Documento de arquitectura

Iván Escuín González,684146
Adrián Martínez Pérez,576307

1.Hoja de ruta

El objetivo a tratar es la documentación de la arquitectura que se ha seguido para elaborar una aplicación web de índole “Red social”, concretamente para la publicación y votación en encuestas por parte de los usuarios. Para ello se han registrado 3 tipos de vistas: Módulos, C&C y Distribución que ayudarán tanto a reflejar la arquitectura y las decisiones que se han tomado en ella por parte de los desarrolladores, como para guiar en la implementación de la aplicación a estos.

2. Visión general del sistema

2.1 Requisitos

2.1.1 Diccionario de datos

- **Perfil de usuario**: Datos públicos de cada usuario registrado que pueden ser consultados por otros usuarios registrados. Consta de nombre, fecha de nacimiento, país, sexo y una lista de encuestas que ha publicado. Los datos obligatorios son nombre, fecha de nacimiento y país. Los datos modificables son nombre, fecha de nacimiento, país y sexo
- **Cuenta de usuario**: Datos usados para permitir acceso a la aplicación. Constan de nombre de usuario y contraseña, y tiene asociada un perfil de usuario.
- **Usuario registrado**: Se entiende al usuario como aquél que accede a la aplicación mediante una cuenta de usuario.
- **Usuario anónimo**: Usuario de la aplicación que no posee una cuenta de usuario.
- **Encuesta**: Se conforma de una pregunta y una serie de respuestas, hasta un máximo de 10. No se puede votar 2 respuestas diferentes en una encuesta por un mismo usuario. No puede haber 2 respuestas iguales.
- **Catálogo de encuestas:** Conjunto de encuestas creadas por usuarios registrados.

2.1.2 Requisitos funcionales

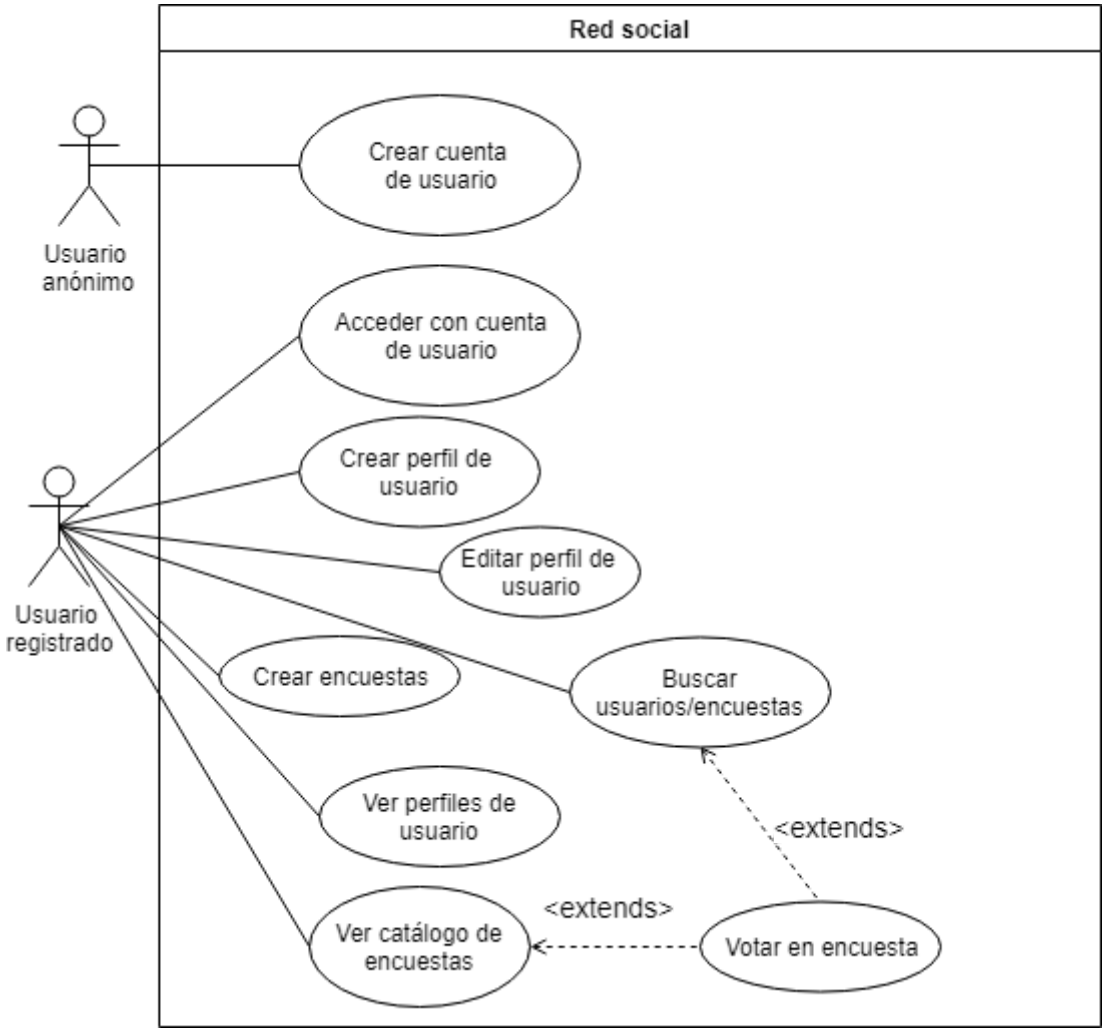
RF1	El usuario anónimo podrá crear una cuenta de usuario.
RF2	El usuario registrado podrá crear encuestas.
RF3	El usuario registrado podrá votar en las encuestas.
RF4	El usuario registrado podrá ver encuestas.

RF5	El usuario registrado podrá crear y modificar su perfil de usuario.
RF6	El usuario registrado podrá ver perfiles de usuario de otros usuarios registrados.
RF7	El usuario registrado podrá ver un catálogo de encuestas.
RF8	El usuario registrado podrá acceder a la aplicación mediante su cuenta de usuario.
RF9	El usuario registrado podrá buscar perfiles de usuario y encuestas.

2.1.3 Requisitos no funcionales

RNF1	Las encuestas deberán tener asociado un tiempo de expiración, tras el cual se impedirá el voto.

2.2 Diagrama de casos de uso



2.3 Descripción de casos de uso

Caso de uso: Crear cuenta de usuario
Id: 1
Actores: Usuario anónimo
Precondición: El usuario intenta usar la aplicación sin cuenta de usuario.
Postcondición: El usuario anónimo pasara a ser usuario registrado.
Resumen:
El usuario anónimo selecciona la opción de crear cuenta y la crea.

Curso normal:

1	El caso de uso comienza cuando el usuario anónimo entra en el sitio web y selecciona la opción de crear cuenta.		
2	El usuario introduce los datos solicitados.		
		3	El sistema crea la cuenta de usuario introducidos y lo lleva a la creación de perfil.

Caso de uso: Acceder con cuenta de usuario
Id: 2
Actores: Usuario registrado
Precondición: El usuario intenta usar la aplicación con cuenta de usuario.
Postcondición: El usuario accede al sitio web como usuario registrado.
Resumen:
El usuario registrado introduce sus datos de cuenta de usuario.

Curso normal:

1	El caso de uso comienza cuando el usuario registrado entra en el sitio web.		
2	El usuario introduce los datos en los campos y pulsa sobre “acceder”.		

		3	El sistema comprueba si los datos son correctos.
		4	Si lo son, el usuario accede a la página principal del sitio web como usuario registrado.

Cursos alternos:

4a	Si los datos introducidos no son correctos, se le indica al usuario “los datos introducidos no son correctos”.

Caso de uso: Crear perfil de usuario
Id: 3
Actores: Usuario registrado
Precondición: El usuario registrado accede por primera vez al sitio web.
Postcondición: La cuenta de usuario del usuario registrado tendrá asociado un perfil de usuario
Resumen:
El usuario registrado introduce sus datos de perfil de usuario.

Curso normal:

1	El caso de uso comienza cuando el usuario registrado accede al sitio web con su cuenta por primera vez.	2	El sistema muestra los campos de creación de perfil.
3	El usuario introduce los datos, como mínimo los obligatorios.		
		4	El sistema crea el nuevo perfil y lo asocia a la cuenta de usuario.

Cursos alternos:

4a	Si no se introducen los datos obligatorios, se le indica al usuario cuales faltan.

Caso de uso: Editar perfil de usuario

Id: 4
Actores: Usuario registrado
Precondición: El usuario registrado posee un perfil asociado.
Postcondición: -
Resumen:
El usuario registrado modifica sus datos de perfil de usuario.

Curso normal:

1	El caso de uso comienza cuando el usuario registrado accede al sitio web y pulsa sobre “editar perfil”.	2	El sistema muestra los campos de creación de perfil, con los datos guardados.
3	El usuario introduce los datos, como mínimo los obligatorios.		
		4	El sistema modifica el nuevo perfil de usuario.

Cursos alternos:

4a	Si no se introducen los datos obligatorios, se le indica al usuario cuales faltan.

Caso de uso: Crear encuestas
Id: 5
Actores: Usuario registrado
Precondición: -
Postcondición: Se creará una nueva encuesta asociada a un perfil de usuario.
Resumen:
El usuario registrado crea una encuesta para que otros usuarios voten en ella.

Curso normal:

1	El caso de uso comienza cuando el usuario registrado accede al sitio web y pulsa sobre “nueva encuesta”.	2	El sistema muestra los campos de creación de encuesta.
3	El usuario introduce los datos y un tiempo de expiración.		

		4	El sistema crea una nueva encuesta, la asocia al perfil de usuario y le asigna el tiempo de expiración.

Cursos alternos:

3a	El usuario no introduce un tiempo de expiración.
4a	El sistema le asignará a la encuesta un tiempo por defecto.

Caso de uso: Ver perfiles de usuario
Id: 6
Actores: Usuario registrado
Precondición: -
Postcondición: -
Resumen:

El usuario registrado ve el perfil de otro usuario registrado.

Curso normal:

1	El usuario accede al perfil vía enlace en el sitio web.	2	El sistema muestra los datos asociados al perfil de ese usuario.

Cursos alternos:

1a	El usuario accede al perfil directamente con la URI asociada.

