTED", 'views/users/friends.html')
});
```

El resultado sería el siguiente.



2. Servicios web – Implementación de la API de Servicios Web REST

Para la realización de esta parte se ha creado un nuevo fichero “api.js” dentro del directorio “routes”. En cada uno de los casos de uso se ha implementado una función para cumplir con ellos.

2.0. Identificarse con usuario – token

```
app.post("/api/login/", function (req, res) {
    let seguro = app.get("crypto").createHmac('sha256',
app.get('key')).update(req.body.password).digest('hex');
    let user = {
        email: req.body.email,
        password: seguro
    };
    repository.getElements(user, "users", function (users) {
        if (users == null || users.length == 0) {
            res.status(401);
            res.json({
                authenticated: false,
                error: 'Identificación no válida'
            });
            app.get("logger").error('Identificación no válida con token
en la API');
        } else {
            let token = app.get('jwt').sign({
                user: user.email,
                time: Date.now() / 1000
            }, "secreto");
            res.status(200);
            res.json({
                authenticated: true,
                token: token
            });
            app.get("logger").info('Identificación válida con token en la
API del usuario ' + user.email);
        }
    });
});
```

2.1. Usuario identificado: listar todos los amigos

Para este caso de uso se creado una función, denominada “getFriends”, a parte ya que gran parte de su funcionalidad se va a utilizar en los siguientes casos de uso.

```
function getFriends(res, email, functionCallBack) {
    repository.getElements({}, "users", function (users) {
        if (users == null || users.length == 0) {
            res.status(403); // Forbidden
            res.json({
                error: 'No hay usuarios'
```

```
            });
            app.get("logger").error('No existen usuarios');
        } else {
            repository.getElements(email, "users", function (user) {
                let request = {
                    receiver: user[0]._id.toString(),
                    status: "ACCEPTED"
                };
                repository.getElements(request, "requests", function
(requests) {
                    if (requests == null || requests.length == 0) {
                        res.status(403);
                        res.json({
                            error: 'El usuario ' + email.email + " no
tiene amistades."
                        });
                        app.get("logger").error('\'El usuario \' +
email.email + " no tiene amistades"');
                    } else {
                        let collection = users.filter(function (user) {
                            for (let i = 0; i < requests.length; i++) {
                                if (user._id.toString() ==
requests[i].sender) {
                                    delete user.password;
                                    return true;
                                }
                            }
                            return false;
                        });
                        functionCallBack(collection)
                    }
                });
            });
        }
    });
}
```

Ahora para implementar este caso de uso se ha creado la siguiente función.

```
app.get("/api/users/", function (req, res) {
    let email = {
        email: res.user
    };
    getFriends(res, email, function (emails) {
        res.status(200);
        res.send(JSON.stringify(emails));
        app.get("logger").info('Listando los amigos del usuario ' +
res.user + ' en la API');
```

```
    });
});
```

2.2. Usuario identificado: Crear un mensaje

```
app.post("/api/messages/", function (req, res) {
    let email = {
        email: res.user
    };
    getFriends(res, email, function (friends) {
        let friend = friends.filter(function (element) {
            return element.email == req.body.email;
        });
        if (friend.length == 0) {
            res.status(401);
            res.json({
                error: 'El receptor no es amigo'
            });
            app.get("logger").error('El usuario ' + req.body.email + ' no
es amigo del usuario ' + res.user);
        } else {
            let message = {
                sender: res.user,
                receiver: req.body.email,
                message: req.body.message,
                date: new Date(),
                read: false
            }
            repository.addElement(message, "messages", function (id) {
                if (id == null) {
                    res.status(501);
                    res.json({
                        error: 'Error en el servidor al crear el mensaje'
                    });
                    app.get("logger").error('Error al crear el mensaje
entre los usuarios ' + req.body.email + ' y ' + res.user);
                } else {
                    res.status(200);
                    res.send(JSON.stringify(message));
                    app.get("logger").info('Exito al crear el mensaje
entre los usuarios ' + req.body.email + ' y ' + res.user);
                }
            });
        }
    });
});
```

2.3. Usuario identificado: Obtener mis mensajes de una “conversación”

