**Proyecto de**

**Ingeniería de Software**

# 

# 

# 

**Descripción de la Arquitectura**

**Grupo 01**

|  |  |
| --- | --- |
|  |  |
| **Integrante**  Santiago Acevedo | **CI**  4.996.001-3 |
| Belén Brandino | 4.850.415-3 |
| Manuel Eirea | 5.411.499-2 |
| Leonardo Fallini | 5.288.229-8 |
| Rodrigo Gallardo | 4.921.178-9 |
| Guillermo Guerrero | 4.817.343-1 |
| Facundo Gutierrez | 5.479.610-6 |
| Sebastián Lagomarsino | 4.803.767-1 |
| Federico Peña | 4.862.495-7 |
| Eliana Rosselli | 5.220.298-5 |
| José María Souto | 5.129.801-0 |
| Paula Tavidian | 4.925.631-1 |

|  |  |
| --- | --- |
| Director de proyecto: | Diego Vallespir |

**Descripción de la Arquitectura**

**Versión 2.0**

Historia de Revisiones

|  |  |  |  |
| --- | --- | --- | --- |
| Fecha | Versión | Descripción | Autor |
| 10/09/2020 | 1.0 | Creación del documento | Leonardo Fallini |
| 12/09/2020 | 1.1 | Modificación Del Documento | Guillermo Guerrero,  Leonardo Fallini |
| 13/09/2020 | 1.2 | Revisión SQA | Paula Tavidian |
| 17/09/2020 | 1.3 | Modificación Del Documento | Manuel Eirea, Facundo Gutierrez, Belén Brandino |
| 20/09/2020 | 1.4 | Refinamiento de casos de uso,  actualización de diagramas | Guillermo Guerrero,  Leonardo Fallini |
| 20/09/2020 | 1.5 | Actualización de componentes,  actualización de diagramas | Guillermo Guerrero,  Leonardo Fallini |
| 27/09/2020 | 1.6 | Revisión SQA | Paula Tavidian |
| 3/10/2020 | 1.7 | Actualizar CU | Guillermo Guerrero, Leonardo Fallini |
| 11/10/2020 | 1.8 | * Agregada descripción sobre los actores de los casos de uso. * Eliminados los actores "Browser" y "Usuario" pues no iniciaban ningún caso de uso. * Se muestran los casos de uso más estructurados. * Actualizados los nombres de componentes, funciones y demás a idioma Inglés para mantener coherencia. * Se cambiaron los diagramas donde se muestra el diseño de los casos de uso * Se eliminaron las interfaces en los componentes, debido a que no estaba contemplado tener múltiples implementaciones de las clases. * Se actualizaron los nombres de las clases y de los componentes. | Guillermo Guerrero, Leonardo Fallini |
| 19/10/2020 | 1.9 | Se agregó la sección 8 de restricciones del lenguaje Javascript | Eliana Rosselli |
| 1/11/2020 | 1.10 | Modificaciones sugeridas por el cliente sobre el documento "Descripción de la Arquitectura v2" que se dejó disponible. | Leonardo Fallini |
| 7/11/2020 | 2.0 | Actualización diagramas | Leonardo Fallini |

**Contenido**

# 

[**1. Introducción**](#_rqdqk5z2q9ia) **6**

[1.1. Propósito](#_yvgby4lg4t6i) 6

[1.2. Alcance](#_2et92p0) 6

[1.3. Definiciones, siglas y abreviaturas.](#_xmbn31di6dn0) 6

[1.4. Referencias](#_kzeghpb2t96h) 6

[1.5. Visión general](#_5s44kf28u37h) 6

[**2. Vista del Modelo de Casos de Uso**](#_w2fws98dw791) **6**

[2.1. Diagrama de Casos de Uso relevantes a la Arquitectura](#_2s8eyo1) 6

[2.2. Descripción de los actores](#_oha4iqx75ytx) 7

[2.2.1. Aplicación móvil RP](#_ri0axujptjed) 7

[2.2.2. Usuario final / Usuario de Aplicación Móvil RP](#_ry2ghtmbj5ym) 7

[2.3. Casos de Uso relevantes a la Arquitectura](#_9aj3817yo1fp) 8

[2.3.1. Consideraciones generales](#_jfq7l7seak0m) 8

[2.3.2. Inicializar componente](#_3la2dvz5kaas) 8

[2.3.3. Autenticar usuario final](#_sfmc2itz3ryb) 9

[2.3.4. Obtener token para usuario final autenticado](#_pgnk8qcpb9uj) 10

[2.3.5. Refrescar token para usuario final autenticado](#_xk93h0kfqjbi) 11

[2.3.6. Obtener Información de un Usuario Final Autenticado](#_d7cjnrk1hskd) 12

[2.3.7. Verificar Token Obtenido para Usuario Final Autenticado](#_j2katubi3yab) 13