Caso de uso: Votar en encuesta
Id: 7
Actores: Usuario registrado
Precondición: La encuesta elegida aún no ha expirado.
Postcondición: Se añade un voto a la encuesta.
Resumen:

El usuario registrado vota una de las respuestas de una encuesta.

Curso normal:

1	El caso de uso comienza cuando el usuario registrado abre una encuesta.	2	El sistema muestra la encuesta: la pregunta y sus respuestas asociadas.
3	El usuario elige una de las respuestas.		
		4	El sistema registra el voto en la encuesta y le muestra al usuario el número de votos de la encuesta.

Caso de uso: Ver el catálogo de encuestas
Id: 8
Actores: Usuario registrado
Precondición: Existen encuestas sin expirar en el sistema.
Postcondición: -
Resumen:

El usuario ve una serie de encuestas con opción a votar en ellas.

Curso normal:

1	El caso de uso comienza cuando el usuario registrado accede al sitio web y pulsa sobre “ver encuestas”.	2	El sistema elige un conjunto de encuestas.
		3	El sistema muestra una encuesta
4	El usuario vota en la encuesta. Si no se cumple ninguna de las condiciones de 5, se vuelve a 3. **Pto de extensión**: Caso 7, votar en encuesta.		
5	El usuario decide dejar de votar	5	El sistema deja de mostrar encuestas

Cursos alternos:

4a	El usuario decide no votar en esa encuesta y pulsa sobre “siguiente encuesta”.

Caso de uso: Buscar usuarios/encuestas
Id: 10
Actores: Usuario registrado
Precondición: -
Postcondición: -
Resumen:

El usuario introduce terminos de busqueda y se le muestran posibles resultados

Curso normal:

1	El caso de uso comienza cuando el usuario registrado introduce algo en el campo de búsqueda.		
		2	El sistema muestra una serie de encuestas y usuarios acordes a lo introducido
3	El usuario entra en una encuesta o perfil de usuario.	4	El sistema muestra la encuesta o perfil.
5	El usuario puede votar en la encuesta. **Pto de extensión**: Caso 7, votar en encuesta.		

Cursos alternos:

2a	El sistema no encuentra ningún resultado y se lo indica al usuario.

2.4 Tecnologías de implementación

Capa de presentación: Bootstrap 4.

- Lenguajes de programación: HTML 5, CSS 3 y Javascript con JQuery 3.3.1.

Capa de aplicación: Spring Boot 2.0.0 con Gradle 4.0 para gestión de dependencias.

- Dependencias de Spring Framework: starter data REST, starter data Neo4j (OGM) y starter test.
- Lenguajes de programación: Java 1.8 y un DSL basado en Groovy.

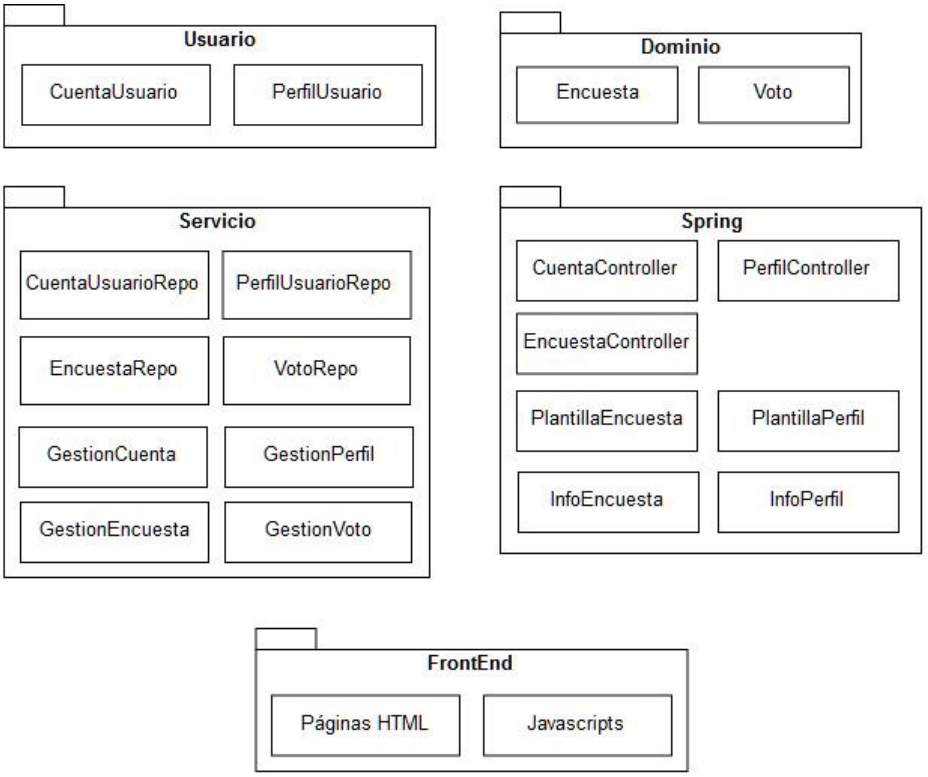
Capa de infraestructura: BD Neo4j Server 3.3.3 con bolt driver.

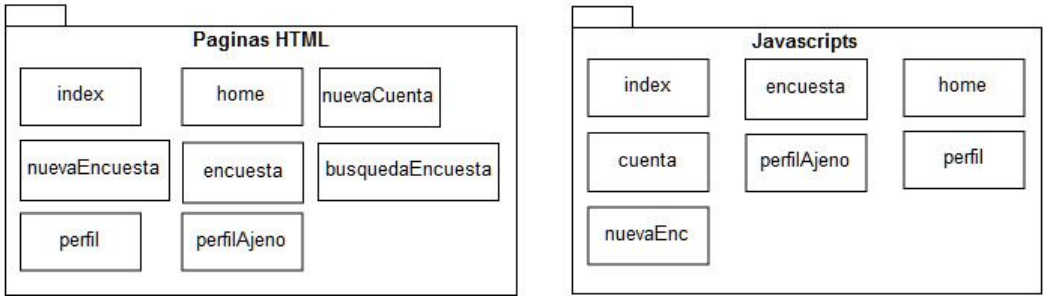
3. Documentos de las vistas

3.1 Vista de módulos

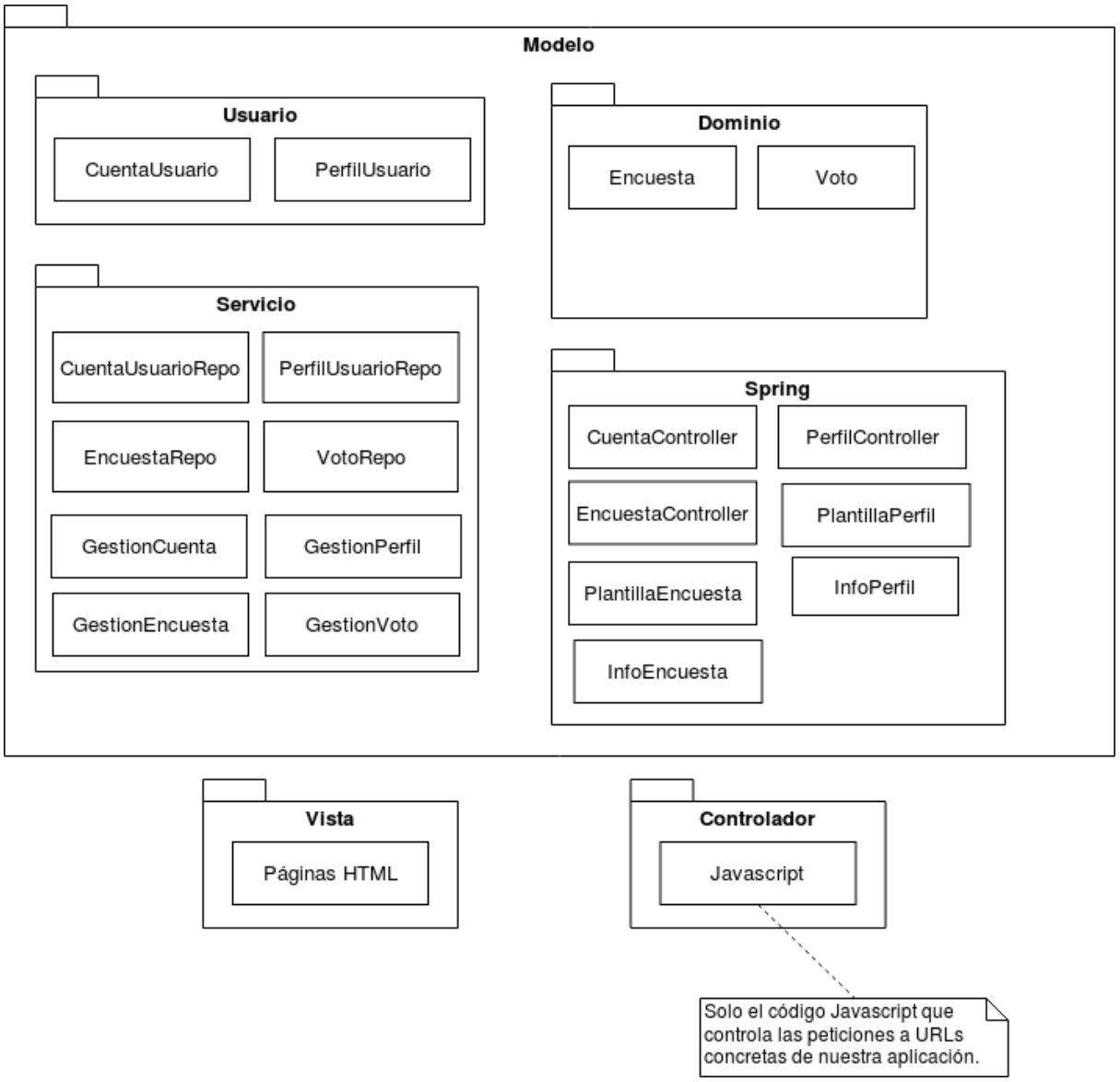
3.1.1 Presentación primaria

3.1.1.a Estilo de descomposición (paquetes)