```
app.get("/api/messages/", function (req, res) {
    repository.getElements(repository.getMessagesBySenderAndReceiver(req,
res), "messages", function (conversation) {
        res.status(200);
        res.send(JSON.stringify(conversation));
        app.get("logger").info('Listando los mensajes entre los usuarios
' + res.user + " y " + req.params.email);
    });
});
```

2.4. Usuario identificado: Marcar mensaje como leído

```
app.put("/api/messages/:id", function (req, res) {
    var message = {
        "_id": new ObjectId(req.params.id)
    };
    repository.getElements(message, "messages", function (conversation) {
        if (conversation == null || conversation.length == 0) {
            res.status(403);
            res.json({
                error: 'No existe el mensaje'
            });
            app.get("logger").error('No existe el mensaje con id ' +
req.params.id);
        } else {
            if (conversation[0].receiver == res.user) {
                var updateMessage = {
                    sender: conversation[0].sender,
                    receiver: conversation[0].receiver,
                    message: conversation[0].message,
                    date: conversation[0].date,
                    read: true
                };
                repository.updateElement(conversation[0], updateMessage,
"messages", function (result) {
                    if (result == null) {
                        res.status(501);
                        res.json({
                            error: 'No se puede actualizar el mensaje'
                        });
                        app.get("logger").error('No se ha podido
actualizar el mensaje con id ' + req.params.id);
                    } else {
                        res.status(200);
                        res.json({
                            result: 'Mensaje marcado correctamente'
                        });
```

```
                        app.get("logger").info('Actualizado correctamente
el mensaje con id ' + req.params.id);
                    }
                });
            }
        }
    });
});
```

3. Cliente – Aplicación jQuery

3.0. Página principal

Primero hay que crear un fichero denominado “main.html” que se usara de base y se modificara en tiempo de ejecución, para ser exactos el elemento “main” será el que se modificara.

```
<!DOCTYPE html>
<html lang="es">

<head>
    <meta charset="UTF-8">
    <title>Cliente jQuery</title>
    <meta name="author" content="Iván González Mahagamage" />
    <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" />
    <link rel="stylesheet" type="text/css" href="/css/chat.css" />
    <script src="/script/jquery.js"></script>
    <script src="/script/popper.min.js"></script>
    <script src="/script/bootstrap.min.js"></script>
    <script src="/script/cookie.js"></script>
</head>

<body>
    <nav class="navbar navbar-expand-lg navbar-dark bg-primary">
        <a class="navbar-brand" href="/main.html">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">
            <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="/main.html?w=friends">Amigos</a>
                </li>
```

```
            </ul>
        </div>
    </nav>
    <main class="container">
    </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>
```

3.1. Autentificación del usuario

Este caso de uso usara el fichero “widget-login.html” que se encuentra en “public/widgets”.

```
<h1>Identifícate</h1>
<div id="widget-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" id="email" class="form-control"
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">
```

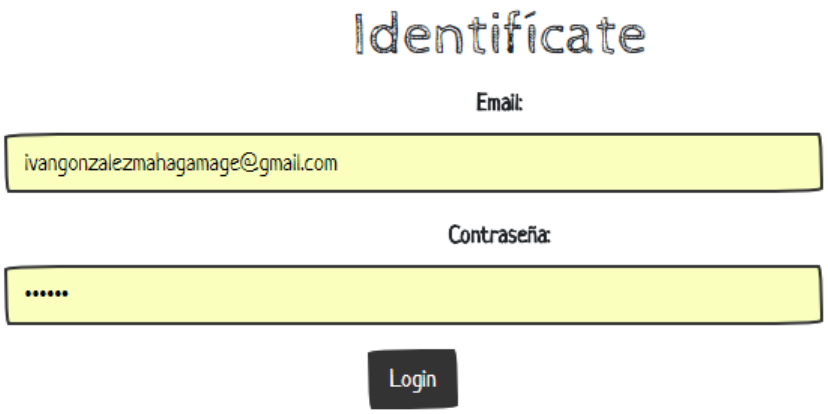
```
                <input type="password" id="password" class="form-control"
name="password" placeholder="Introduzca su contraseña" title="Introduzca
su contraseña"
                    pattern=".{2,}" required="required" />
            </div>
        </div>
        <div class="form-group">
            <div class="col-sm-offset-2 col-sm-10">
                <button type="submit" class="btn btn-primary" id="boton-
login">Login</button>
            </div>
        </div>
    </fieldset>
</div>
<script>
    window.history.pushState("", "", "/main.html?w=login");
    $("#boton-login").click(function () {
        $.ajax({
            url: URLbase + "/login",
            type: "POST",
            data: {
                email: $("#email").val(),
                password: $("#password").val()
            },
            dataType: 'json',
            success: function (res) {
                token = res.token;
                Cookies.set('email', $("#email").val());
                Cookies.set('token', res.token);
                $("main").load("widgets/widget-friends.html");
            },
            error: function (error) {
                Cookies.remove('token');
                $("#error").prepend("<div class='alert alert-
danger'>Usuario no encontrado</div>");
            }
        });
    });
</script>
```

Hay que añadir al fichero “main.html” un script para que realice la autentificación a través de la API de la aplicación y cargue cada uno de los “widgets” que usará la aplicación.

