

# Desarrollo Web: NodeJS, API y Cliente

SISTEMAS DISTRIBUIDOS E INTERNET

SAÚL CASTILLO VALDÉS – UO251370

PELAYO DÍAZ SOTO – UO251000

## Tabla de contenido

<b>Aplicación Web .....</b>	<b>1</b>
1.Registrarse como Usuario.....	1
2. Iniciar Sesión.....	2
3. Listar y Buscar Usuarios .....	2
4. Enviar una Invitación de Amistad .....	3
5. Listar Invitaciones Recibidas. ....	4
6. Aceptar Invitación Recibida. ....	5
7. Listar Usuarios Amigos .....	6
<b>Servicio Web .....</b>	<b>7</b>
S.1 Identificarse con usuario – token .....	7
S.2. Usuario identificado: listar todos los amigos .....	8
S.3. Usuario identificado: Crear un mensaje.....	9
S.4. Usuario identificado: Obtener mis mensajes de una “conversación” .....	10
S.5. Usuario identificado: Marcar mensaje como leído .....	11
<b>Cliente jQuery.....</b>	<b>12</b>
C.1. Autenticación del usuario .....	12
C.2. Mostrar la lista de amigos.....	13
C.3. Mostrar los mensajes .....	13
C.4. Crear mensaje .....	14
C.5. Marcar mensajes como leídos de forma automática.....	14
C.6. Mostrar el número de mensajes sin leer .....	15
<b>Pruebas Unitarias: Selenium .....</b>	<b>16</b>
Prueba 1: Registro de un nuevo usuario con datos correctos. ....	17
Prueba 2: Registro Incorrecto .....	17
Prueba 3: Inicio de Sesión Fallido.....	18
Prueba 4: Inicio de sesión válido y acceso al listado de usuarios. ....	18
Prueba 5: Acceso inválido a listado de usuarios desde URL.....	18
Prueba 6: Búsqueda válida .....	19
Prueba 7: Búsqueda inválida .....	19
Prueba 8 y 9: Enviar petición de amistad de forma válida e inválida. ....	19
Prueba 10: Listar invitaciones de un usuario .....	20
Prueba 11: Aceptar una petición.....	20
Prueba 12: Listar los amigos de un usuario. ....	20
Prueba 13 – Cliente: Inicio de sesión inválido.....	21

Prueba 14 - Cliente: Inicio de sesión válido y listado de amigos. ....	21
Prueba 15 – Cliente: Filtrado de amigos. ....	21
Prueba 16 – Cliente: Acceder a los mensajes. ....	22
Prueba 17 – Cliente: Crear nuevo mensaje .....	22
Prueba 18 – Cliente: Intercambio de mensajes correcto .....	22
Pruebas 19 y 20 – Cliente: 3 mensajes.....	22

## Aplicación Web

### 1.Registrarse como Usuario

Para llevar a cabo este apartado se ha creado en primer lugar un fichero JavaScript de nombre `usuarios.js` que se exporta en forma de función. Dentro de él se ha creado un método `post` que responde a la url `"/usuarios"` para llevar a cabo el registro.

```
app.post('/usuario', function(req, res) {  
  var registerInfo = {  
    email: req.body.email,  
    name: req.body.name,  
    lastName: req.body.lastName,  
    password1: req.body.password,  
    password2: req.body.passwordConfirm  
  }  
}
```

Este método recoge de la petición el email, nombre, apellidos, y las dos versiones de la contraseña introducida que deben coincidir entre ellas para que el registro sea válido.

Tras realizar una serie de comprobaciones, si estas resultan satisfactorias, entonces se procede a cifrar la contraseña y a llamar al método que se encarga de guardar al nuevo usuario en la base de datos.

```
var seguro = app.get("crypto").createHmac('sha256', app.get('clave'))  
  .update(req.body.password).digest('hex');  
var usuario = {  
  email: req.body.email,  
  nombre: req.body.name,  
  apellido: req.body.lastName,  
  password: seguro  
}  
gestorBD.insertarObjeto(usuario, 'usuarios', function (id) {  
  if (id == null) {  
    res.redirect("/signup?mensaje=Error al registrar usuario");  
  } else {  
    res.redirect("/login?mensaje=Nuevo usuario registrado");  
  }  
});
```

```
insertarObjeto : function(usuario, coleccion, funcionCallback) {  
  this.mongo.MongoClient.connect(this.app.get('db'), function (err, db) {  
    if (err) {  
      funcionCallback(null);  
    } else {  
      var collection = db.collection(coleccion);  
      collection.insert(usuario, function (err, result) {  
        if (err) {  
          funcionCallback(null);  
        } else {  
          funcionCallback(result.ops[0]._id);  
        }  
      });  
      db.close();  
    }  
  });  
}
```

Este método es declarado en fichero aparte que se exporta como un objeto JavaScript, de forma que siempre se tiene acceso al mismo objeto desde cualquier parte de la aplicación. Este método recibe el objeto usuario a crear y lo inserta en la colección usuarios de la base de datos, colección

que será creada en caso de no existir de manera previa. Como comentario decir que se han realizado métodos genéricos para la búsqueda, inserción y paginaciones de las diferentes peticiones de la aplicación. De tal forma que se evita la duplicidad de código al tener un solo método para cada acción al que se le debe indicar el nombre de la colección en la que realizarla.

En cuanto al fichero html creado, consiste en un bloque que hereda del fichero `"base.html"` y se llama `"bregistro.html"`. Este fichero es un simple formulario con una serie de inputs en los que introducir los datos.

## 2. Iniciar Sesión.

En cuanto al inicio de sesión, se ha procedido a crear en el mismo fichero que el apartado anterior un nuevo método post que responde a la URL “login”.

```
app.post("/login", function(req, res) {
  var seguro = app.get("crypto").createHmac('sha256', app.get('clave'))
    .update(req.body.password).digest('hex');
  var criterio = {
    email : req.body.email,
    password : seguro
  }
  gestorBD.obtenerObjetos(criterio, 'usuarios', function(usuarios) {
    if (usuarios == null || usuarios.length == 0) {
      req.session.usuario = null;
      res.redirect("/login" +
        "?mensaje=Email o password incorrecto"+
        "&tipoMensaje=alert-danger ");
    } else {
      req.session.usuario = usuarios[0].email;
      req.session.usuarioId = usuarios[0]._id;
      res.redirect("/user/list");
    }
  });
});
```

Este método recibe del fichero html “blogin” un usuario y una contraseña. Esta última es hasheada con el mismo método que con el que se guardó, y junto con el nombre de usuario son enviados a un método de la base de datos para saber si existe dicho usuario. En caso de existir, se guarda en el objeto sesión el email y el id.

Este objeto sesión será utilizado más adelante. Puesto que se ha creado un router para controlar el acceso a distintas zonas de la aplicación.

En cuanto al método de la base de datos, es un simple método para encontrar a los usuarios en base al criterio de email-password en la tabla usuarios.

```
// routerUsuarioSession
var routerUsuarioSession = express.Router();
routerUsuarioSession.use(function(req, res, next) {
  console.log("routerUsuarioSession");
  if ( req.session.usuario ) {
    // dejamos correr la petición
    next();
  } else {
    console.log("va a : "+req.session.destino)
    res.redirect("/login");
  }
});
```

## 3. Listar y Buscar Usuarios

```
app.get("/user/list", function (req, res) {
  var pg = (req.query.pg == null) ? 1 : parseInt(req.query.pg); //Es String!!
  var criterio = (req.query.búsqueda == null) ? {} : { $or : [ // Coincidencia en nombre
    { "nombre" : { $regex : ".*"+req.query.búsqueda+".*" } },
    { "email" : { $regex : ".*"+req.query.búsqueda+".*" } } ]
  };
  gestorBD.obtenerObjetosPg( criterio, pg, 'usuarios', function (usuarios, total) { //0
    if (usuarios==null){
      res.send("Error al buscar los usuarios.")
    } else {
      var criterio2 = {
        "usuario" : req.session.usuario
      }
    }
  });
});
```

Para el listado de usuarios se crea un método get que responde a la URL “user/list”. La primera de las variables se utiliza para saber si la URL lleva consigo un parámetro para sacar una página concreta.