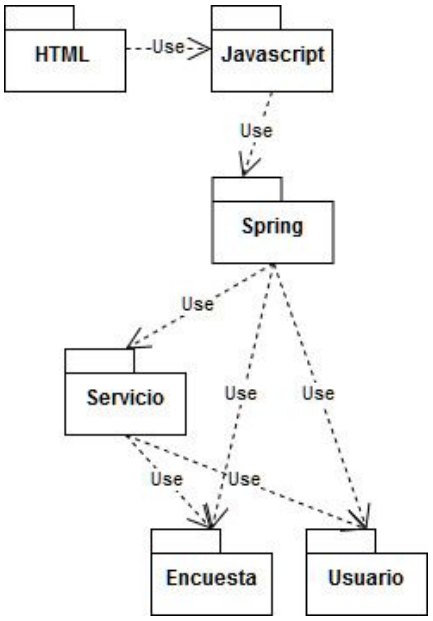




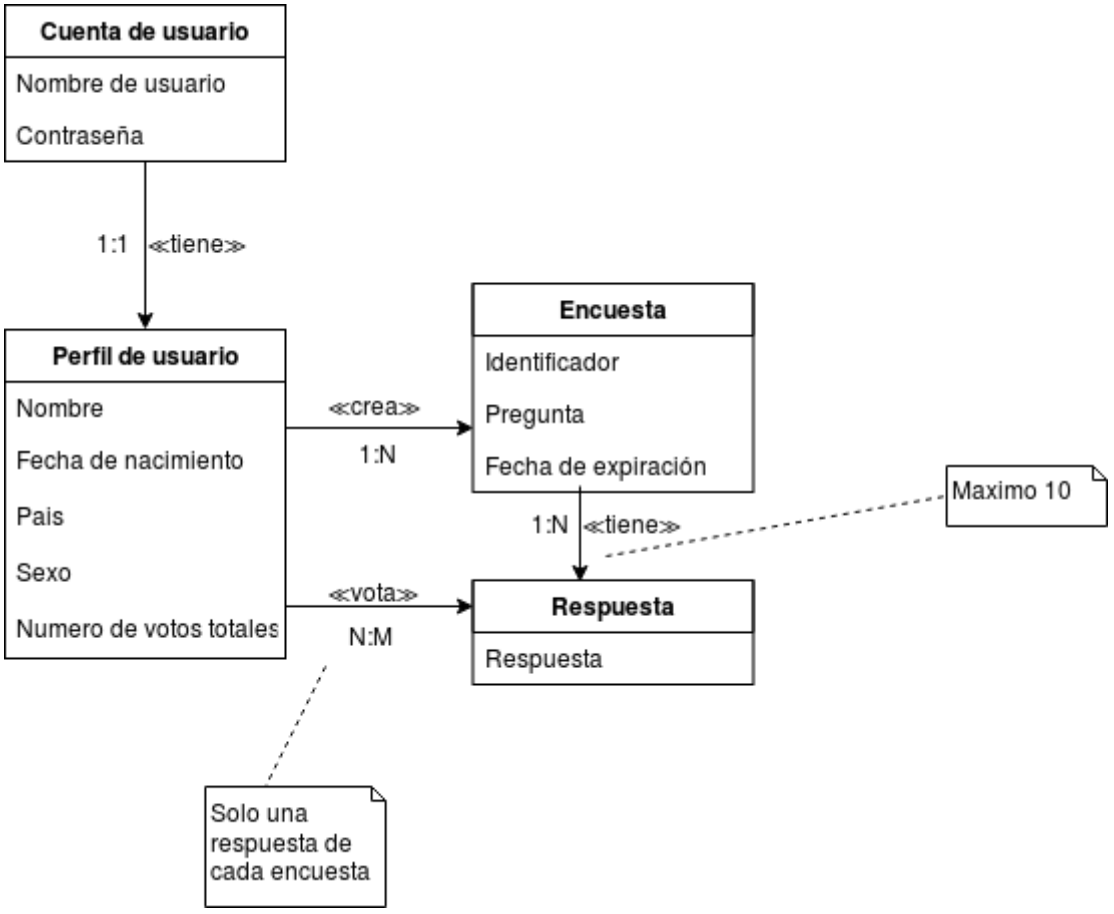
3.1.1.b Estilo de descomposición (MVC)



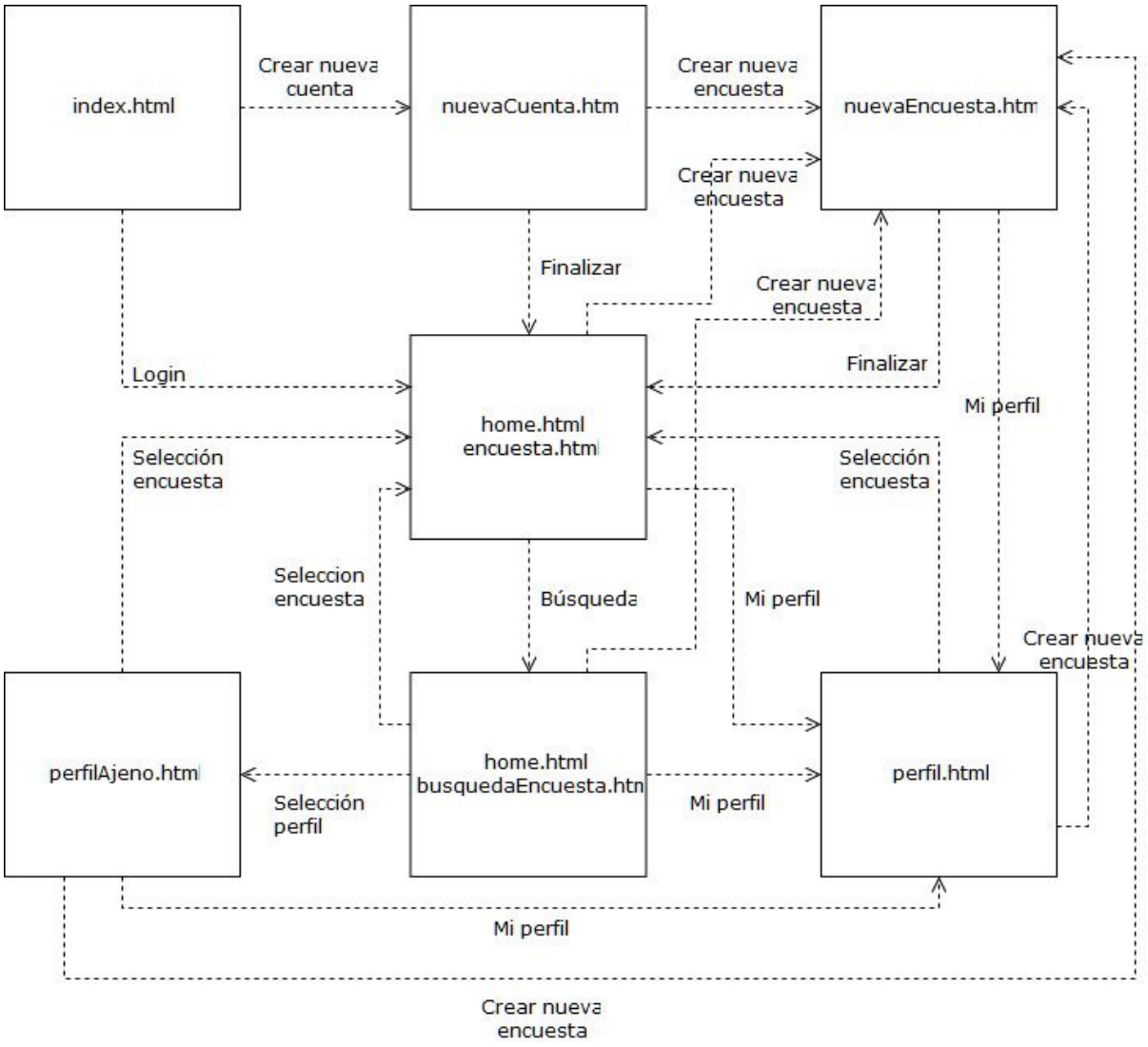
3.1.1.c Estilo de uso



3.1.1.d Modelo de datos



3.1.1.e Mapa de navegación



3.1.2 Catálogo de la vista

3.1.2.a Elementos y sus propiedades

Módulos

- Usuario:
  -Responsabilidad: representación de perfiles de usuario y cuentas de usuario.
  - CuentaUsuario:
    -Responsabilidad: representación de la cuenta que permite acceder a la aplicación.
  - PerfilUsuario:
    -Responsabilidad: representación del perfil asociado a la cuenta de usuario que contiene los datos especificados.
- Dominio:

-Responsabilidad: Representación de encuestas y votos.
    - Encuesta:
        -Responsabilidad: representación de una encuesta.
    - Voto:
        -Responsabilidad: representación de un voto.
- Servicio:
    -Responsabilidad: servicios de aplicación que hacen cumplir los requisitos.
    - CuentaUsuarioRepo:
        -Responsabilidad: relación con los nodos CuentaUsuario.
    - PerfilUsuarioRepo:
        -Responsabilidad: relación con los nodos PerfilUsuario.
    - EncuestaRepo:
        -Responsabilidad: relación con los nodos Encuesta.
    - VotoRepo:
        -Responsabilidad: relación con los nodos Voto.
    - GestionCuenta:
        -Responsabilidad: servicios relacionados con las cuentas de usuario.
    - GestionPerfil:
        -Responsabilidad: servicios relacionados con los perfiles de usuario.
    - GestionEncuesta:
        -Responsabilidad: servicios relacionados con las encuestas.
    - GestionVoto:
        -Responsabilidad: servicios relacionados con los votos.

- Spring:
    -Responsabilidad: comunicación HTTP con el front-end.
    - CuentaController:
        -Responsabilidad: servicios relacionados con cuentas de usuario.
    - PerfilController:
        -Responsabilidad: servicios relacionados con perfiles de usuario.
    - EncuestaController:
        -Responsabilidad: servicios relacionados con encuestas.
    - PlantillaPerfil:
        -Responsabilidad: Objeto DTO para la creación de perfiles.
    - PlantillaEncuesta:
        -Responsabilidad: Objeto DTO para la creación de encuestas.
    - InfoPerfil:
        -Responsabilidad: Objeto DTO para la visualización/modificación de perfiles.
    - InfoEncuesta:
        -Responsabilidad: Objeto DTO para la visualización/modificación de encuestas.
- Páginas HTML:
    -Responsabilidad: vistas de usuario. Contienen a su vez, importados, los scripts controladores de las acciones del usuario.
    - index:

-Responsabilidad: vista inicial de la aplicación. Login de usuario.