```
<script>
    var token;
    var URLbase = "http://localhost:8081/api";
    $("main").load("/widgets/widget-login.html");
    if (Cookies.get('token') != null) {
```

```
        token = Cookies.get('token');
        var url = new URL(window.location.href);
        var w = url.searchParams.get("w");
        console.log(w);
        switch (w) {
            case "friends":
                $("main").load("widgets/widget-friends.html");
                break;
            case "messages":
                $("main").load("widgets/widget-messages.html");
                break;
            default:
                $("main").load("widgets/widget-login.html");
                break;
        }
    }
</script>
```

El resultado de este script es el siguiente.



3.2. Mostrar la lista de amigos

Lo primero es crear un “widget” para este caso de uso, denominado “widget-friends.html”. Aquí se muestra su contenido html.

```
<h2 align="center">
    <span> Usuario Autenticado como :</span>
    <b id="user"></b>
</h2>
<h1>Amistades</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">Mensajes no leídos</th>
        </tr>
    </thead>
    <tbody id="bodyTable">
    </tbody>
</table>
<h2>Buscar usuario</h2>
<div class="navbar-form">
    <div class="form-group">
        <input id="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"
onclick="getFriends()">Buscar</button>
</div>
```

Ahora se va a explicar el script que utiliza este fichero para mostrar los amigos del usuario.

- GetFriends: llama a la API de la aplicación para cargar los amigos del usuario en sesión.

```
function getFriends() {
    $.ajax({
        url: URLbase + "/users",
        type: "GET",
        data: {},
        dataType: 'json',
        headers: {"token": token},
        success: function (res) {
            users = res;
            search();
            loadTable(users);
        },
        error: function (error) {
            $("main").load("widget-login.html");
        }
    });
}
```

- Search: recoge el parámetro por cual el usuario en sesión quiere filtrar sus amigos y realiza un filter en la lista de amigos, es decir los filtra mediante JavaScript, no MongoDB.

```
function search() {
    var param = $("#searchText").val();
    users = users.filter(function (user) {
        if (user.name.includes(param) || user.surName.includes(param) ||
user.email.includes(param)) {
            return true;
```

```
        }
        return false;
    });
}
```

- LoadTable: modifica la tabla donde se muestran las amistades.

```
function loadTable(users) {
    $("#bodyTable").empty();
    for (var i = 0; i < users.length; i++) {
        $("#bodyTable").append(
            "<tr class='table-light'>" +
            "<td>" + users[i].name + "</td>" +
            "<td>" + users[i].surName + "</td>" +
            "<td>" + users[i].email + "</td>" +
            "</tr>"
        );
    }
}
```

El resultado final de estos pasos sería el siguiente, en esta imagen se muestra la pantalla final con el resto de casos de uso implementado por eso se muestra información adicional.

3.3. Mostrar los mensajes

Para mostrar este caso de uso se ha implementado un “widget” adicional denominado “widget-friends.html” en el directorio “public/widgets”.