La segunda, se utiliza por si se introduce algún criterio de búsqueda con el que encontrar usuarios por nombre o por email.

Ambas variables son pasadas a un método de la base de datos que busca usuarios por criterio, todos si no existe; y por página, la primera si no se indica.

#### 4. Enviar una Invitación de Amistad

Continuando con el listado de usuarios, lo que se hace a continuación, una vez se tienen los usuarios, es realizar dos búsquedas más en la base de datos. Una primera para buscar las peticiones enviadas por el usuario, y una segunda para sus amistades.

```
var criterio2 = {
  "usuario" : req.session.usuario
}
gestorBD.obtenerObjetos(criterio2, 'peticiones', function (peticiones) {
  if (peticiones==null){
    res.send("Error al buscar las peticiones");
  } else {
    var id = gestorBD.mongo.ObjectId(req.session.usuarioId);
    var criterio3 = {$or : [ // Coincidencia en amistad 1 o 2
      { "amigo1_id" : id},
      { "amigo2_id" : id}]
    };
    gestorBD.obtenerObjetos(criterio3, 'amistades', function (amistades) {
      if (amistades==null){
        res.send("Error al buscar los amigos.")
      } else {
```

Una vez se ha obtenido las amistades, se procesa esta colección, como se explica más adelante, para encontrar los amigos. Una vez tenemos los amigos del usuario en sesión se procede a procesar la colección de usuarios para ir marcándolos en base a si son amigos o si ya han recibido una petición.

```
var seguirAmigos = true;
var seguirAmistades = true;

for (i=0; i<usuarios.length; i++){
  seguirAmigos = true;
  for (j=0; j<amigos.length; j++){
    if (seguirAmigos) {
      if (usuarios[i]._id.toString() == amigos[j]._id.toString()) {
        usuarios[i].esAmigo = true;
        seguirAmigos = false;
      } else {
        usuarios[i].esAmigo = false;
      }
    }
  }

  seguirAmistades = true;
  for (z=0; z<peticiones.length; z++){
    if (seguirAmistades) {
      if (usuarios[i]._id.toString() == peticiones[z].IdDestino.toString()) {
        usuarios[i].tienePetición = true;
        seguirAmistades = false;
      } else {
        usuarios[i].tienePetición = false;
      }
    }
  }
}
```

Una vez marcados, se envían a la vista donde se mostrarán los botones de enviar petición o texto en base a la condición que presente el usuario. Al botón se le asigna el id del usuario al que mandar la petición cuando se presiona.

```
{% for usuario in usuarios %}
  <tr>
    <td> {{usuario.email}}</td>
    <td> {{usuario.nombre}}</td>
    <td> {{usuario.apellido}}</td>
    {% if usuario.esAmigo %}
    <td>Ya eres amigo de este usuario</td>
    {% elseif usuario.tienePetición %}
    <td>Ya has enviado una petición.</td>
    {% elseif usuario.email == sesion %}
    <td>¿Quieres ser amigo de ti mismo?</td>
    {% else %}
    <td><a class="btn btn-primary pull-right"
      href="/peticion/mandar/{{usuario._id.toString() }}">
      Mandar Solicitud de Amistad</a></td>
    {% endif %}
  </tr>
{% endfor %}
```

```

app.get('/peticion/mandar/:id', function (req, res) {

    var usuarioIdDestino = gestorBD.mongo.ObjectId(req.params.id);
    var sessionId = gestorBD.mongo.ObjectId(req.session.usuarioId);

    gestorBD.obtenerObjetos({_id: sessionId}, 'usuarios', function (usuarios) {
        var peticion = {
            usuario : req.session.usuario,
            nombre: usuarios[0].nombre,
            apellido: usuarios[0].apellido,
            IdDestino : usuarioIdDestino
        }
        gestorBD.insertarObjeto(peticion, 'peticiones', function(idPeticion){
            if ( idPeticion == null ){
                res.send(respuesta);
            } else {
                res.redirect("/user/list?mensaje=Petición Mandada");
            }
        });
    });
});

```

Este método gestiona el Id que recibe y crea un objeto petición que contiene el email, nombre y apellidos del usuario en sesión, y el id del usuario destino.

Este objeto es mandado al gestor de la BD para que lo inserte en la colección peticiones, que se creará en caso de no existir previamente.

## 5. Listar Invitaciones Recibidas.

```

app.get("/peticion/list", function (req, res) {
    var pg = (req.query.pg == null) ? 1 : parseInt(req.query.pg); //Es String!!
    var criterio = {
        IdDestino: gestorBD.mongo.ObjectId(req.session.usuarioId)
    }
    gestorBD.obtenerObjetosPg(criterio, pg, 'peticiones', function (peticiones, total) {
        var pgUltima = total / 5;
        if (total % 5 > 0) { // Sobran decimales
            pgUltima = pgUltima + 1;
        }
        pgUltima = (pgUltima==0) ? 1 : pgUltima;
        if (peticiones == null) {
            res.send("Error al buscar las peticiones.")
        } else {
            var respuesta = swig.renderFile('views/bRequestList.html', {
                "peticiones": peticiones,
                pgActual: pg,
                pgUltima: pgUltima,
                "sesion": req.session.usuario
            });
            res.send(respuesta);
        }
    });
});

```

Para listar las peticiones recibidas se crea un método, que al igual que con los usuarios, tenga paginación para mostrar un número determinado de peticiones a la vez. Como se trata de buscar las peticiones que el usuario ha recibido, el criterio que se sigue consiste en buscar aquellas peticiones cuyo id de destino coincide con el del usuario en sesión.

Para mostrar estas peticiones en la vista, se pasa la lista entera de las peticiones encontradas y una vez se renderiza el fichero de salida, se crea una tabla de la forma:

```

{% for peticion in peticiones %}
<td> {{peticion.usuario}}</td>
<td> {{peticion.nombre}}</td>
<td> {{ peticion.apellido }}</td>
<td>Quiere ser tu amigo.</td>
<td><a class="btn btn-primary pull-right"
    href="/peticion/aceptar/{{peticion._id.toString()}}">
    Aceptar Solicitud de Amistad</a></td>
{% endfor %}

```

## 6. Aceptar Invitación Recibida.

A la hora de generar los botones para aceptar las peticiones, se incluyen las id de cada petición para que se forme una URL que sea capturada por el siguiente método:

```
app.get('/peticion/aceptar/:id', function (req, res) {
  var petitionId = gestorBD.mongo.ObjectId(req.params.id);
  var criterio = {
    _id : petitionId
  }
  gestorBD.obtenerObjetos(criterio, 'peticiones', function(peticion){
    if ( petition == null || petition.length == 0){
      res.send("/user/list" +
        "?mensaje=Ha ocurrido un error");
    } else {
      var criterio = {$or : [
        {"email" : petition[0].usuario},
        {"email" : req.session.usuario}
      ]}
      gestorBD.obtenerObjetos(criterio, 'usuarios', function (usuarios) {
        if ( usuarios == null || usuarios.length != 2){
          res.send("/user/list" +
            "?mensaje=Ha ocurrido un error");
        } else {
          var amistad = {
            amigo1: usuarios[0],
            amigo2: usuarios[1]
          }
          gestorBD.crearAmistad(amistad, petition, function(id){
            if (id == null) {
              res.redirect("/user/friendsList?mensaje=Error al haceros Amigos");
            } else {
              res.redirect("/user/friendsList?mensaje=Petición Aceptada");
            }
          })
        }
      })
    }
  })
})
```

Este método tiene tres partes: en primer lugar, se encarga de buscar la petición de la que acaba de recibir el id. Una vez encontrada crea un criterio de búsqueda con el que buscar usuarios. Este criterio hace que se busquen dos usuarios, el que está actualmente en sesión, y que por tanto recibe la petición, y el que la mandó, usuario del que tenemos el email al encontrar la petición.