- nuevaCuenta:
  -Responsabilidad: vista de creación de cuenta y perfil asociado.
- home:
  -Responsabilidad: vista principal de la aplicación. Búsqueda de encuestas y perfiles, votación de encuestas.
- encuesta:
  -Responsabilidad: vista de una encuesta, votada y sin votar. Pregunta, número de votos y respuestas y sus porcentajes de voto respecto de los totales, si procede.
- busquedaEncuesta:
  -Responsabilidad: vista de la lista de encuestas encontradas en una búsqueda.
- nuevaEncuesta:
  -Responsabilidad: vista de creación de encuestas.
- index:
  -Responsabilidad: vista inicial de la aplicación.
- perfil:
  -Responsabilidad: vista del perfil del usuario logueado. Modificación del perfil.
- perfilAjeno:
  -Responsabilidad: vista de un perfil de usuario distinto al nuestro, y sus encuestas.

- Javascripts:
  -Responsabilidad: scripts controladores de las acciones del usuario y la carga de la página que lo importa.
  - index:
    -Responsabilidad: controlador de login.
  - cuenta:
    -Responsabilidad: controlador de creación de cuenta y perfil de usuario.
  - home:
    -Responsabilidad: controlador de obtención de perfil propio y de búsqueda de encuestas y perfiles.
  - encuesta:
    -Responsabilidad: controlador de obtención y búsqueda de encuestas.
  - nuevaEnc:
    -Responsabilidad: controlador de creación de encuesta.
  - perfil:
    -Responsabilidad: controlador de obtención y modificación de perfil, y lista de encuestas asociada.
  - perfilAjeno:
    -Responsabilidad: controlador de obtención de perfil, y lista de encuestas asociada.

3.1.2.b Interfaces de los elementos

-Identidad y Recursos:

- CuentaUsuario:
    - **CuentaUsuario(String nombreUsuario,String password)**: Constructor público de la clase CuentaUsuario, devuelve una nueva instancia de este objeto con los atributos nombreUsuario y password asignados.
    - **String getNombreUsuario()**: Devuelve el objeto String correspondiente al atributo NombreUsuario de la instancia.
    - **void asignarPerfil(PerfilUsuario perfilUsuario)**: Modifica la instancia del objeto, asignandole un PerfilUsuario al objeto.
    - **public String getPassword()**: Devuelve el objeto String correspondiente al hash md5 de la contraseña del usuario.
- PerfilUsuario:
    - **PerfilUsuario(String nombre, Date nacimiento, String pais)**:Constructor público de la clase PerfilUsuario, devuelve una nueva instancia de este objeto con los atributos nombre,nacimiento y pais asignados.
    - **PerfilUsuario(String nombre, Date nacimiento, String pais, String sexo)**: Constructor público de la clase PerfilUsuario, devuelve una nueva instancia de este objeto con los atributos nombre,nacimiento,pais y sexo asignados.
    - **Long getId()**: Devuelve el objeto Long correspondiente al atributo id de la instancia.
    - **String getNombre()**: Devuelve el objeto String correspondiente al atributo nombre de la instancia.
    - **Date getNacimiento()**: Devuelve el objeto Date correspondiente al atributo nacimiento de la instancia.
    - **String getPais()**: Devuelve el objeto String correspondiente al atributo pais de la instancia.
    - **String getSexo()**: Devuelve el objeto String correspondiente al atributo sexo de la instancia.
    - **void modificarPerfil(InfoPerfil nuevoPerfil)**: Modifica los atributos de esta instancia, haciendo las conversiones necesarias, dado un objeto de tipo InfoPerfil creado desde la interfaz. Se usa para la actualización de un PerfilUsuario.
- Encuesta:
    - **Encuesta(PerfilUsuario perfilUsuario, String pregunta, ArrayList<String> respuestas)**: Constructor público de la clase Encuesta, devuelve una nueva instancia de este objeto con los atributos perfilUsuario,pregunta y respuestas asignados.
    - **String getPregunta()**: Devuelve el objeto String correspondiente al atributo pregunta de la instancia.
    - **String getAutor():** Devuelve el objeto String correspondiente al atributo autor de la instancia.
    - **Map<String, String> getListaRespuestas()**: Devuelve el objeto Map clave String valor String correspondiente la lista de respuestas de la instancia. La clave representa al número de respuesta y el valor al texto de la respuesta.

    - **List<Voto> getVotos():** Devuelve el objeto List de Votos correspondiente al atributo que representa todos los votos de la instancia.
- CuentaUsuarioRepo:
    - **CuentaUsuario findByNombreUsuario(String nombreUsuario):** Devuelve un objeto CuentaUsuario de la base de datos dado un nombreUsuario, si no existe devuelve null.
- PerfilUsuarioRepo:
    - **PerfilUsuario findByNombre(String nombre):** Devuelve un objeto PerfilUsuario de la base de datos dado un nombre, si no existe devuelve null.
    - **List<PerfilUsuario> findByNombreContainingOrNombreIsLike(String nombre1,String nombre2):** Devuelve un objeto List de PerfilUsuario de la base de datos cuyo nombre contenga nombre1 o sea similar a nombre2, en la práctica se usa con nombre1=nombre2. Si no encuentra ninguno, devuelve un objeto List vacío.
- EncuestaRepo:
    - **Encuesta findByPregunta(String pregunta):** Devuelve un objeto Encuesta de la base de datos dado una pregunta, si no existe devuelve null.
    - **List<Encuesta> findByPreguntaContainingOrPreguntaIsLike(String pregunta1,String pregunta2):** Devuelve un objeto List de Encuesta de la base de datos cuyas preguntas contengan pregunta1 o sea similar a pregunta2, en la práctica se usa con pregunta1=pregunta2. Si no encuentra ninguno, devuelve un objeto List vacío.
    - **List<Encuesta> findByPerfilUsuarioNombreContaining(String nombre):** Devuelve un objeto List de Enceusta de la base de datos cuyos nombres asociados a perfilUsuario contenga nombre. Si no encuentra ninguno, devuelve un objeto List vacío. Solamente se usa para la carga de datos inicial.
- VotoRepo:
    - **List<Voto> findByPerfilUsuarioNombre(String nombre):** Devuelve un objeto List de Voto de la base de datos cuyos nombres asociados a perfilUsuario contenga nombre. Si no encuentra ninguno, devuelve un objeto List vacío.
- GestionCuenta:
    - **boolean crearCuenta(CuentaUsuario cuenta):** Añade un nuevo Objeto CuentaUsuario a la base de datos.
    - **CuentaUsuario obtenerCuenta(String nombre):** Extrae un objeto CuentaUsuario de la base de datos dado el atributo nombre. Devuelve null si no existe.
    - **boolean verificarCuenta(CuentaUsuario cuenta):** Verifica que la CuentaUsuario con la que se está trabajando es correcta. Devuelve true si lo es y false en caso contrario
    - **void borrarTodo():** Elimina todos los objetos CuentaUsuario de la base de datos. Solamente se usa en la carga de datos inicial.
- GestionPerfil:
    - **boolean crearPerfil(CuentaUsuario cuenta, PerfilUsuario perfil):** Añade un nuevo Objeto PerfilUsuario a la base de datos. Antes de añadir verifica

que cuenta es correcta. Si la operacion se realiza correctamente devuelve true, en caso contrario devuelve false.
    - **boolean modificarPerfil(Long id, InfoPerfil perfil, CuentaUsuario cuenta):** Modifica un Objeto PerfilUsuario identificado por su id en la base de datos, usando los datos presentes en perfil. Antes de modificar verifica que cuenta es correcta. Si la operacion se realiza correctamente devuelve true, en caso contrario devuelve false.
    - **List<PerfilUsuario> buscarPerfilNombre(String nombre):** Realiza una búsqueda de objetos PerfilUsuario dado un atributo nombre. Si la búsqueda no da resultados, devuelve una lista vacía.
    - **PerfilUsuario obtenerPerfil(Long id):** Extrae un objeto PerfilUsuario de la base de datos dado su atributo id. Devuelve null si no existe.
    - **void borrarTodo():** Elimina todos los objetos PerfilUsuario de la base de datos. Solamente se usa en la carga de datos inicial.
- GestionEncuesta:
    - **Encuesta buscarEncuestaPorPregunta(String pregunta):** Extrae un objeto Encuesta de la base de datos dado su atributo pregunta. Devuelve null si no existe.
    - **List<Encuesta> buscarEncuestasPorPreguntaAproximada(String pregunta):** Realiza una búsqueda de objetos Encuestas en la base de datos cuyas preguntas se aproximen a pregunta. Si no encuentra ninguno, devuelve un objeto List vacío.
    - **List<Encuesta> buscarEncuestasPorPerfilUsuarioAproximado(String nombre):** Realiza una búsqueda de objetos Encuestas en la base de datos cuyos nombres de los objetos PerfilUsuario asociados se aproximen a pregunta. Si no encuentra ninguno, devuelve un objeto List vacío.
    - **boolean crearEncuesta(CuentaUsuario cuenta, Encuesta encuesta):** Añade un nuevo Objeto Encuesta a la base de datos. Antes de añadir verifica que cuenta es correcta. Si la operacion se realiza correctamente devuelve true, en caso contrario devuelve false.
    - **void borrarTodo():** Elimina todos los objetos Encuesta de la base de datos. Solamente se usa en la carga de datos inicial.
- GestionVoto:
    - **boolean crearVoto(CuentaUsuario cuenta, Voto voto):** Añade un nuevo Objeto Voto a la base de datos. Antes de añadir verifica que cuenta es correcta. Si la operacion se realiza correctamente devuelve true, en caso contrario devuelve false.
    - **Pair<Boolean,SortedMap<Integer, Integer>> obtenerVotosporRespuesta(Encuesta encuesta, String cliente):** Devuelve una tupla que contiene el número de votos de cada respuesta perteneciente a encuesta y comprueba si el PerfilUsuario con el nombre cliente ya ha realizado un voto.
    - **void borrarTodo():** Elimina todos los objetos Voto de la base de datos. Solamente se usa en la carga de datos inicial.
- CuentaController:

    - **ResponseEntity crearNuevaCuenta(@RequestBody CuentaUsuario nuevaCuenta):** Responde ante una petición HTTP tipo POST al endpoint “/crearCuenta”. Crea y guarda un nuevo Objeto CuentaUsuario usando los datos JSON del cuerpo del mensaje. Si todo va bien devuelve un código HTTP 200, en caso contrario devuelve un 403.
    - **ResponseEntity login(@RequestParam(value = "nombreUsuario") String nombreUsuario, @RequestParam(value = "password") String password):** Responde ante una petición HTTP tipo GET al endpoint “/login”. Verifica las credenciales de logueo nombreUsuario y password e inicia la sesión de usuario en la aplicación. Si todo va bien devuelve un código HTTP 200, en caso contrario devuelve un 404.
- PerfilController:
    - **ResponseEntity crearPerfil(@RequestBody PlantillaPerfil nuevoPerfil):** Responde ante una petición HTTP tipo POST al endpoint “/perfilUsuario”. Crea y guarda un nuevo Objeto PerfilUsuario usando los datos JSON del cuerpo del mensaje. Si todo va bien devuelve un código HTTP 200, en caso contrario devuelve un 403.
    - **InfoPerfil obtenerPerfil(@RequestParam(value="nombreCuenta") String nombreCuenta, HttpServletResponse response):** Responde ante una petición HTTP tipo GET al endpoint “/perfilUsuario”. Intenta obtener un objeto PerfilUsuario dado un nombre de CuentaUsuario. Si todo va bien devuelve un código HTTP 200 y un objeto InfoPerfil, en caso contrario devuelve un 403.
    - **InfoPerfil obtenerPerfilDe(@PathVariable("id") String id, HttpServletResponse response):** Responde ante una petición HTTP tipo GET al endpoint “/perfilUsuario/{id}”. Intenta obtener un objeto PerfilUsuario dado un nombre de CuentaUsuario. Si todo va bien devuelve un código HTTP 200 y un objeto InfoPerfil, en caso contrario devuelve un 403.
    - **List<InfoPerfil> buscarPerfil(@RequestParam(value="busqueda") String busqueda, HttpServletResponse response):** Responde ante una petición HTTP tipo GET al endpoint “/buscarPerfilUsuario”. Intenta obtener un objeto/s PerfilUsuario realizando una búsqueda con el nombre de PerfilUsuario busqueda. Si todo va bien devuelve un código HTTP 200 y un objeto Lista de InfoPerfil, en caso contrario devuelve un 403.
    - **void actualizarPerfil(@RequestBody InfoPerfil perfil,@PathVariable String id,HttpServletResponse response):** Responde ante una petición HTTP tipo PUT al endpoint “/perfilUsuario/{id}”. Actualiza el objeto PerfilUsuario con id = {id} con la información presente en perfil.
- EncuestaController:
    - **void crearNuevaCuenta(@RequestBody PlantillaEncuesta nuevaPlantilla):** Responde ante una petición HTTP tipo POST al endpoint “/crearEncuesta”. Crea y guarda un nuevo Objeto Encuesta usando los datos JSON del cuerpo del mensaje. Si todo va bien devuelve un código HTTP 200, en caso contrario devuelve un 403.
    - **List<InfoEncuesta> obtenerEncuesta(@RequestParam String nombreUsuario,@RequestParam(required = false) String usuarioEncuesta,HttpServletResponse response):**Responde ante una

petición HTTP tipo GET al endpoint “/encuesta”. Intenta obtener un objeto/s infoEncuesta dado un nombre de PerfilUsuario que representa al autor de una Encuesta. Además comprueba si el PerfilUsuario con nombre nombreUsuario ya ha realizado algún voto en las Encuestas resultado. Si todo va bien devuelve un código HTTP 200 y un objeto Lista de InfoEncusta, en caso contrario devuelve un 404.

- **List<InfoEncuesta> buscarEncuesta(@RequestParam String nombreUsuario,@RequestParam String pregunta, HttpServletResponse response):** Responde ante una petición HTTP tipo GET al endpoint “/buscarEncuesta”. Intenta obtener un objeto/s infoEncuesta dado un parámetro pregunta que representa a una posible pregunta de una Encuesta. Además comprueba si el PerfilUsuario con nombre nombreUsuario ya ha realizado algún voto en las Encuestas resultado. Si todo va bien devuelve un código HTTP 200 y un objeto Lista de InfoEncusta, en caso contrario devuelve un 404.
- **void votarRespuesta(@RequestParam String nombreCuenta, @RequestParam String nombreEncuesta,@RequestParam int nRespuesta):** Responde ante una petición HTTP tipo PUT al endpoint “/encuesta”.Crea y almacena un objeto Voto relacionado con la Encuesta con pregunta nombreEncuesta y el PerfilUsuario perteneciente a la CuentaUsuario con nombre nombreCuenta. Si todo va bien devuelve un código HTTP 200 y un objeto Lista de InfoEncusta, en caso contrario devuelve un 500