[2.3.8. Finalizar Sesión en OP para Usuario Final Autenticado](#_34zy23qiyakz) 13

[**3. Vista del Modelo de Diseño**](#_44sinio) **14**

[3.1. Diseño de Clases](#_3ps2v1y68aas) 14

[3.2. Diseño de Casos de Uso](#_1ci93xb) 15

[3.2.1. Inicializar componente](#_m9o5bbfzibk3) 15

[3.2.2. Autenticar usuario final](#_9ulca39hydw9) 16

[3.2.3. Obtener token para usuario final autenticado](#_b1j4otvw1ym1) 16

[3.2.4. Refrescar token para usuario final autenticado](#_jso76dcekb27) 17

[3.2.5. Obtener Información de Usuario Final Autenticado](#_7smm9yfn67f8) 18

[3.2.6. Verificar Token Obtenido para Usuario Final Autenticado](#_sv1cvfk1ky8w) 18

[3.2.7. Finalizar Sesión en OP para Usuario Final Autenticado](#_25gg7k8sjta) 19

[**4. Trazabilidad desde el Modelo de Diseño al Modelo de Implementación**](#_sqyk7gz9ne99) **19**

[**5. Vista del Modelo de Implementación**](#_megzfcxs8avh) **20**

[5.1. Componentes](#_ulu23j3tw1l7) 20

[5.1.1. Interface](#_1pxezwc) 20

[5.1.1.1. Descripción de Procedimientos](#_ncnjxgx8p9f1) 20

[5.1.2. Configuration](#_laarvh5gqfq5) 22

[5.1.2.1. Descripción de procedimientos](#_d60i3pntrkge) 22

[5.1.3. Requests](#_kadly2cht9b3) 22

[5.1.3.1. Descripción de procedimientos](#_i42odgjcg2iy) 23

[5.1.4. Security](#_cfgepv758m0k) 23

[5.1.4.1. Descripción de procedimientos](#_k88orna44xqr) 24

[**6. Vista del Modelo de Distribución**](#_jitomrkqzy48) **25**

[6.1. Diagrama de Distribución](#_3o7alnk) 25

[6.2. Nodos](#_74sjgjivhzat) 25

[6.2.1. Dispositivo Móvil](#_50um1yp9kvgv) 25

[6.2.2. Servidor Usuario gub.uy](#_kbtc0xy90nuz) 25

[6.3. Conexiones](#_y6gohxrqqke1) 25

[6.3.1. HTTPS](#_41mghml) 25

[**7. Vista del Modelo de Procesos**](#_d2c0475w62ma) **25**

[7.1. Inicializar componente](#_x5l6xgd0u6ue) 26

[7.2. Autenticar usuario final](#_g11d25fzxhsu) 26

[7.3. Obtener Token para Usuario Final Autenticado](#_l14wnpbos9pi) 26

[7.4. Obtener Información de Usuario Final Autenticado](#_v2nc4qusa8k0) 27

[7.5. Verificar Token Obtenido para Usuario Final Autenticado](#_hbc1uf97ntj4) 28

[7.6. Finalizar Sesión en OP para Usuario Final Autenticado](#_qmairzqaao62) 28

[**8. Restricciones del diseño respecto al lenguaje Javascript**](#_4ry8rds9k0qp) **29**

[8.1 Restricciones generales del paradigma basado en prototipos](#_n0wnit1c3rt1) 29

[8.2 Programación asíncrona](#_aphu1k2e358v) 29

# 

# 

# 1. Introducción

## 1.1. Propósito

Este documento proporciona una apreciación global y comprensible de la arquitectura del sistema usando diferentes puntos de vista para mostrar distintos aspectos del sistema. Se intenta capturar y fundamentar las decisiones de arquitectura críticas que han sido hechas en el sistema.

Uno de los objetivos perseguidos durante el planeamiento de la arquitectura presentada a continuación fue la búsqueda de su adaptabilidad a la mayor cantidad de lenguajes de programación de plataformas *mobile*.