```
<h1>Chat con
    <span id="friend"></span>
</h1>
<div id="conversation" style="overflow-y: scroll; height: 600px">
</div>
<div>
    <input id="message" class="form-control" />
    <button id="send" type="button" class="btn btn-success"
onclick="sendMessage()">Enviar</button>
</div>
```

Al igual que en el caso anterior, se va proceder a explicar el script de funcionamiento de la página por partes para facilitar su comprensión.

- getMessages: devuelve todos los mensajes existentes entre el usuario en sesión y el usuario amigo indicado.

```
function getMessages() {
    $.ajax({
        url: URLbase + "/messages?email=" + email,
        type: "GET",
        data: {},
        dataType: 'json',
        headers: {
            "token": token
        },
        success: function (res) {
            var messages = res;
            loadMessages(messages);
        },
        error: function (error) {
            $("main").load("widgets/widget-login.html");
        }
    });
}
```

- loadMessages: modifica el contenido de la página para mostrar la conversación entre ambos usuarios.

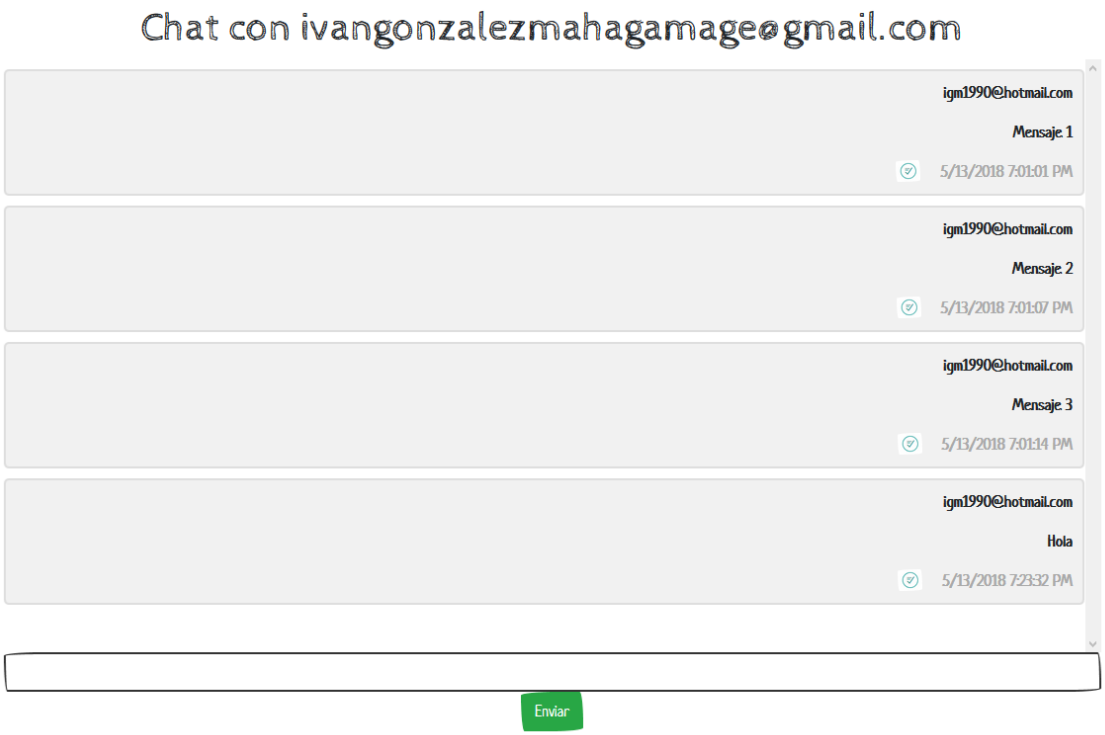
```
function loadMessages(messages) {
    $("#conversation").empty();
    for (var i = 0; i < messages.length; i++) {
        var message = "";
        if (messages[i].receiver == email) {
            message = "<div class='containerChat'>" +
                "<p class='text-right'>" + messages[i].sender + "</p>" +
                "<p class='text-right'>" + messages[i].message + "</p>" +
                "<span class='time-right'>" + timeStamp(new
Date(messages[i].date)) + "</span>";
            if (messages[i].read) {
                message += "<img src='img/check.png' style='width: 25px'
class='rounded float-right'>";
            }
            message += "</div>";
        } else {
            message = "<div class='containerChat darker'>" +
                "<p class='text-left'>" + messages[i].sender + "</p>" +
                "<p class='text-left'>" + messages[i].message + "</p>" +
                "<span class='time-left'>" + timeStamp(new
Date(messages[i].date)) + "</span>";
            if (messages[i].read) {
```