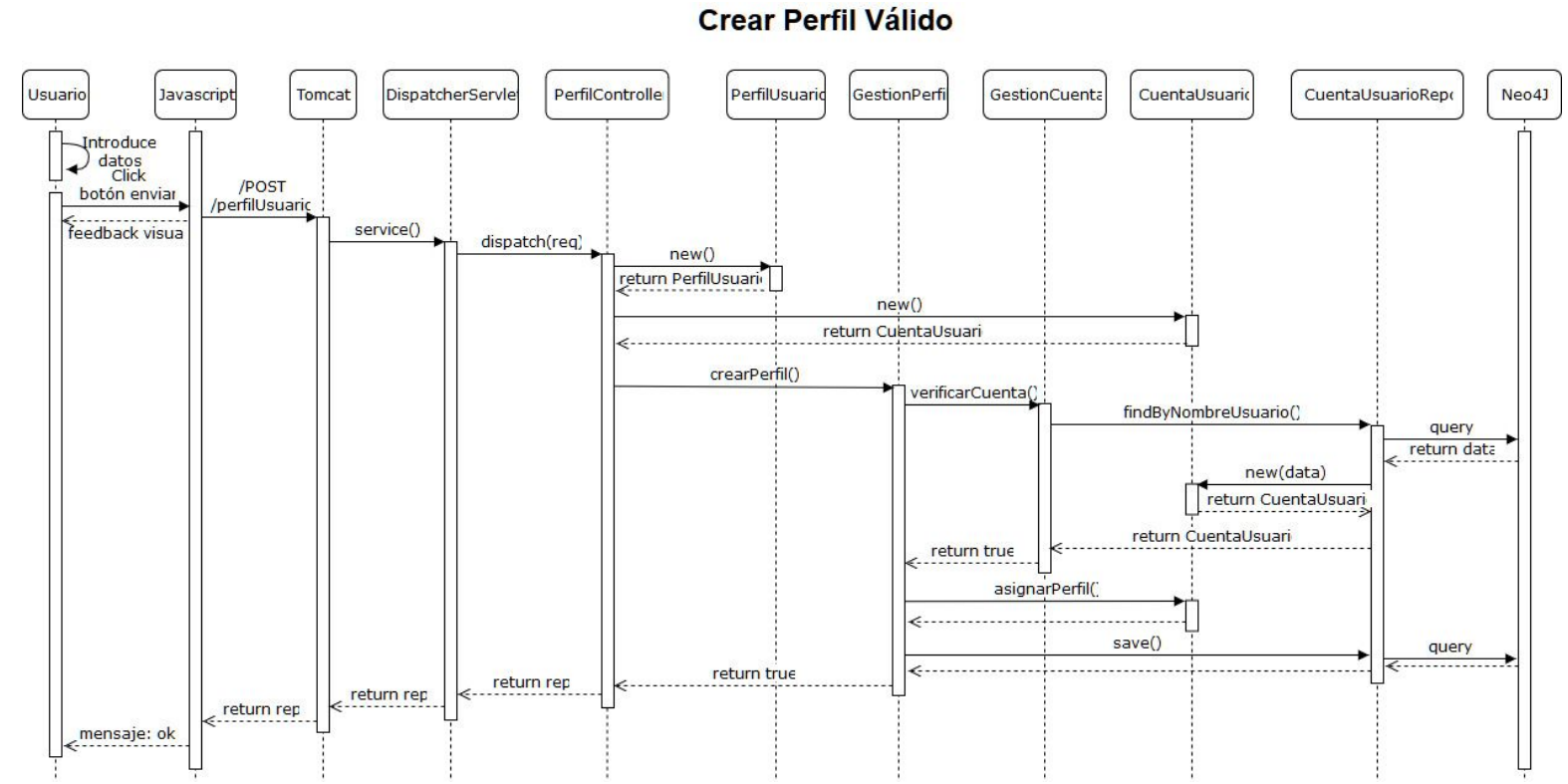
3.1.2.c Tipos de datos y constantes

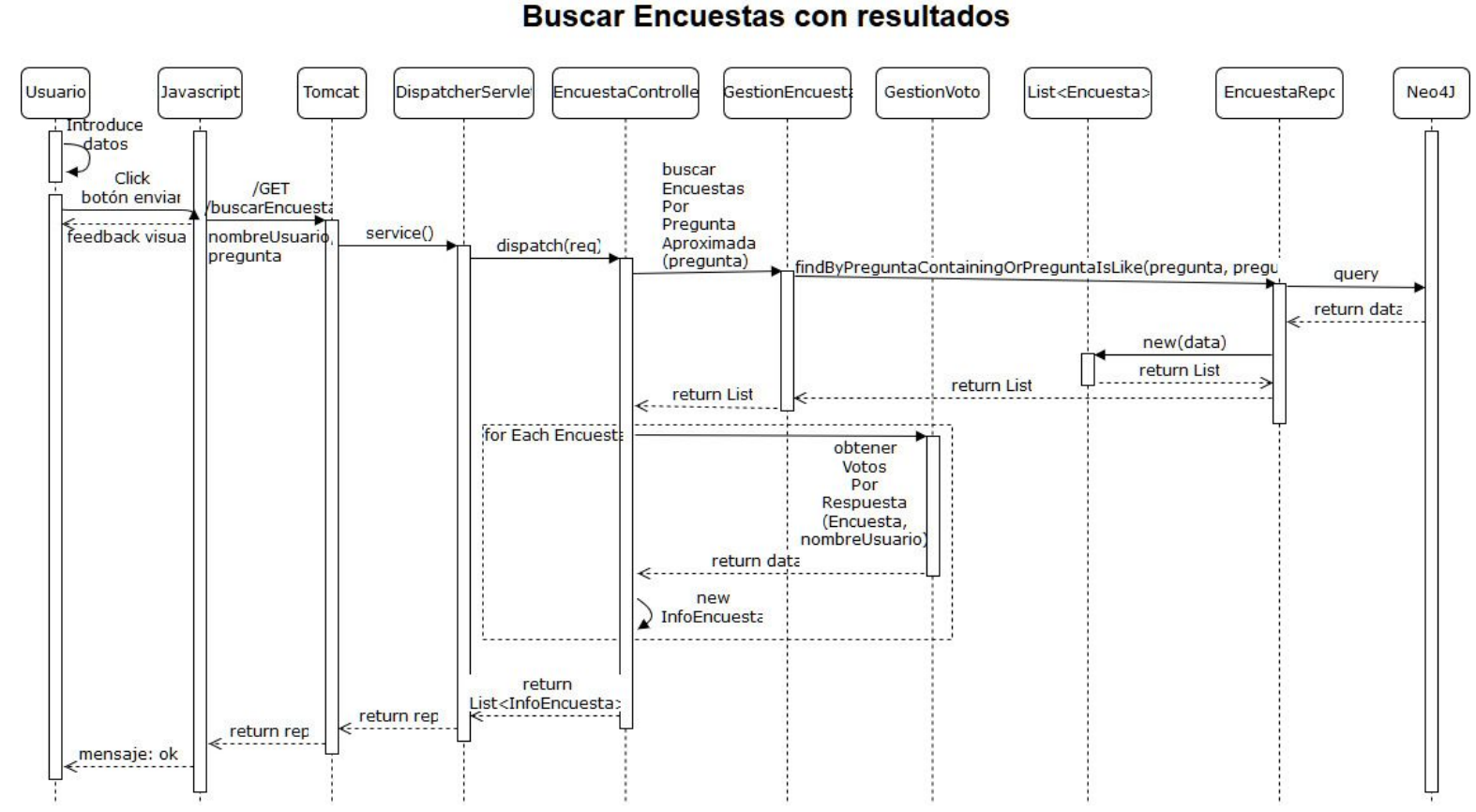
Para trabajar con las interfaces pertenecientes al módulo de Spring, y algunas de Usuario, es necesario usar los datos de tipo InfoPerfil, InfoEncuesta, PlantillaPerfil y PlantillaEncuesta ya que además de imponer restricciones en la entrada y salida de datos, son usadas por el framework como objetos de serialización de JSON. Para el resto de interfaces se usan primitivas de Java.

3.1.2.d Exposición de razones y asuntos de diseño.

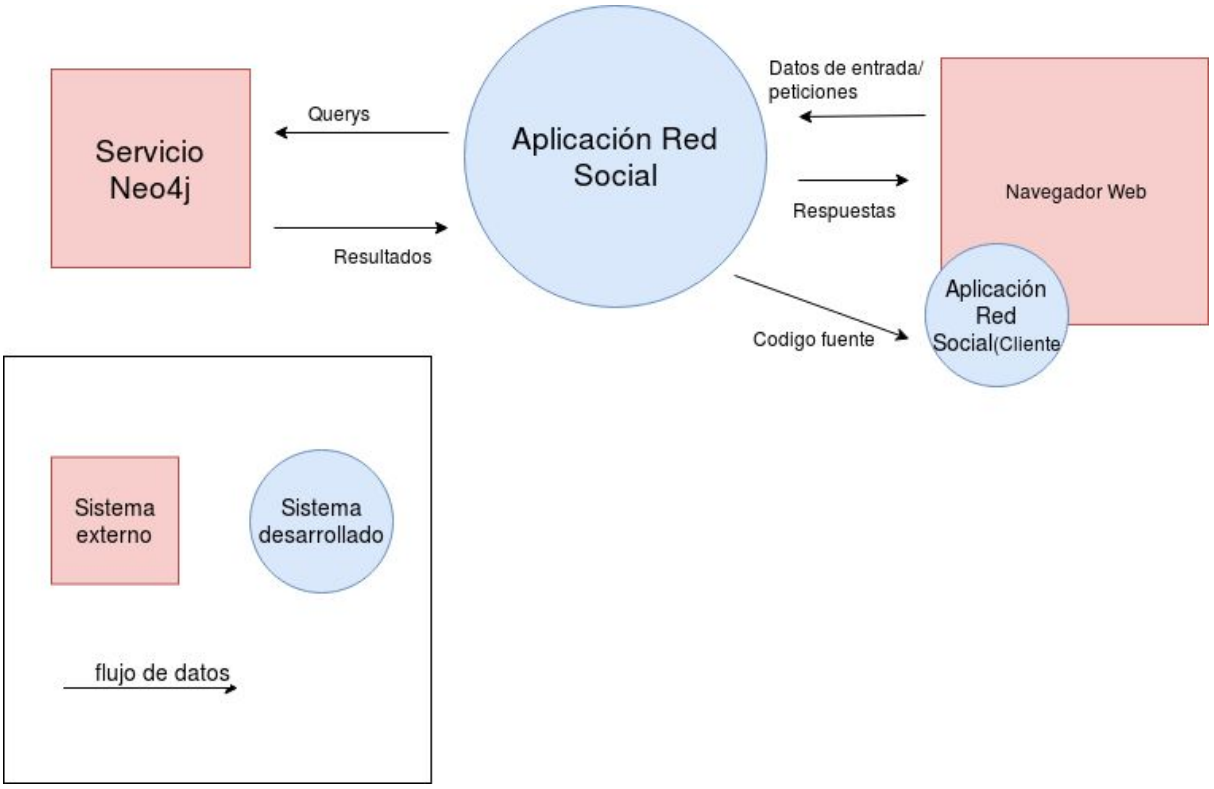
En alguna interfaz, como por ejemplo **findByPreguntaContainingOrPreguntaIsLike(String pregunta1,String pregunta2)** en la interfaz de EncuestaRepo, nos hemos visto restringidos por la sintaxis impuesta para los repositorios de SpringJPA. En este caso no permitía usar un mismo parámetro String para la primera parte de la condición y para la segunda parte.

3.1.2.e Comportamiento de los elementos





3.1.3 Diagrama de contexto



3.1.4 Exposición de razones

3.1.4.a Módulos identificados

1. Asunto: La identificación de los módulos y submódulos que tendrá el sistema y las responsabilidades de cada uno para dar mayor o menor importancia a estos módulos.
2. Decisión tomada e implementada
3. Asunciones: Las tomadas en las Tecnologías de implementación
4. Alternativas: Módulo adicional de búsqueda, descomposición más temática de los submódulos pertenecientes a Usuario, Dominio y Servicio.
5. Argumentos: Se consideró innecesario la adición de un módulo de búsqueda porque al final toda funcionalidad de búsqueda recae sobre los sub módulos de Spring. Se decidió agrupar los submódulos por funcionalidad, para que los módulos estuvieran más cohesionados.
6. Implicaciones: Cambios en la vista de módulos. Se estableció un orden en las invocaciones a métodos entre submódulos de distinto módulo.

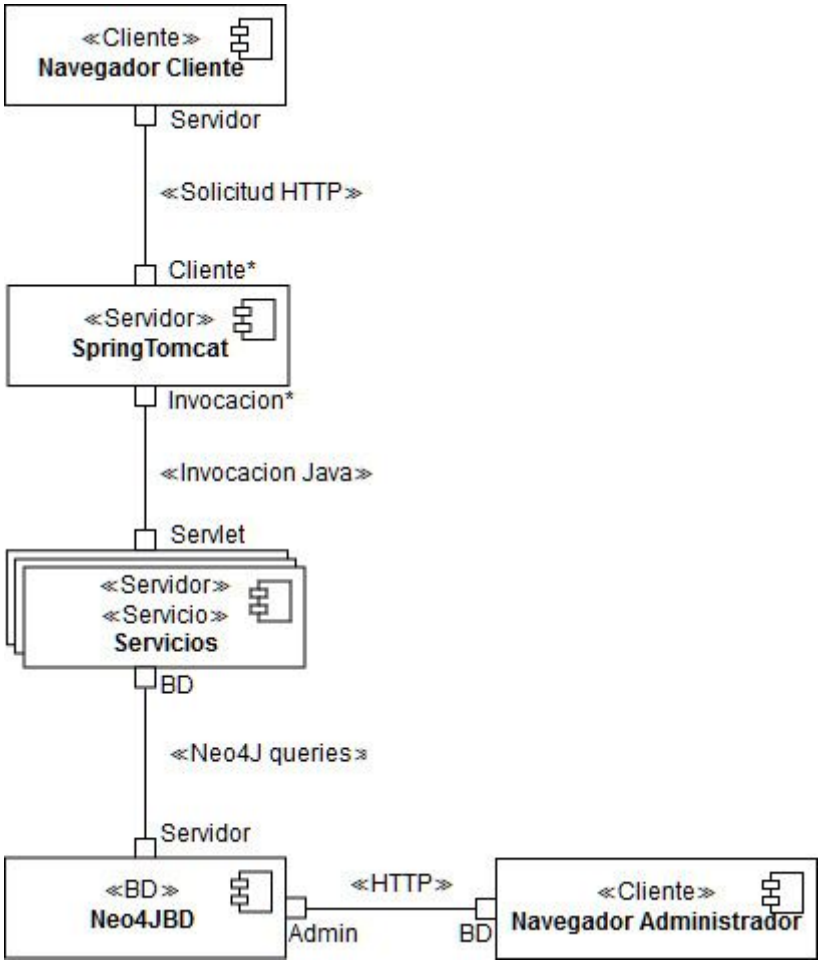
3.1.4.b Acotar requisitos

1. Asunto: Alguno de los requisitos iniciales se han visto afectados por decisiones en la tecnología o por falta de tiempo.