Una vez tenemos los usuarios que deben hacerse amigos, entonces creamos un objeto amistad con los datos de ambos usuarios, y lo enviamos al método de la base de datos encargado de su creación.

```
crearAmistad : function(amistad, petition,functionCallback) {
  this.mongo.MongoClient.connect(this.app.get('db'), function (err, db) {
    if (err) {
      functionCallback(null);
    } else {
      var collection = db.collection('amistades');
      collection.insert(amistad, function (err, result) {
        if (err) {
          functionCallback(null);
        } else {
          var result = result.ops[0]._id;
          collection = db.collection("peticiones");
          collection.remove({"_id" : petition[0]._id}, function (err, db) {
            if (err) {
              functionCallback(null);
            } else {
              functionCallback(result);
            }
          })
        }
      })
    }
  })
  db.close();
}
```

El método que se encarga de crear la amistad, también se ocupa de borrar la petición que genero el evento para que deje de aparecer en la base de datos. Podría haberse optado por añadir un atributo que la marcarse como aceptada.



## 7. Listar Usuarios Amigos

```
app.get("/user/friendsList", function (req, res) {
    var pg = (req.query.pg == null) ? 1 : parseInt(req.query.pg); //Es String!!
    var userId = gestorBD.mongo.ObjectId(req.session.usuarioId);
    var criterio = {$or : [ // Coincidencia en amistad 1 o 2
        { "amigo1._id" : userId},
        { "amigo2._id" : userId}]
    };

    gestorBD.obtenerObjetosPg( criterio, pg, 'amistades',function (amistades, total) {
        if (amistades==null){
            res.send("Error al buscar las amistades.")
        } else {
            var pgUltima = total / 5;
            if (total % 5 > 0) { // Sobran decimales
                pgUltima = pgUltima + 1;
            }
            var amigos = [];

            for (i = 0; i<amistades.length; i++){
                if (amistades[i].amigo1._id.toString()== req.session.usuarioId){
                    amigos.push(amistades[i].amigo2);
                } else if (amistades[i].amigo2._id.toString()==req.session.usuarioId) {
                    amigos.push(amistades[i].amigo1);
                }
            }

            var respuesta = swig.renderFile('views/bListFriends.html', {
                "amigos" : amigos,
                pgActual: pg,
                pgUltima: pgUltima,
                "sesion": req.session.usuario
            });
            res.send(respuesta);
        }
    });
});
```

Para buscar las amistades de un usuario en sesión se sigue el mismo criterio en cuanto a paginación que en anteriores ocasiones. Para buscar los amigos lo que se hace es establecer un criterio donde al menos uno de los id de los objetos amistad se correspondan con el id del usuario en sesión.

Una vez tenemos las amistades que coinciden, no se puede enviar a la vista esto directamente, puesto que lo que se quiere ver es quienes son los amigos. Para ello, debemos procesar la colección que recibimos de la base de datos de tal forma que se comparan los ids de ambos usuarios con el del usuario en sesión, y se introduce aquel que no coincida en una nueva colección, pues este será la amistad.

Una vez procesada la colección, la mandamos a la vista para que muestre los amigos:

```
<tbody>
  {% for amigo in amigos %}
    <tr>
      <td> {{amigo.email}}</td>
      <td> {{amigo.nombre}}</td>
      <td> {{amigo.apellido}}</td>
    </tr>
  {% endfor %}
</tbody>
```

## Servicio Web

### S.1 Identificarse con usuario – token

Para realizar este apartado se ha tenido que:

- Crear un nuevo fichero .js en la carpeta routes llamado “**rapiusuarios.js**”, en el que implementaremos todas las peticiones REST correspondientes al servicio web.
- En el fichero app.js debemos importar (require) **jsonwebtoken** para poder trabajar con el token de autenticación del usuario. Además, en este mismo fichero añadiremos un nuevo router por el que tendrán que pasar todas las peticiones a la API, menos la de POST de autenticación (y de esta manera miraremos si tienen o no token de identificación).
- Y, por último, en el fichero que hemos creado añadimos la petición POST “**/api/login**”.

```
<<rapiusuarios.js>>
module.exports = function(app, gestorBD) {
  ...

  app.post("/api/login", function(req, res) {
    var seguro = app.get("crypto").createHmac('sha256', app.get('clave'))
      .update(req.body.password).digest('hex');
    var criterio = {
      email : req.body.email,
      password : seguro
    }

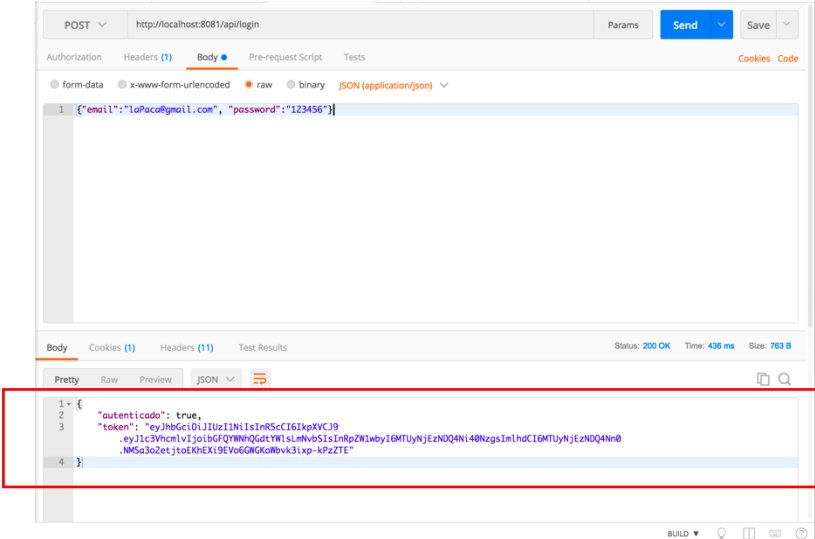
    gestorBD.obtenerUsuarios(criterio, function(usuarios) {
      if (usuarios == null || usuarios.length == 0) {
        res.status(401); // Unauthorized
        res.json({
          autenticado : false
        })
      } else {
        var token = app.get('jwt').sign(
          {
            usuario: criterio.email ,
            tiempo: Date.now()/1000
          },
          "secreto");
        res.status(200);
        res.json({
          autenticado: true,
          token : token
        });
      }
    });
  });
}
```

```
<< app.js>>
// routerUsuarioToken
var routerUsuarioToken = express.Router();
routerUsuarioToken.use(function(req, res, next) {
  // obtener el token, puede ser un parámetro GET , POST o HEADER
  var 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) > 240 ){
        res.status(403); // Forbidden
        res.json({
          acceso : false,
          error: 'Token invalido o caducado'
        });
        // También podríamos comprobar que intoToken.usuario existe
        return;
      } else {
        // dejamos correr la petición
        res.usuario = infoToken.usuario;
        next();
      }
    });
  } else {
    res.status(403); // Forbidden
    res.json({
      acceso : false,
      mensaje: 'No hay Token'
    });
  }
});
```

Cada una de estas líneas se añaden cuando se implemente la petición, en futuros apartados.

```
// Aplicar routerUsuarioToken
app.use('/api/friendsList', routerUsuarioToken);
app.use('/api/mensaje', routerUsuarioToken);
app.use('/api/mensaje/:id', routerUsuarioToken);
app.use('/api/mensaje/:idEmisor/:idReceptor', routerUsuarioToken);
...
```

Probamos esta funcionalidad:



Obtenemos el token del usuario que se autentica.