El documento está destinado al equipo de trabajo y al cliente. Esta versión no debe tomarse como final ya que posiblemente se introduzcan cambios en el diseño de la arquitectura a medida que la misma es refinada.

## 1.2. Alcance

La descripción de la arquitectura realizada en el presente documento aplica únicamente al componente que pretende integrarse con aplicaciones móviles. No se incluye una descripción del diseño de la aplicación móvil que será utilizada como ejemplo para demostrar la integración con el componente SDK.

## 1.3. Definiciones, siglas y abreviaturas.

La terminología utilizada en este documento se encuentra disponible en el documento: Glosario en su versión: 1.5.

## 1.4. Referencias

* Glosario. Versión 1.5.
* Especificación de requerimientos. Versión 1.8.

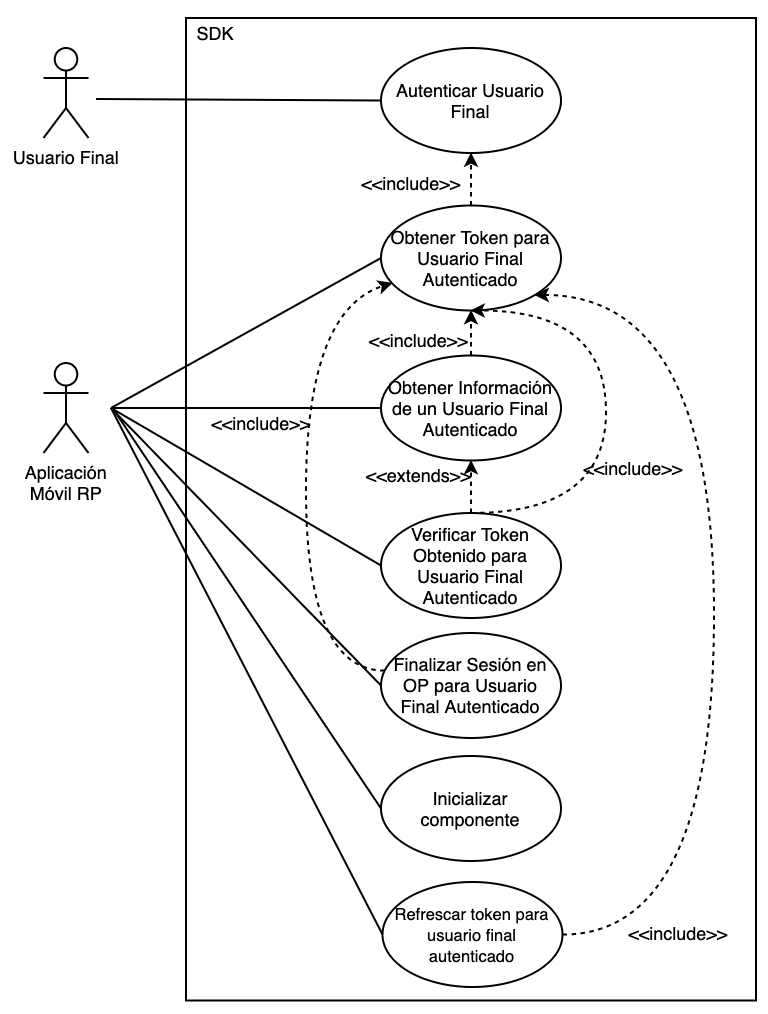
## 1.5. Visión general

El resto del documento realiza una aproximación de la arquitectura del componente SDK utilizando algunas vistas del modelo 4+1. De esta forma se presenta la arquitectura desde distintos puntos de vista para g

enerar un mayor entendimiento del diseño del producto a implementar en etapas posteriores.

# 2. Vista del Modelo de Casos de Uso

## 2.1. Diagrama de Casos de Uso relevantes a la Arquitectura



## 2.2. Descripción de los actores

### 2.2.1. Aplicación móvil RP

Aplicación móvil que utiliza el componente SDK para acceder a las funcionalidades de los *EndPoints* publicados por la API ID Uruguay.

### 2.2.2. Usuario final / Usuario de Aplicación Móvil RP

El usuario final es una persona que interactúa con la Aplicación Móvil RP. A través del componente el usuario final será capaz de iniciar una sesión en la Aplicación móvil RP mediante su Usuario gub.uy.