2. Decisión tomada.
3. Asunciones: Los cambios no afectan en gran medida a la funcionalidad core de la aplicación.
4. Alternativas: Los requisitos antiguos: Se mostraba el número de votos totales en el perfil de un usuario y los catálogos de encuestas se podían filtrar y ordenar.
5. Argumento: Debido a la naturaleza de la tecnología usada para la base de datos, se daban ocurrencias de relaciones cíclicas al obtener ciertos objetos de la base de datos y serializarlos a JSON. Se solucionó el problema para ciertos módulos de la aplicación, pero no para todos.
6. Implicaciones: Se han recortado algunos requisitos y se han creado submódulos adicionales para suplir el problema.
7. Decisiones relacionadas: 3.1.4.b
8. Requisitos relacionados: RF5, RF6 y RF7.
9. Elementos arquitecturales afectados: Vista de módulos.

3.2 Vista de componentes y conectores

3.2.1 Presentación primaria



3.2.2 Catálogo de la vista

3.2.2.a Elementos y sus propiedades

- Navegador Cliente: puede haber múltiples instancias al mismo tiempo.
- SpringTomcat: contenedor de servlets con un único servlet, DispatcherServlet.
  Enruta la petición HTTP hacia los controladores correspondientes, que Spring inicializa con anterioridad.
  - Cliente:
    -Protocolo: HTTP 1.1.
    -Puerto: 8080.
- Servicios: cumplen cada uno de los requisitos de la aplicación. Llamados desde el servlet inicializado por Spring, DispatcherServlet.
  - Servlet:
    -Contexto: JVM, invocación a métodos.

- Neo4JBD:
    - □ Servidor BD: todos los servicios acceden a la BD. Spring oculta los detalles de implementación de esos accesos (clases java) y sus operaciones (queries) (implementacion externa).
      -Protocolo: Bolt.
    - □ Admin: acceso a la página web de administración de Neo4J (implementacion externa).
      -Protocolo: HTTP 1.1.
      -Puerto: 7474.
      -Usuario: neo4j.
      -Password: 1234.
- Navegador Administrador: puede haber múltiples instancias al mismo tiempo.

3.2.2.b Interfaces y sus propiedades

El orden de los parámetros que van en el body es irrelevante. Los tipos de datos están escritos en términos de Javascript. Todos los datos son sensibles a mayúsculas y minúsculas.

- SpringTomcat: puerto Cliente.
    - ○ GET /login.
      head: application/json
      body: {nombreUsuario: String, password: String}
      Success return: 200 OK {}
      Error return: 404 NOT_FOUND {}
    - ○ POST /crearCuenta
      head: application/json
      body: {nombreUsuario: String, password: String}
      Success return: 200 OK {}
      Error return: 403 FORBIDDEN {}
    - ○ POST /perfilUsuario
      head: application/json
      body: {nombre: String, pais: String, nacimiento: Date, nombreCuenta: String, (optional)Sexo: String}
      Success return: 200 OK {}
      Error return: 403 FORBIDDEN {}
    - ○ GET /perfilUsuario
      head: application/json
      body: {nombreCuenta: String}
      Success return: 200 OK {nombre: String, pais: String, nacimiento: Date, Sexo: String, id: Integer}
      Error return: 403 FORBIDDEN {}

- GET /perfilUsuario/{id}
  head: application/json
  body: {}
    Success return: 200 OK {nombre: String, pais: String, nacimiento: Date, Sexo: String, id: Integer}
    Error return: 403 FORBIDDEN {}
- GET /buscarPerfilUsuario
  head: application/json
  body: {busqueda: String}
    Success return: 200 OK [{nombre: String, pais: String, nacimiento: Date, Sexo: String, id: Integer}]
    Error return: 403 FORBIDDEN {}
- PUT /perfilUsuario/{id}
  head: application/json
  body: {nombre: String, pais: String, nacimiento: Date, Sexo: String, (optional)nombreCuenta: String}
    Success return: 200 OK {}
- POST /crearEncuesta
  head: application/json
  body: {pregunta: String, respuestas: [String], autor: String}
    Success return: 200 OK [{nombre: String, pais: String, nacimiento: Date, Sexo: String, id: Integer}]
- GET /encuesta
  head: application/json
  body: {nombreUsuario: String, (optional)usuarioEncuesta: String}
    Success return: 200 OK [{pregunta: String, respuestas: [String], votosPorRespuesta: [Integer], votos: Integer, votada: boolean}]
    Error return: 404 NOT_FOUND {}
- GET /buscarEncuesta
  head: application/json
  body: {nombreUsuario: String, pregunta: String}
    Success return: 200 OK [{pregunta: String, respuestas: [String], votosPorRespuesta: [Integer], votos: Integer, votada: boolean}]
    Error return: 404 NOT_FOUND {}
- PUT /encuesta
  head: application/json
  body: {nombreCuenta: String, nombreEncuesta: String, nRespuesta: Integer}
    Success return: 200 OK {}

- Servicios: puerto Servlet.
  - Todos los métodos de las clases controller.
- Neo4JBD: puerto Servidor BD.
  - Neo4J Query Languaje: Cypher.
    https://neo4j.com/developer/cypher-query-language/

- Neo4JBD: puerto Admin.
    - Interfaz web de la página web de administración de la BD Neo4J.

3.2.3 Exposición de razones

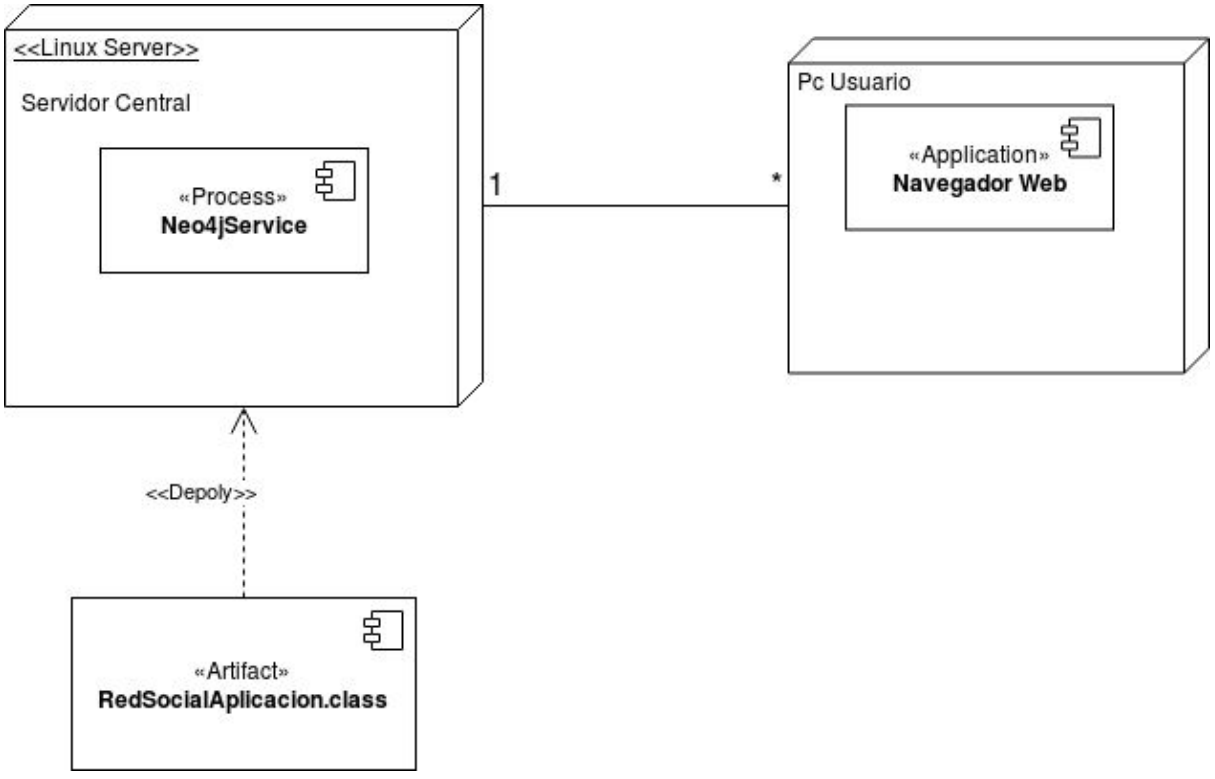
3.2.3.a Nombrado de interfaces

1. Asunto: tener una manera clara y ordenada de relacionar los comportamientos del API y los servicios que la aplicación ofrece con sus distintas implementaciones y representaciones.
2. Decisión y estado: llamadas a URL y nombres de métodos iguales. Atributos de objetos de backend con representación directa en los JSON usados en frontend. Implementado.
3. Argumento: Spring framework facilita enormemente ésta tarea.
4. Implicaciones: es posible simplificar los diagramas, y convertir las relaciones que en éstos saldrían a simples enlaces en el documento.
5. Elementos arquitecturales afectados: diagramas de módulos y de CyC.