## S.2. Usuario identificado: listar todos los amigos

En este apartado solo tendremos que añadir una nueva petición POST al fichero rapiusuarios.js:

```
...
app.get("/api/friendsList", function (req, res) {

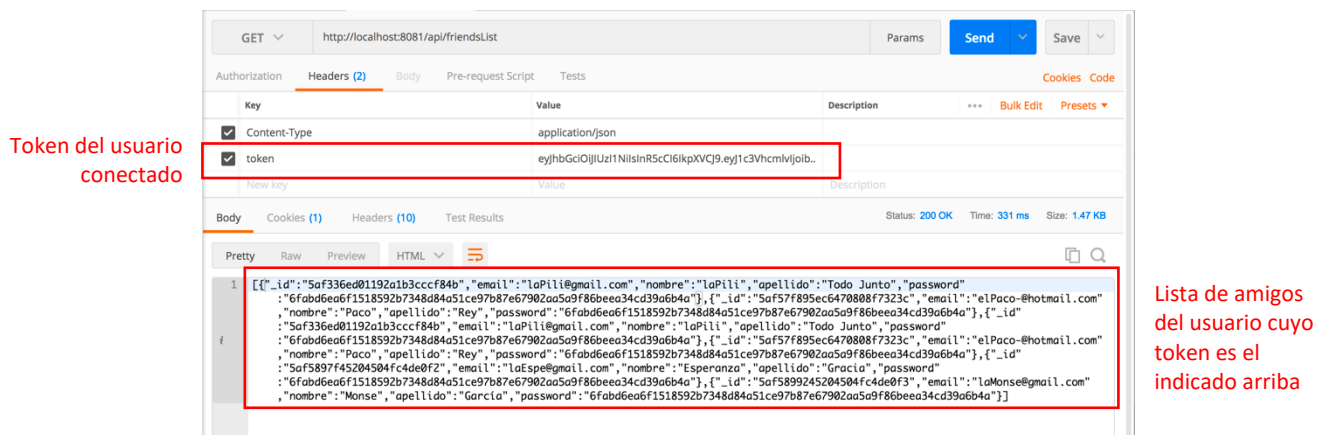
    //var userMail = res.usuario;
    var emailUser = res.usuario;
    var criterio = { $or : [ // Coincidencia en amistad 1 o 2
        { "amigo1.email" : emailUser },
        { "amigo2.email" : emailUser }
    ] };

    gestorBD.obtenerAmistades( criterio, function (amistades, total) {
        if (amistades==null){
            res.status(500);
            res.json({error: "Se ha producido un error"})
        } else {
            var amigos = [];

            for (i = 0; i<amistades.length; i++){
                if (amistades[i].amigo1.email== emailUser){
                    amigos.push(amistades[i].amigo2);
                } else if (amistades[i].amigo2.email==emailUser) {
                    amigos.push(amistades[i].amigo1);
                }
            }

            res.status(200);
            res.send(JSON.stringify(amigos));
        }
    })
}
```

Probamos la funcionalidad de esta petición:



### S.3. Usuario identificado: Crear un mensaje

Añadimos a la API la siguiente petición PUSH:

```
app.post("/api/mensaje", function (req, res) {
  var mensaje = {
    emisor: res.usuario,
    leido: false
  }
  if (req.body.destino != null)
    mensaje.destino = req.body.destino;
  if (req.body.text != null)
    mensaje.text = req.body.text;

  var criterio = {
    $or: [
      {
        $and: [{"amigo1.email": mensaje.destino},
              {"amigo2.email": mensaje.emisor}]
      },
      {
        $and: [{"amigo1.email": mensaje.emisor},
              {"amigo2.email": mensaje.destino}]
      }
    ]
  };

  gestorBD.obtenerAmistades(criterio, function (amistades, total) {
    if (amistades==null){
      res.status(500);
      res.json({error: "No sois amigos"})
    } else {
      gestorBD.crearMensaje(mensaje, function(id){
        if (id==null) {
          res.status(500);
          res.json({error: "Se ha producido un error"})
        } else {
          res.status(201);
          res.send(JSON.stringify(mensaje));
        }
      });
    }
  });
});
```

#### S.4. Usuario identificado: Obtener mis mensajes de una “conversación”

Agregamos una petición GET a la API:

```
app.get("/api/mensaje/:idReceptor", function (req, res){

  gestorBD.obtenerUsuarios({email : res.usuario }, function(usuarios) {
    if (usuarios == null || usuarios.length == 0) {
      res.status(500);
      res.json({error: "No se ha encontrado el usuario"});
    } else {
      var idEmisor = usuarios[0]._id;
      var idReceptor = gestorBD.mongo.ObjectId(req.params.idReceptor);

      var criterio = {$or : [ // Coincidencia en amistad 1 o 2
        { "_id" : idEmisor},
        { "_id" : idReceptor}
      ]};

      gestorBD.obtenerUsuarios(criterio, function(usuarios){
        if (usuarios==null || usuarios.length==0){
          res.status(500);
          res.json({error: "No se han encontrado los usuarios"});
        } else {
          criterio = {
            $or: [
              {
                $and: [{"emisor":usuarios[0].email},
                  {"destino": usuarios[1].email}]
              },
              {
                $and: [{"emisor":usuarios[1].email},
                  {"destino": usuarios[0].email}]
              }
            ]
          };
          gestorBD.obtenerMensajes(criterio, function (mensajes) {
            if (mensajes==null){
              res.status(500);
              res.json({error: "No se han encontrado mensajes"});
            } else {
              res.status(200);
              res.send(JSON.stringify(mensajes));
            }
          });
        }
      });
    }
  });
}
```

## S.5. Usuario identificado: Marcar mensaje como leído

En la API, añadimos una petición PUT:

```
app.put("/api/mensaje/:id", function (req, res){
  var criterio = { "_id": gestorBD.mongo.ObjectId(req.params.id)}

  var mensaje = {leido : true}

  gestorBD.modificarMensaje(criterio, mensaje, function (result) {
    if (result==null){
      res.status(500);
      res.json({error: "No se ha encontrado el mensaje"});
    } else {
      res.status(200);
      res.json({
        mensaje : "mensaje modificado",
        _id : req.params.id
      });
    }
  })
})
```

## Cliente jQuery

### C.1. Autenticación del usuario

Aunque lo ideal para implementar un cliente es separar éste del servicio, lo hemos juntado todo en el mismo proyecto debido al pequeño tamaño del mismo.

Para implementarlo, por tanto, hemos tenido que añadir código al fichero *app.js*, y hemos creado dos archivos *.html* (*clientes.html* y *widget-login.html*).

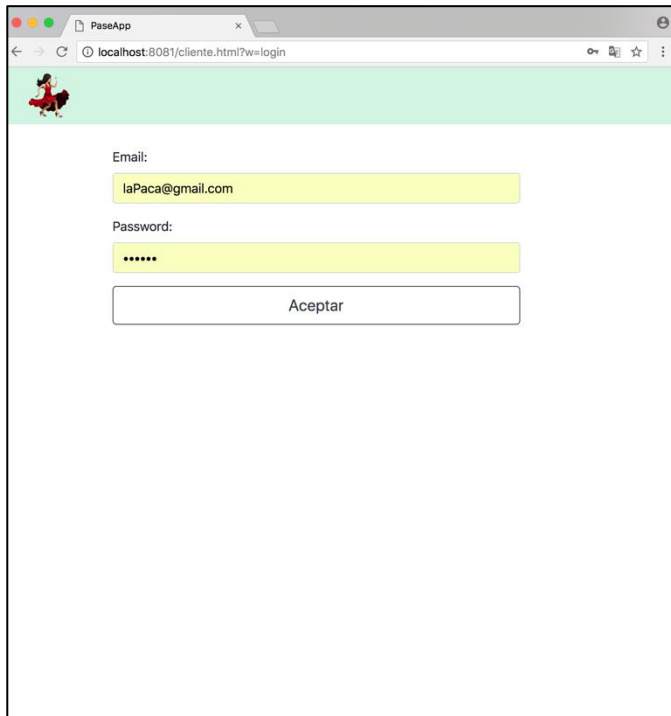
```
<< fichero app.js >>
app.use(function(req, res, next) {
  res.header("Access-Control-Allow-Origin", "*");
  res.header("Access-Control-Allow-Credentials", "true");
  res.header("Access-Control-Allow-Methods", "POST, GET, DELETE, UPDATE, PUT");
  res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept, token");
  // Debemos especificar todas las headers que se aceptan. Content-Type ,
  // token
  next();
});
```

El *cliente.html* será el encargado de definir la estructura de *.html* del cliente, así como redirigir las diferentes URL al *.html* indicado.