```
                message += "<img src='img/check.png' style='width: 25px'
class='rounded float-left'>";
            }
            message += "</div>";
        }
        $("#conversation").append(message);
    }
}
```

- timeStamp: es una función sacada de internet cuya única finalidad es mostrar de forma elegante la fecha de creación del mensaje.

```
function timeStamp(now) {
    // Create an array with the current month, day and time
    var date = [now.getMonth() + 1, now.getDate(), now.getFullYear()];

    // Create an array with the current hour, minute and second
    var time = [now.getHours(), now.getMinutes(), now.getSeconds()];

    // Determine AM or PM suffix based on the hour
    var suffix = (time[0] < 12) ? "AM" : "PM";

    // Convert hour from military time
    time[0] = (time[0] < 12) ? time[0] : time[0] - 12;

    // If hour is 0, set it to 12
    time[0] = time[0] || 12;

    // If seconds and minutes are less than 10, add a zero
    for (var i = 1; i < 3; i++) {
        if (time[i] < 10) {
            time[i] = "0" + time[i];
        }
    }
    // Return the formatted string
    return date.join("/") + " " + time.join(":") + " " + suffix;
}
```

El resultado final de todos estos pasos sería el siguiente:



Para llamar a esta función se ha creado la función “loadConversation” en “widget-friends.html”.

```
function loadConversation(email) {
    Cookies.set('friend', email);
    $("main").load("widgets/widget-messages.html");
}
```

Esta función es llamada en el evento onclick del nombre en la lista de amigos, es decir cuando el usuario en sesión hace clic encima del nombre de un amigo.

```
"<td><a onclick=loadConversation('" + users[i].email + "')>" +
users[i].name + "</a></td>" +
```

3.4. Crear mensaje

Para su implantación, se ha añadido una nueva función al “widget” “widget-messages.html” denominada “sendMessage”.

```
function sendMessage() {
    if ($("#message").val() != '') {
        $.ajax({
            url: URLbase + "/messages",
            type: "POST",
            data: {
                email: email,
                message: $("#message").val(),
                token: token
```

```
            },
            dataType: 'json',
            success: function (respuesta) {
                getMessages();
                $("#message").val('');
            },
            error: function (error) {
                $("main").append("<div class='alert alert-danger'>Error
enviando el mensaje</div>");
            }
        });
    }
}
```

3.5. Marcar mensajes como leídos de forma automática

Para su implantación, se ha añadido una nueva función al "widget" "widget-messages.html" denominada "readMessage".

```
function readMessage(id) {
    $.ajax({
        url: URLbase + "/messages/" + id,
        type: "PUT",
        data: {},
        dataType: 'json',
        headers: {
            "token": token
        },
        success: function (res) {},
        error: function (error) {
            $("main").append("<div class='alert alert-danger'>Error al
marcar el mensaje</div>");
        }
    });
}
```

Y añadir una llamada a esta función dentro de loadMessages.

```
else {
    readMessage(messages[i]._id.toString());
    message = "<div class='containerChat darker'>" +
        "<p class='text-left'>" + messages[i].sender + "</p>" +
        "<p class='text-left'>" + messages[i].message + "</p>" +
        "<span class='time-left'>" + timeStamp(new
Date(messages[i].date)) + "</span>";
    if (messages[i].read) {
        message += "<img src='img/check.png' style='width: 25px'
class='rounded float-left'>";
    }
```

```
    message += "</div>";
}
```

3.6. Mostrar el número de mensajes sin leer

Se ha modificado la función “loadTable” del “widget” “widget-friends.html” para añadir esta funcionalidad. Por cada uno de los amigos del usuario se crea un contador, se recorre la lista de mensajes y por cada mensaje entre ellos se aumenta en uno el contador correspondiente. Una vez finalizado la comprobación, se modifica el JSON de los amigos para añadir este contador y modificar la vista.