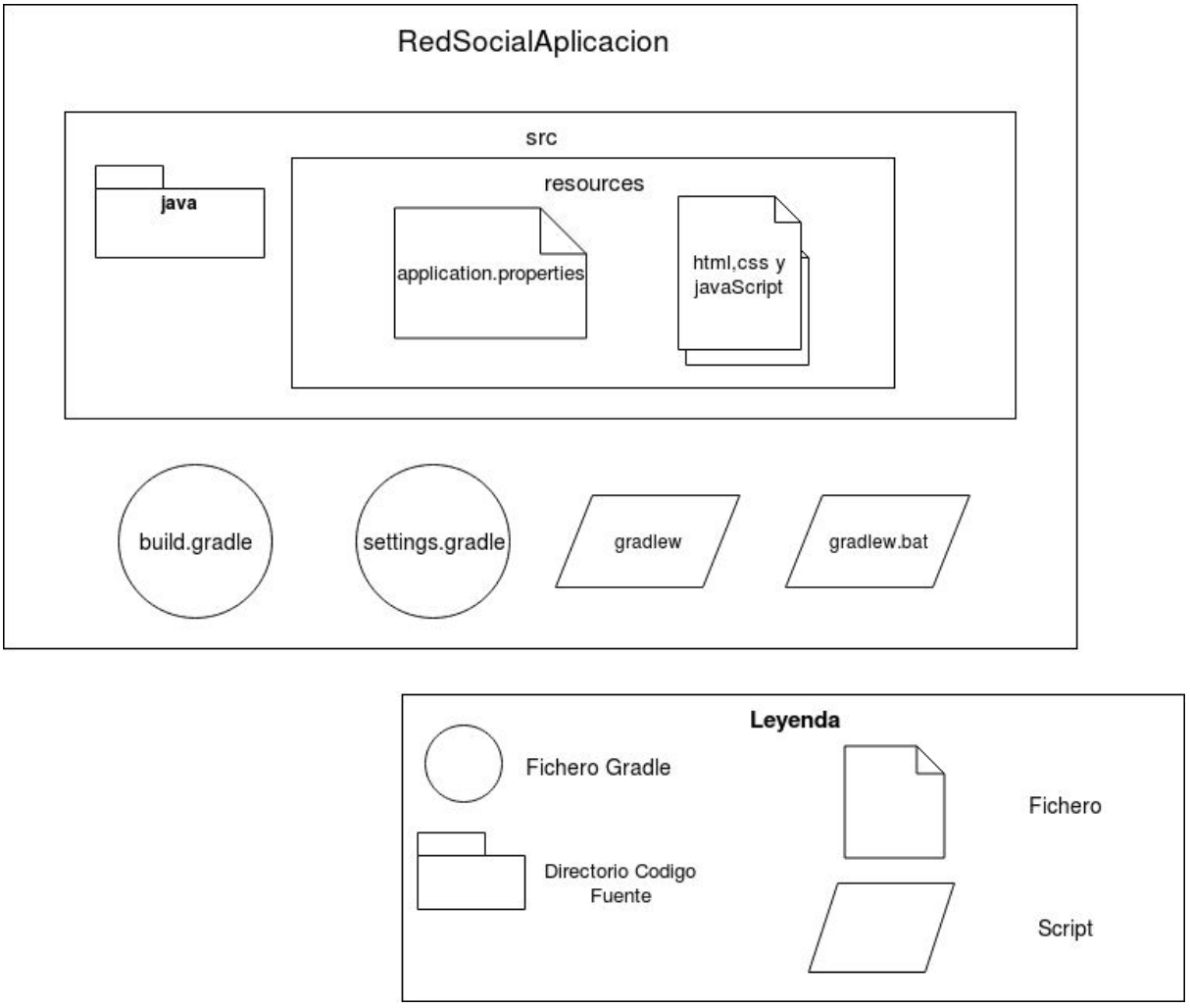
3.3 Vista de Distribución

3.3.1 Presentación primaria

3.3.1.a Estilo de Despliegue



3.3.1.b Estilo de Instalación



3.3.1.c Estilo de asignación de trabajo

Módulo	**Submódulo**	**Asignado**
redsocial (Servidor)	Dominio	Adrián
	Servicio	Grupo 3
	spring	Iván
	Usuario	Grupo 3

cliente	-	Iván

3.3.2 Catálogo de la vista

3.3.2.a Elementos y sus propiedades.

Servidor Central

- Puerto http 8080 habilitado.
- Proceso Neo4jService es un servicio del sistema, debe estar siempre en funcionamiento.
- Disco duro de alta velocidad, preferiblemente SSDs. Alta capacidad de Memoria.

Pc Usuario

- Navegador Web debe soportar JavaScript

RedSocialAplicacion

- Número alto de Querys a la BD, uso intensivo en Disco y Memoria.

Gradle

- La aplicación se construye y ejecuta mediante la herramienta Gradle, por lo que debe estar instalada en su versión 4.0 o superior.
- La aplicación se ejecuta mediante el comando `gradle bootRun`

Java

- La aplicación usa Java 1.8.

.properties

- El fichero application.properties contiene la configuración para la conexión a la base de datos Neo4j. Por defecto supone que la base de datos está en la misma máquina que la aplicación y se expone en el puerto *bolt* por defecto, se usa usuario “neo4j” y contraseña “1234”.

3.3.2.b Interfaces de los elementos

Servidor Central

-Identidad y recursos: Llamadas al sistema Linux, proporcionando recursos Hardware, API de Neo4j ofrecida por el servicio, ofreciendo entrada/salida en la base de datos, y envío y recepción de mensajes http a través del puerto 8080.
-Variabilidad: Es posible configurar variables de entorno, como por ejemplo el uso de otro puerto que no sea 8080 para la comunicación con el servidor.

PC Usuario

-Identidad y recursos: Llamadas al motor de JavaScript del navegador, envío y recepción de mensajes HTTP.
-Tipos de datos y constantes: Uso estandarizado del protocolo http, uso de JSON para el envío y recepción de datos en el cuerpo de los mensajes.

-Manejo de errores: Si ocurren errores en el lado del cliente, solamente nos queda hacer comprobaciones extra en el lado del servidor. En producción se usa la consola y debugger Javascript presente en las herramientas de desarrollador de Fifrefox o Chrome.

RedSocialAplicación
Interfaz REST proporcionada por los submódulos de spring: CuentaController, PerfilController y EncuestaController.

Gradle
-Identidad y recursos: API oficial de Gradle, ficheros de script gradlew.sh y gradlew.bat. Proporcionan los paquetes de dependencias para nuestra aplicación y se gestiona el encendido/apagado de nuestra aplicación.

Java
-Identidad y recursos: API oficial de Java, nos da el runtime de la aplicación.
-Variabilidad: dar la opción -ea para habilitar las aserciones en el código y dar más seguridad a la aplicación.
-Manejo de errores: Output de la traza de la excepción.

3.3.3 Exposición de razones

3.3.3.a Distribución del trabajo

1. Asunto: Repartir los módulos obtenidos en la vista de módulos entre los miembros del equipo como unidades de trabajo.
2. Decisión tomada
3. Asunciones: Los módulos identificados a la hora de tomar la decisión son los finales.
4. Implicaciones: Cada miembro del grupo deberá asumir responsabilidad de los módulos a su cargo. Se genera un estilo de asignación de trabajo en la Vista de Distribución.
5. Decisiones relacionadas: 3.1.4.a
6. Elementos arquitecturales relacionados: Estilo de asignación de trabajo 3.3.1.c

4. Mapeo entre vistas

Podemos ver el módulo “Modelo” que contiene varios módulos identificados en la Vista de módulos como un mapeo directo del componente “servicios” en la vista de C&C, el componente “RedSocialAplicación” en el estilo de despliegue de la vista de Distribución o el artefacto “Java” en el estilo de instalación.
También se mapea el componente Neo4jBD de la vista de C&C con el artefacto “Neo4jService” del estilo de despliegue en la vista de distribución.

5. Exposición de Razones

5.1 Neo4j como base de datos.

1. Asunto: Se elige usar la base de datos noSQL de grafos Neo4j ante otras posibilidades.
2. Decisión tomada e implementada.
3. Alternativas: Se barajaron posibles alternativas para BD tanto SQL(Postgre,MySql,H2) como noSQL (Mongo,Redis).
4. Argumento: Se tomó la decisión final de adoptar Neo4j porque tras investigar se vió que el modelo de grafos encajaba perfectamente con la aplicación, ya que en una red social predominan las relaciones entre los datos. Además se integraba sin ningún tipo de problemas con el framework a usar, Spring Boot, y existe abundante documentación e información. Además llamó la atención la oportunidad de experimentar con algo con lo que no habíamos trabajado.
5. Implicaciones: Se crea un modelo de datos específico para la base, también las operaciones de lectura/escritura que lo respaldan.
6. Decisiones relacionadas: 5.2
7. Requisitos relacionados: RF1,RF2,RF3,RF4,RF5,RF6,RF7,RF9.

5.2 Spring Boot para backend.

1. Asunto: Uso del framework Spring Boot como apoyo central para construir el backend.
2. Decisión tomada e implementada.
3. Alternativas: No se llegó realmente a considerar ninguna alternativa, quizás se podría haber aplicado el stack MEAN que hemos estudiado en la asignatura de Sistemas y Tecnologías Web en este curso.
4. Asunciones: Ambos miembros del equipo conocen Spring y han trabajado con él.
5. Argumento: El equipo contaba con experiencia en el desarrollo Web con Spring, facilitandonos la comunicación con el frontend como con la base de datos.
6. Implicaciones: Desde un inicio se diseña la aplicación centrándose en Spring, creándose así una fuerte dependencia de las funcionalidades de la aplicación con la infraestructura de Spring.
7. Requisitos relacionados: Todos
8. Elementos arquitecturales relacionados: Vista de módulos, vista de C&C.

5.3 Insensibilidad a mayúsculas y minúsculas

1. Asunto: el API de la aplicación siempre devolverá el mismo resultado a las diferentes peticiones aún si los parámetros cambian minúsculas por mayúsculas y viceversa.
2. Decisión y estado: no implementado/por implementar.
3. Asunciones: el usuario debe tener constancia de esto.

4. Alternativas: pasar los input del usuario a minúsculas antes de hacer llamadas a la BD.
5. Argumento: los repositorios deberían usar métodos con *custom queries*, ya que no existen palabras clave con las que nombrar esos métodos que permita construir una query automática *case insensitive*, y los desarrolladores no han tenido tiempo de aprender ese lenguaje.
6. Elementos afectados: repositorios del backend.

5.4 Seguridad

1. Asunto: el API de la aplicación no es seguro frente a cambios ilícitos en los datos del frontend.
2. Decisión y estado: usar JWT (JSON Web Tokens) en nuestra API REST. Módulo creado sin funcionalidad/por implementar.
3. Asunciones: buena y comprobada forma de tener seguridad en los datos.
4. Alternativas: pedir contraseña en las operaciones de modificación.
5. Argumento: los API REST no tienen estado en el servidor, así que para garantizar la integridad de los datos necesitamos un método de cifrado, como mínimo.
6. Requisitos relacionados: RF1 y RF8.
7. Elementos afectados: servicio de validación en backend, API REST.