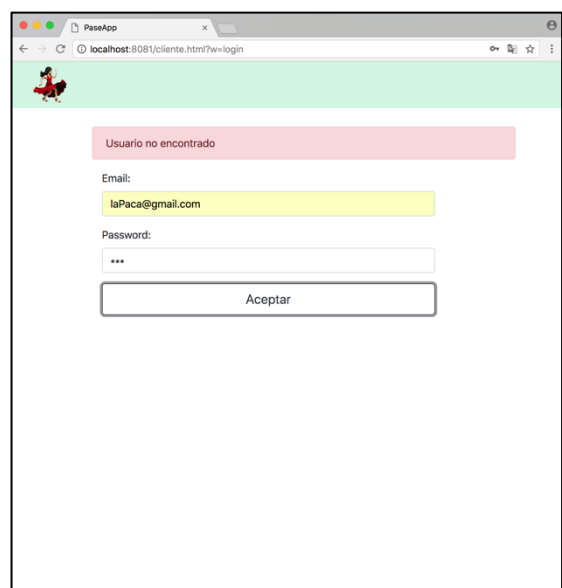
El *widget-login.html* contiene el formulario para iniciar sesión y, por tanto, se encarga de hacer la petición POST de autenticar a la API.

*<< No se ha añadido aquí el código debido a su extensión >>*

Esta funcionalidad se visualiza de la siguiente manera:



Si se introduce algún dato erróneo se muestra un mensaje de error.



## C.2. Mostrar la lista de amigos

Para este apartado es necesario crear un nuevo fichero .html, llamado **widget-listFriends.html**. En este .html, de momento, se hará una petición GET a la API para obtener los amigos del usuario conectado.

Además, existe la opción de filtrar estos usuarios por nombre y, desde un botón, actualizar la lista.

<< No se ha añadido aquí el código debido a su extensión >>

Este apartado se muestra de la siguiente manera:

Aquí se inserta el nombre o parte del mismo, y en la tabla se mostrará solo los usuarios cuyos nombres coincidan o contengan la cadena introducida.

Se muestran los datos de los usuarios amigos del usuario autenticado

Refresca la tabla de usuarios amigos

Email	Nombre	Apellido	Sin leer
laPili@gmail.com	laPili	Todo Junto	0
elPaco@hotmail.com	Paco	Rey	0
laEspe@gmail.com	Esperanza	Gracia	0
laMonse@gmail.com	Monse	García	0

## C.3. Mostrar los mensajes

Como en los otros casos, para la realización de este punto creamos un .html, en este caso, lo llamamos: **widget-chat.html**.

A este .html se accederá pulsando el nombre de cada usuario. Una vez pulsado, se abrirá el nuevo .html en el que se muestran los mensajes recibidos (en la parte de la izquierda) y los mensajes enviados (en la parte derecha) a ese usuario por parte del usuario conectado.

En este proceso se hace una petición GET a la API para obtener los mensajes entre el usuario conectado y el seleccionado.

Esta petición se realiza cada 7 segundos para mantener actualizado el chat.

<< No se ha añadido aquí el código debido a su extensión >>

Al pulsar el nombre Paco, se abre la ventana con todos los mensajes de su conversación con el usuario conectado.

Chat

\*\* leído \*\* laPaca@gmail.com 123456

elPaco@hotmail.com \*\* leído \*\*  
Te quiero paca

Introduce tu mensaje... Enviar

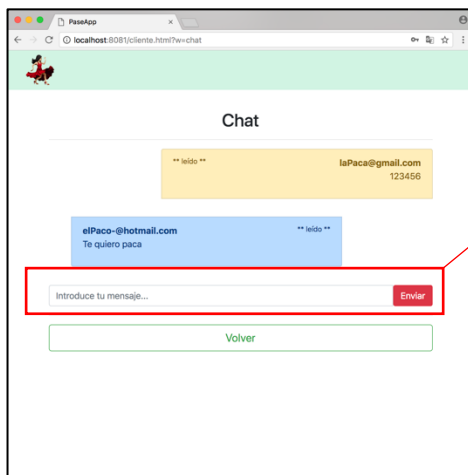
Volver



#### C.4. Crear mensaje

Esta funcionalidad se encuentra en el .html creado en el anterior punto (en el botón enviar). Cuando se pulsa el botón enviar y hay texto en el input, se realiza la petición POST a la API para crear un nuevo mensaje, además de actualizar con la petición GET, mencionada en el apartado anterior, la lista de mensajes.

<< No se ha añadido aquí el código debido a su extensión >>

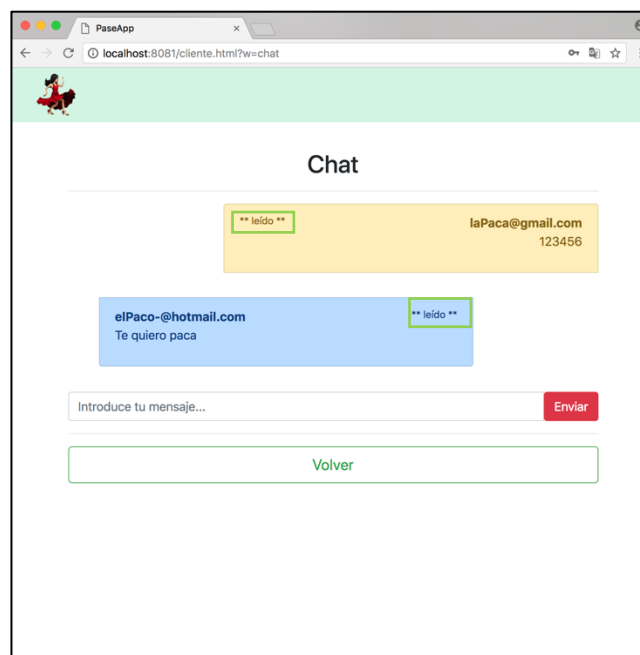


Se escribe aquí el esto y cuando se pulsa enviar el mensaje se crea.

#### C.5. Marcar mensajes como leídos de forma automática

Cada vez que un mensaje se le enseña a un usuario, se comprueba si está leído. Si no está leído se cambia a leído realizando la pertinente petición PUT a la API.

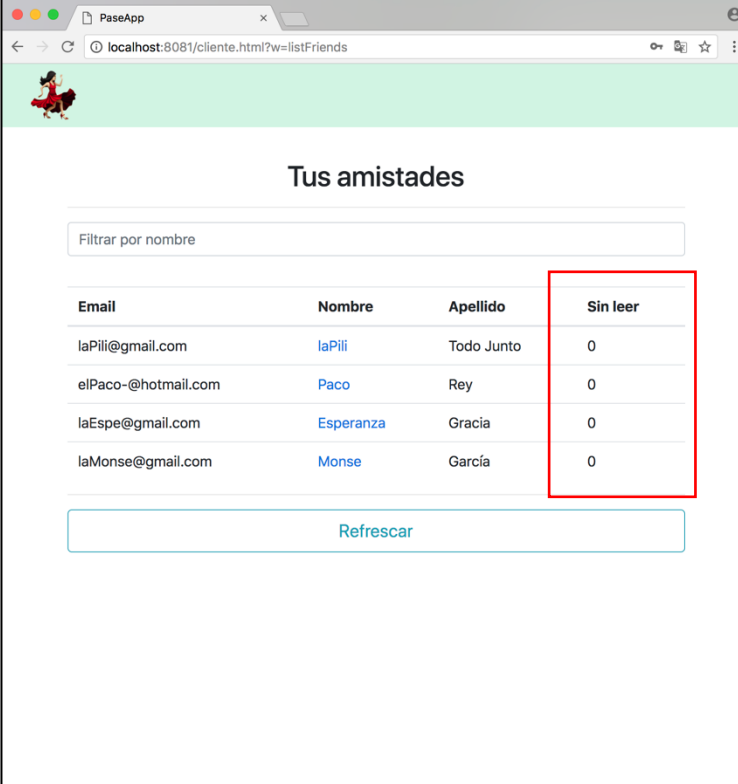
<< No se ha añadido aquí el código debido a su extensión >>



## C.6. Mostrar el número de mensajes sin leer

Para añadir este punto a nuestro cliente debemos agregar al ***widget-listFriends.html*** una petición GET a la API de los mensajes de cada amigo (cada x segundos), para mostrar el numero de mensajes sin leer en la tabla.