## 2.3. Casos de Uso relevantes a la Arquitectura

### 2.3.1. Consideraciones generales

* Los parámetros se encuentran especificados en la sección 4.2 donde se puede observar en detalle para cada caso de uso los parámetros que utiliza. Para una descripción de cada parámetro referirse al Glosario.
* Cuando se menciona que los parámetros del componente son almacenados en el sistema, se refiere a que son guardados en memoria principal, durante la ejecución de la Aplicación móvil RP. En caso de cerrarse la aplicación, el componente no mantendrá los parámetros para la próxima vez que se haga uso del mismo.
* Dato **bien formado** se refiere a datos de tipo correcto y no maliciosos. Por ejemplo, si se solicita el *client\_id*, un *client\_id* bien formado es un string con un código numérico.
* Dato **mal formado** refiere a un dato que puede ser de tipo incorrecto, de un tamaño excesivamente grande, vacío o nulo, o mal intencionado. Por ejemplo, si el dato será utilizado por un intérprete, se podría ingresar un dato malicioso para intentar hacer una inyección de comando.
* Dato **válido** se refiere a un dato que está bien formado y a su vez tiene validez en el contexto de la aplicación. Por ejemplo, ingresar un *client\_id* que fue registrado y aprobado por el OP y permite acceder a ciertas funcionalidades.
* Dato **invalido** se refiere a un dato que está bien formado, pero no tiene validez en el contexto de la aplicación. Por ejemplo, ingresar un *client\_id* que no está registrado en el OP.
* Un parámetro del sistema se considera bien formado al momento de su utilización si cumple que no es vacío, o en el caso particular del parámetro *id\_token*, si cumple con la especificación del protocolo *OpenID*.
* Cuando se menciona que el componente se encuentra inicializado, se refiere a un estado en el sistema en el cual es posible invocar correctamente al *Authorization Endpoint* de la API ID Uruguay.

### 2.3.2. Inicializar componente

|  |  |
| --- | --- |
| **Nombre**: | Inicializar Componente. |
| **Objetivo**: Establecer los parámetros necesarios para la comunicación del componente con el OP. Estos parámetros son utilizados en todas las comunicaciones entre el componente y el OP, por lo que establecerlos una única vez le brinda a la Aplicación móvil RP mayor comodidad en el uso del componente. | |
| **Actores**:  Aplicación móvil RP. | |
| **Precondiciones**:  No hay precondiciones. | |
| **Descripción**:  El caso de uso comienza cuando la Aplicación móvil RP requiere hacer uso de las funcionalidades del componente.  Este caso de uso permite a la Aplicación móvil RP inicializar el componente con parámetros que serán utilizados para establecer la comunicación con el OP. Dichos parámetros pueden estar bien formados o mal formados, por lo que el propio componente se encarga de verificarlos y almacenarlos. Adicionalmente se retorna al usuario un código indicando la validez o no de los parámetros. | |
| **Flujo normal**:   1. La Aplicación móvil RP ingresa los parámetros deseados. 2. El sistema realiza la validación y el almacenamiento de los parámetros y retorna al usuario un mensaje indicando que la validación fue exitosa. 3. Fin CU. | |
| **Flujos Alternativos:**  2A. Existen parámetros ingresados que están mal formados  2A.1 El sistema realiza la validación de los parámetros y retorna a la Aplicación móvil RP un mensaje indicando la presencia de parámetros mal formados. El sistema no almacena ningún parámetro ingresado por la Aplicación móvil RP.  2A.2 Fin CU. | |
| **Poscondiciones**:  El sistema almacena los parámetros ingresados por la Aplicación móvil RP. El componente se encuentra inicializado. | |

### 2.3.3. Autenticar usuario final

Una vez completado el flujo normal de este caso de uso, se satisface el requerimiento *R01*.