```
function loadTable(users, messages) {
    $("#bodyTable").empty();
    for (let i = 0; i < users.length; i++) {
        users[i].count = 0;
        if (messages != null || messages.length > 0) {
            for (let j = messages.length - 1; j >= 0; j--) {
                if (messages[j].receiver == Cookies.get('email') &&
                    messages[j].sender == users[i].email) {

                    if (messages[j].read == false) {
                        users[i].count++;
                    }
                }
            }
        }
    }
    for (let i = 0; i < users.length; i++) {
        $("#bodyTable").append(
            "<tr class='table-light'>" +
            "<td><a onclick=loadConversation('" + users[i].email + "')>"
+ users[i].name + "</a></td>" +
            "<td>" + users[i].surName + "</td>" +
            "<td>" + users[i].email + "</td>" +
            "<td style='color: red'>" + users[i].count + "</td>" +
            "</tr>"
        );
    }
}
```

3.7. Ordenar la lista de amigos por último mensaje

Para este caso de uso se ha vuelto a modificar la función “loadTable” del “widget” “widget-friends.html”. Al recorrer la lista de mensaje desde el final obtenemos los más recientes primeros. En este recorrido se pueden dar dos casos:

- Si existe un mensaje entre el usuario de la lista y el usuario en sesión se modifica el JSON del usuario en la lista para añadir este mensaje.
- Si no existe, se crea un mensaje con fecha del 2000 para que se coloque el ultimo de lista a la hora de ordenar.

Después de modificar todos los usuarios, los ordenamos por la fecha de este mensaje añadido usando la función “filter”.

```
function loadTable(users, messages) {
    $("#bodyTable").empty();
    for (let i = 0; i < users.length; i++) {
        users[i].count = 0;
        if (messages != null || messages.length > 0) {
            for (let j = messages.length - 1; j >= 0; j--) {
                if (messages[j].receiver == Cookies.get('email') &&
                    messages[j].sender == users[i].email) {
                    if (users[i].lastMessage == undefined) {
                        users[i].lastMessage = messages[j];
                    }
                    if (messages[j].read == false) {
                        users[i].count++;
                    }
                } else if (messages[j].sender == Cookies.get('email') &&
                    messages[j].receiver == users[i].email &&
users[i].lastMessage == undefined) {
                    users[i].lastMessage = messages[j];
                }
            }
        }
        if (users[i].lastMessage == undefined)
            users[i].lastMessage = {
                date: new Date(2000, 11, 24, 10, 33, 30, 0)
            };
    }

    users = users.sort(function (a, b) {
        return new Date(b.lastMessage.date) - new
Date(a.lastMessage.date);
    });

    for (let i = 0; i < users.length; i++) {
        $("#bodyTable").append(
            "<tr class='table-light'>" +
            "<td><a onclick=loadConversation('" + users[i].email + "')>"
+ users[i].name + "</a></td>" +
            "<td>" + users[i].surName + "</td>" +
            "<td>" + users[i].email + "</td>" +
            "<td style='color: red'>" + users[i].count + "</td>" +
            "</tr>"
        );
    }
}
```

Prueba Unitarias

0. Configuración de las pruebas y utilidades

1.1. Registro de Usuario con datos válidos.

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

2.1. Inicio de sesión con datos válidos.

2.2. Inicio de sesión con datos inválidos.

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

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

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

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

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

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

6.1. Listar las invitaciones recibidas por un usuario.

7.1. Aceptar una invitación recibida.

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

C1.1. Inicio de sesión con datos válidos.

C1.2. Inicio de sesión con datos inválidos (usuario no existente en la aplicación).

C2.1. Acceder a la lista de amigos de un usuario, que al menos tenga tres amigos.

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.

C3.1. Acceder a la lista de mensajes de un amigo “chat”, la lista debe contener al menos tres mensajes.

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.

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.

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.

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.