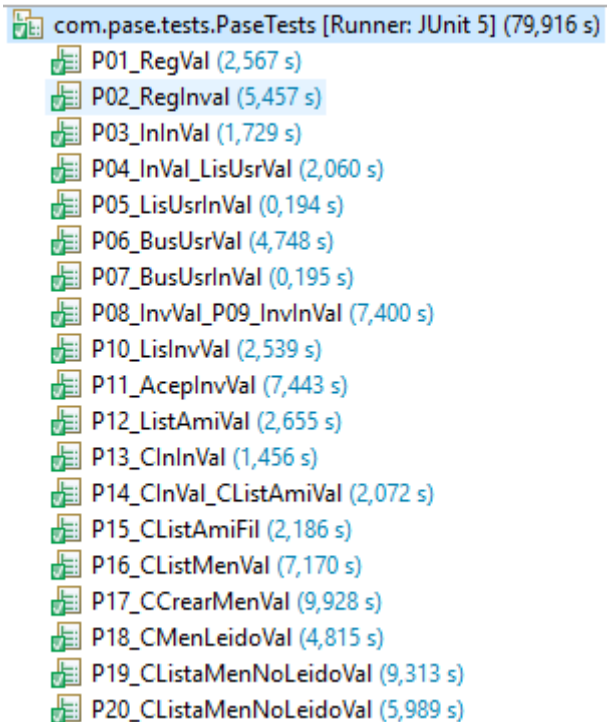
<< No se ha añadido aquí el código debido a su extensión >>



The screenshot shows a web browser window with the address bar displaying 'localhost:8081/cliente.html?w=listFriends'. The page has a green header with a small icon of a person. Below the header, the title 'Tus amistades' is centered. Underneath the title is a search bar labeled 'Filtrar por nombre'. Below the search bar is a table with four columns: 'Email', 'Nombre', 'Apellido', and 'Sin leer'. The table contains five rows of data. The 'Sin leer' column is highlighted with a red box. Below the table is a button labeled 'Refrescar'.

Email	Nombre	Apellido	Sin leer
laPili@gmail.com	laPili	Todo Junto	0
elPaco-@hotmail.com	Paco	Rey	0
laEspe@gmail.com	Esperanza	Gracia	0
laMonse@gmail.com	Monse	García	0

## Pruebas Unitarias: Selenium



Para el desarrollo de las pruebas se ha utilizado Selenium, para simular la interacción que realizaría un usuario con la web.

Se han desarrollado un total de 20 casos de prueba que han conseguido ejecutarse de manera satisfactoria.

El estilo de la implementación de las pruebas es el mismo que el visto en las prácticas de la asignatura.

Deben ejecutarse en este orden, pues algunas dependen de información introducida por casos anteriores.

Las pruebas se han documentado en el código mediante comentarios para seguir fácilmente la traza de las operaciones que se realizan.

Algunos casos de prueba agrupan dos pruebas en una por ser una consecuencia de la otra, y así no repetir el procedimiento en otra prueba diferente sólo para realizar una comprobación más. Es el caso de la P04, P08 y P14.

Cabe mencionar que se ha realizado un método que responde a una petición para borrar los datos generados durante las pruebas y no contaminar así la base de datos.

```
app.get('/borrarPruebas/:usuario', function (req, res) {
  var criterioUsuario = {
    email: req.params.usuario
  }
  var criterioAmistad = {$or : [ // Coincidencia en amistad 1 o 2
    { "amigo1.email" : req.params.usuario},
    { "amigo2.email" : req.params.usuario}]
  }

  gestorBD.borrarPruebas(criterioUsuario, criterioAmistad, function (resultado) {
    res.redirect("/login");
  });
});
```

El método recibe el correo de un usuario y establece dos criterios. El primero de ellos se ocupa de buscar en la colección usuarios, aquel que fuera creado durante las pruebas: "laLoles@gmail.com". El segundo criterio se encarga de buscar aquellas amistades de las que forme parte para borrarlas también.

Por último, aunque no se establece un criterio, el método también se encarga de borrar todos los mensajes generados durante la ejecución de las pruebas.

```

borrarPruebas: function (criterioUsuario, criterioAmistad, funcionCallback) {
    this.mongo.MongoClient.connect(this.app.get('db'), function (err, db) {
        if (err) {
            funcionCallback(null);
        } else {
            var collection = db.collection("usuarios");
            collection.remove(criterioUsuario, function (err, result) {
                if (err) {
                    funcionCallback(null);
                } else {
                    collection = db.collection("amistades");
                    collection.remove(criterioAmistad, function (err, result) {
                        if (err) {
                            funcionCallback(null);
                        } else {
                            collection = db.collection("mensajes");
                            collection.remove({"text": {$regex: "Test.*"}}, function (err, result) {
                                if (err) {
                                    funcionCallback(null);
                                } else {
                                    funcionCallback("correcto");
                                }
                            });
                        }
                    });
                }
            });
        }
    });
}

```

Prueba 1: Registro de un nuevo usuario con datos correctos.

```

@Test
public void P01_RegVal() {
    // Vamos al formulario de registro
    PO_HomeView.clickOption(driver, "signup", "class", "btn btn-primary");
    // Comprobamos que estamos en la página correcta
    PO_View.checkElement(driver, "id", "registro");
    // Rellenamos el formulario.
    PO_RegisterView.fillForm(driver, "laLoles@gmail.com", "Loles", "Fuentes", "123456", "123456");
    // Comprobamos que nos dirige al login
    PO_View.checkElement(driver, "id", "login");
}

```

Esta prueba se basa en ir hasta el formulario de registro y rellenarlo con datos válidos para incorporar un nuevo usuario a la aplicación.

Para ello presionamos la opción de menú correspondiente con el registro para a continuación comprobar que estamos en la página adecuada.

Siguiendo con la arquitectura Page Object dejamos que sea PO\_RegisterView la encargada de rellenar el formulario.

Una vez rellenado el formulario e iniciada la sesión se comprueba que, tras esto, se ha redirigido a la página correcta.

## Prueba 2: Registro Incorrecto

En esta prueba trataremos de comprobar que el formulario de registro produce un error si no se cumplen todas las condiciones.

```

@Test
public void P02_RegInval() {
    // Vamos al formulario de registro
    PO_HomeView.clickOption(driver, "signup", "class", "btn btn-primary");
    // Comprobamos que estamos en la página correcta
    PO_View.checkElement(driver, "id", "registro");
    // Rellenamos el formulario: email repetido.
    PO_RegisterView.fillForm(driver, "laLoles@gmail.com", "Josefo", "Perez", "77777", "77777");
    // Comprobamos que permanecemos en la página.
    PO_View.checkElement(driver, "id", "registro");
    // Comprobamos el error
    PO_View.checkElement(driver, "text", "Ya existe un usuario con ese email");
}

```

Para ello presionamos la opción de menú de registro y comprobamos que estamos en la página correcta. Una vez ahí, procederemos a rellenar el formulario con diferentes casos erróneos,

estos son: email repetido, nombre y apellidos cortos, contraseña corta o no coincidente al volver a introducirla.

### Prueba 3: Inicio de Sesión Fallido.

Se siguen los pasos anteriores, pero esta vez nos dirigimos al botón de inicio de sesión y es esa la página que comprobamos. Rellenemos el formulario a través de su Page Object correspondiente y probamos dos casos: el usuario no existe y contraseña incorrecta.

```
@Test
public void P03_InInVal() {
    // Vamos al formulario de inicio de sesion
    PO_HomeView.clickOption(driver, "login", "class", "btn btn-primary");
    //Comprobamos que estamos en la página correcta
    PO_View.checkElement(driver, "id", "login");

    //Rellenamos el formulario: Usuario no existe
    PO_LogInView.fillForm(driver, "noExisto@gmail.com", "123456");
    //Comprobamos el fallo comprobando que seguimos en la página
    PO_View.checkElement(driver, "id", "login");
    //Comprobamos el error
    PO_View.checkElement(driver, "text", "Email o password incorrecto");

    //Rellenamos el formulario: Contraseña inválida
    PO_LogInView.fillForm(driver, "laLoles@gmail.com", "1234567");
    PO_View.checkElement(driver, "id", "login");
    //Comprobamos el error
    PO_View.checkElement(driver, "text", "Email o password incorrecto");
}
```

### Prueba 4: Inicio de sesión válido y acceso al listado de usuarios.

```
@Test
public void P04_InVal_LisUsrVal(){
    // Vamos al formulario de inicio de sesion
    PO_HomeView.clickOption(driver, "login", "class", "btn btn-primary");
    //Comprobamos que estamos en la página correcta
    PO_View.checkElement(driver, "id", "login");
    //Rellenamos el formulario
    PO_LogInView.fillForm(driver, "laLoles@gmail.com", "123456");
    // Comprobamos que entramos en la sección privada
    PO_ListUsersView.checkElement(driver, "id", "listadoUsuarios");
}
```

En esta prueba hemos juntado dos: el inicio de sesión correcto y el acceso al listado de amigos. Para ello, seguimos los pasos del test anterior pero esta vez rellenando el formulario de forma correcta. A continuación, comprobamos que la página a la que se nos redirige es la del listado de usuarios de la aplicación.

### Prueba 5: Acceso inválido a listado de usuarios desde URL

Para realizar esta prueba, tratamos de navegar a la página de listado de usuarios y vemos como se nos redirige automáticamente a la página de Log In. Al no estar iniciados de sesión, no tenemos acceso.

```
public void P05_LisUsrInVal(){
    // Intentamos acceder al listado de usuarios sin logearnos /user/list
    driver.navigate().to(URL+"/user/list");
    //Comprobamos que se nos redirige a la pagina de inicio de sesion
    PO_LogInView.checkElement(driver, "id", "login");
}
```

## Prueba 6: Búsqueda válida

Para esta prueba en primer lugar deberemos iniciar sesión en la aplicación, y mediante el PO\_ListUsersView rellenamos el formulario de búsqueda para a continuación, comprobar que nos ha encontrado el usuario solicitado. Para ello se busca por texto en la página si existe el correo del usuario que debería dar como resultado.

```
@Test
public void P06_BusUsrVal() {
    // Vamos al formulario de inicio de sesion
    PO_HomeView.clickOption(driver, "login", "class", "btn btn-primary");
    //Comprobamos que estamos en la página correcta
    PO_ListUsersView.checkElement(driver, "id", "login");
    //Rellenamos el formulario
    PO_LogInView.fillForm(driver, "laLoles@gmail.com", "123456");
    //Comprobamos que estamos en la pagina de listado de usuarios
    PO_LogInView.checkElement(driver, "id", "listadoUsuarios");
    //Realizamos la búsqueda por nombre
    PO_ListUsersView.fillForm(driver, "Paco");
    //Comprobamos que aparece el usuario en la pagina y solo el
    List<WebElement> elementos = PO_View.checkElement(driver, "text", "elPaco@hotmail.com");
    assertTrue(elementos.size()==1);
    //Realizamos la búsqueda por email
    PO_ListUsersView.fillForm(driver, "Paca@");
    //Comprobamos que aparece el usuario en la pagina y solo el
    elementos = PO_View.checkElement(driver, "text", "laPaca@gmail.com");
    assertTrue(elementos.size()==1);
}
```

## Prueba 7: Búsqueda inválida

Metemos en la url la búsqueda deseada y vemos como, sin embargo, la aplicación nos redirige a la página de login para que iniciemos sesión pues no estamos autenticados.

```
@Test
public void P07_BusUsrInVal(){
    // Intentamos acceder al listado de usuarios sin logearnos /user/list
    driver.navigate().to(URL+"/user/list?busqueda=Paca%40");
    //Comprobamos que se nos redirige a la pagina de inicio de sesion
    PO_LogInView.checkElement(driver, "id", "login");
}
```

## Prueba 8 y 9: Enviar petición de amistad de forma válida e inválida.

Para realizar la primera de las pruebas, nos vamos a la última de las páginas, buscamos los botones que lanzan la petición de mandar y contamos el número de botones encontrados. Presionamos el primero de ellos y volvemos a la última página para comprobar que el número de botones ha disminuido.

```
public void P08_InvVal_P09_InvInVal(){
    // Vamos al formulario de inicio de sesion
    PO_HomeView.clickOption(driver, "login", "class", "btn btn-primary");
    //Rellenamos el formulario
    PO_LogInView.fillForm(driver, "laLoles@gmail.com", "123456");
    // Comprobamos que entramos en la sección privada
    PO_ListUsersView.checkElement(driver, "id", "listadoUsuarios");
    // Esperamos a que se muestren los enlaces de paginación la lista de usuarios
    List<WebElement> elementos = PO_View.checkElement(driver, "free", "//a[contains(@class, 'page-link')]");
    // Nos vamos a la última página
    elementos.get(3).click();
    //Presionar sobre boton "Enviar petición de amistad"
    elementos = PO_View.checkElement(driver, "free", "//a[contains(@href, '/peticion/mandar')]");
    elementos.get(0).click();
    int tamaño1 = elementos.size();
    try {
        Thread.sleep(1000); //Para darle tiempo a recargar la pagina
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    // Esperamos a que se muestren los enlaces de paginación la lista de usuarios
    elementos = PO_View.checkElement(driver, "free", "//a[contains(@class, 'page-link')]");
    // Nos vamos a la última página
    elementos.get(3).click();
    //Comprobamos que el número de botones se ha reducido, es decir, que no se puede volver a mandar la petición.
    elementos = PO_View.checkElement(driver, "free", "//a[contains(@href, '/peticion/mandar')]");
    int tamaño2 = elementos.size();
    assertTrue(tamaño1==tamaño2+1);
}
```

## Prueba 10: Listar invitaciones de un usuario

```
@Test
public void P10_LisInvVal() {
    // Vamos al formulario de inicio de sesion
    PO_HomeView.clickOption(driver, "login", "class", "btn btn-primary");
    //Rellenamos el formulario
    PO_LogInView.fillForm(driver, "laMonse@gmail.com", "123456");
    //Comprobamos que estamos en la página privada
    PO_ListUsersView.checkElement(driver, "id", "listadoUsuarios");
    //Vamos a la vista peticiones de amistad - href="/request/list"
    List<WebElement> elementos = PO_View.checkElement(driver, "free", "//a[contains(@href,'peticion/list')]");
    elementos.get(0).click();
    //Comprobamos que estamos en la vista de peticiones de amistad
    PO_ListUsersView.checkElement(driver, "id", "peticiones");
    //Comprobar tamaño de la tabla para ver que hay peticiones
    List<WebElement> peticiones = SeleniumUtils.EsperaCargaPagina(driver, "free", "//tbody/tr",
        PO_View.getTimeout());
    assertTrue(peticiones.size() > 0);
}
```

Para realizar este test, seguimos los pasos habituales para iniciar sesión, a continuación, vamos y comprobamos que estamos en la vista de peticiones de amistad, y que su tabla tiene contenido.

## Prueba 11: Aceptar una petición

Seguimos todos los pasos del caso anterior, para a continuación buscar los botones que contengan la referencia “petición/aceptar”. Apretamos el primero de ellos y deberíamos comprobar que el número de botones ha disminuido.

```
//Presionar sobre boton "Aceptar peticion de amistad"
elementos = PO_View.checkElement(driver, "free", "//a[contains(@href,'/peticion/aceptar/')]");
elementos.get(0).click();
//Vamos a la vista peticiones de amistad - href="/request/list"
elementos = PO_View.checkElement(driver, "free", "//a[contains(@href,'peticion/list')]");
elementos.get(0).click();
PO_View.setTimeout(2);
//Vemos que el tamaño de la tabla ha disminuido en 1
peticiones = SeleniumUtils.EsperaCargaPagina(driver, "free", "//tbody/tr", PO_View.getTimeout());
PO_View.setTimeout(5);
```

Como en este test solo había una petición, lo que hacemos en vez de comparar el número de botones es comprobar que se lanza una excepción al no encontrar ningún tr en la tabla.

## Prueba 12: Listar los amigos de un usuario.

Procedemos iniciando sesión para a continuación dirigirnos a la vista de amigos y comprobamos que en efecto hemos llegado a la página adecuada. A continuación, comprobamos que se listan los amigos.

```
public void P12_ListAmiVal() {
    // Vamos al formulario de inicio de sesion
    PO_HomeView.clickOption(driver, "login", "class", "btn btn-primary");
    //Rellenamos el formulario
    PO_LogInView.fillForm(driver, "laMonse@gmail.com", "123456");
    //Comprobamos que estamos en la página privada
    PO_ListUsersView.checkElement(driver, "id", "listadoUsuarios");
    //Vamos a la vista peticiones de amistad - href="/request/list"
    List<WebElement> elementos = PO_View.checkElement(driver, "free", "//a[contains(@href,'user/friendsList')]");
    elementos.get(0).click();
    //Comprobamos que estamos en la vista de listas de amigos
    PO_ListUsersView.checkElement(driver, "id", "listadoAmigos");
    //Comprobar tamaño de la tabla
    List<WebElement> peticiones = SeleniumUtils.EsperaCargaPagina(driver, "free", "//tbody/tr",
        PO_View.getTimeout());
    assertTrue(peticiones.size() > 0);
}
```



### Prueba 13 – Cliente: Inicio de sesión inválido.

Se sigue el mismo protocolo que en el caso del despliegue web anterior, con la excepción de que nos dirigimos a otra URL.

```
@Test
public void P13_CInInVal() {
    // Vamos al formulario de inicio de sesion
    driver.navigate().to("http://localhost:8081/cliente.html");
    //Comprobamos que estamos en la página correcta
    PO_View.checkElement(driver, "id", "widget-login");

    //Rellenamos el formulario: Usuario no existe
    PO_LogInView.fillForm(driver, "noExisto@gmail.com", "123456");
    //Comprobamos el fallo comprobando que seguimos en la página
    PO_View.checkElement(driver, "id", "login");
    //Comprobamos el error
    PO_View.checkElement(driver, "text", "Usuario no encontrado");

    //Rellenamos el formulario: Contraseña inválida
    PO_LogInView.fillForm(driver, "laPaca@gmail.com", "1234567");
    PO_View.checkElement(driver, "id", "login");
    //Comprobamos el error
    PO_View.checkElement(driver, "text", "Usuario no encontrado");
}
```

### Prueba 14 - Cliente: Inicio de sesión válido y listado de amigos.

```
@Test
public void P14_CInVal_CListAmiVal() {
    // Vamos al formulario de inicio de sesion
    driver.navigate().to("http://localhost:8081/cliente.html");
    //Comprobamos que estamos en la página correcta
    PO_View.checkElement(driver, "id", "widget-login");

    //Rellenamos el formulario: Usuario no existe
    PO_LogInView.fillForm(driver, "laPaca@gmail.com", "123456");
    //Comprobamos que accedemos a la página de listar amigos
    PO_View.checkElement(driver, "id", "widget-friendsList");
    try {
        Thread.sleep(1000); //Para darle tiempo a recargar la pagina
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    //Comprobar tamaño de la tabla para ver las amistades
    List<WebElement> amistades = SeleniumUtils.EsperaCargaPagina(driver, "free", "tbody/tr",
        PO_View.getTimeout());
    assertTrue(amistades.size() >= 3);
}
```

Se siguen los mismos pasos que en el caso anterior, solo cambian los ids que se comprueban. Además, dormimos durante un segundo el hilo de ejecución para darle tiempo al navegador a cargar los elementos.

### Prueba 15 – Cliente: Filtrado de amigos.

Seguimos los pasos del caso anterior, pero en esta ocasión rellenamos el cuadro de búsqueda y comprobamos que el tamaño de la tabla que muestra es igual a 1.

```
//Enviamos el criterio de búsqueda
WebElement email = driver.findElement(By.id("filtro-nombre"));
email.click();
email.clear();
email.sendKeys("Pili");
//Comprobar tamaño de la tabla para ver las amistades despues del filtrado
amistades = SeleniumUtils.EsperaCargaPagina(driver, "free", "tbody/tr",
    PO_View.getTimeout());
assertTrue(amistades.size() >= 1);
```



### Prueba 16 – Cliente: Acceder a los mensajes.

Siguiendo con el procedimiento habitual, nos dirigimos a la vista de los mensajes. Para ello, desde la lista de amistados, buscamos los enlaces a los chat.

```
//Buscamos el enlace al chat
elements = SeleniumUtils.EsperaCargaPagina(driver, "free", "//a[contains(@onclick,'chat')]",
    PO_View.getTimeout());
assertTrue(elements.size() > 0);
```

Apretamos el primero de ellos, y, tras dejar un tiempo para darle tiempo al navegador a cargar los mensajes, comprobamos que existen más de tres.

```
//Comprobamos que existen los mensajes
elements = SeleniumUtils.EsperaCargaPagina(driver, "free", "//li",
    PO_View.getTimeout());
assertTrue(elements.size() > 3);
```

### Prueba 17 – Cliente: Crear nuevo mensaje

Mismo procedimiento de antes, pero en esta ocasión, cuando estamos en la vista de chat rellenamos el input para mandar un mensaje, y comprobamos que el numero de mensajes se ha incrementado en uno.

```
//Guardamos el número de mensajes
elements = SeleniumUtils.EsperaCargaPagina(driver, "free", "//li",
    PO_View.getTimeout());
assertTrue(elements.size() > 0);
int tamaño1 = elements.size();
// Introducimos el nuevo mensaje
WebElement mensaje = driver.findElement(By.id("input-chat"));
mensaje.click();
mensaje.clear();
mensaje.sendKeys("Test 17");
//Pulsar el boton de Alta.
By boton = By.id("btn-chat");
driver.findElement(boton).click();
try {
    Thread.sleep(9000); //Para darle tiempo a recargar la pagina
} catch (InterruptedException e) {
    e.printStackTrace();
}
// Comprobamos que el numero de mensajes se ha incrementado en uno
elements = SeleniumUtils.EsperaCargaPagina(driver, "free", "//li",
    PO_View.getTimeout());
assertTrue(tamaño1+1==elements.size());
```

### Prueba 18 – Cliente: Intercambio de mensajes correcto

Para esta prueba utilizaremos el mensaje enviado en la prueba anterior. Iniciaremos sesión como el destinatario y comprobamos que, en efecto, el mensaje aparece.

### Pruebas 19 y 20 – Cliente: 3 mensajes

Debido al tamaño de la prueba, hemos decidido dividirla en dos. La prueba 19 se ocupa de iniciar sesión como un usuario y enviar 3 mensajes, además de comprobar que estos aparecen en el chat.

Por su parte, la prueba 20 se encarga de comprobar que el usuario destinatario tiene 3 mensajes sin leer en las notificaciones, en la vista de amistades.

## Anexo:

Los usuarios activos con los que se puede acceder a la aplicación sin necesidad de registro son:

- [laPaca@gmail.com](mailto:laPaca@gmail.com) → Usuario Principal con mayor número de amistades, mensajes...
- [laPili@gmail.com](mailto:laPili@gmail.com)
- [elPaco-@hotmail.com](mailto:elPaco-@hotmail.com)
- [laVero@gmail.com](mailto:laVero@gmail.com)
- [laEspe@gmail.com](mailto:laEspe@gmail.com)
- [laMonse@gmail.com](mailto:laMonse@gmail.com)
- [elJonas@gmail.com](mailto:elJonas@gmail.com)
- [elJhonan@gmail.com](mailto:elJhonan@gmail.com)
- [xkYoLoValgo@gmail.com](mailto:xkYoLoValgo@gmail.com)

Todos ellos con la contraseña "123456".