|  |  |
| --- | --- |
| **Nombre**: | Autenticar usuario final. |
| **Objetivo**: Permitir al a la Aplicación móvil RP obtener un Code para posteriormente ser intercambiado por un Token.  Permitir al usuario final brindar el acceso de sus datos a la aplicación móvil RP. | |
| **Actores**:  Usuario final. | |
| **Precondiciones**:  El sistema se encuentra inicializado. | |
| **Descripción**:  El caso de uso comienza cuando el Usuario final desea autenticarse en la Aplicación móvil RP mediante su usuario de Usuario gub.uy.  El sistema establece una comunicación con el Authorization Endpoint para permitir que el usuario final se autentique y autorice a la Aplicación móvil RP a acceder a sus datos. El sistema obtendrá la respuesta que genera dicho endpoint como resultado de la autenticación y autorización del usuario final. Esta respuesta será validada y retornada a la Aplicación móvil RP. | |
| **Flujo normal**:   1. El Usuario final selecciona la opción de iniciar sesión con Usuario gub.uy en la Aplicación móvil RP. 2. El sistema genera una *Authentication Request* (utilizando los *Authentication Request Params*) y la envía al *Authorization Endpoint* del OP haciendo uso del navegador web del dispositivo móvil. 3. A través del navegador web del dispositivo móvil, el Usuario final se autentica y autoriza a la *Aplicación móvil RP* a acceder a sus datos. Una vez realizados estos pasos, el Usuario final es redirigido a la aplicación móvil RP. 4. El sistema obtiene una *Authentication Response* generada por el Authorization Endpoint del OP producto de la autenticación y autorización del usuario final. 5. El sistema valida la respuesta, almacena los parámetros de la misma y por último los retorna a la Aplicación móvil RP en forma de Authentication Response Params. 6. Fin CU. | |
| **Flujos Alternativos:**  4A. El usuario final no autoriza a la aplicación móvil RP a acceder a sus datos.  4A.1 El sistema obtiene una Authentication Response generada por el Authorization Endpoint del OP producto de la autenticación y no autorización del usuario final.  4A.2 El sistema valida la respuesta y retorna a la Aplicación móvil RP un mensaje de error correspondiente.  4A.3 Fin CU.  6. La respuesta del Authorization Endpoint no es válida.  6A.1 El sistema retorna a la Aplicación móvil RP un mensaje de error correspondiente.  6A.2 Fin CU.  G1. El Authorization Endpoint no está activo.  Pendiente...  G2. El dispositivo móvil no cuenta con conexión a internet.  Pendiente... | |
| **Poscondiciones**:  El sistema obtiene un *code* válido y lo almacena. | |

### 

### 2.3.4. Obtener token para usuario final autenticado

Este caso de uso se corresponde con el requerimiento R02.

|  |  |
| --- | --- |
| **Nombre**: | Obtener un *token* para un Usuario final autenticado. |
| **Objetivo**: Permitir a la Aplicación móvil RP obtener un *token* (mediante un *code*) para posteriormente utilizarlo con el fin de obtener información del Usuario final previamente autenticado. | |
| **Actores**:  Aplicación Móvil RP. | |
| **Precondiciones**:  El sistema se encuentra inicializado.  El sistema tiene almacenado el parámetro *code*. | |
| **Descripción**:  El caso de uso comienza cuando la Aplicación móvil RP requiere un *token*.  El sistema establecerá una comunicación con el *Token Endpoint* del OP para obtener un *token*. La respuesta obtenida será validada por el componente y los parámetros obtenidos (entre ellos el *token*) serán almacenados en el sistema y retornados a la Aplicación móvil RP. | |
| **Flujo normal**:   1. La Aplicación móvil RP solicita al sistema obtener un *token* para un Usuario final autenticado. 2. El sistema genera una *Token Request* (utilizando los *Token Request* Params) que es enviada al *Token Endpoint* del OP. 3. El sistema obtiene una *Token Response* generada por el *Token Endpoint* del OP producto del pedido realizado. 4. El sistema valida la respuesta, almacena los parámetros de la misma y los retorna a la Aplicación móvil RP en forma de *Token Response Params*. 5. Fin CU. | |
| **Flujos Alternativos:**  2A. Los *Token Request Params* utilizados en la *Token Request* son inválidos.  2A.1 El sistema obtiene una Token Response generada por el Token Endpoint del OP que indica el error correspondiente.  2A.2 El sistema valida la respuesta y la retorna a la Aplicación móvil RP.  2B. El parámetro code está expirado  2A.3 Fin CU. | |
| **Poscondiciones**:  El sistema obtiene un *token* válido y lo almacena. | |

### 

### 2.3.5. Refrescar token para usuario final autenticado

Este caso de uso se corresponde con el requerimiento R04.

|  |  |
| --- | --- |
| **Nombre**: | Refrescar *token* para un Usuario final autenticado |
| **Objetivo**: Permitir a la Aplicación móvil RP obtener un nuevo *token* para reemplazarlo por el token actual vencido. | |
| **Actores**:  Aplicación móvil RP. | |
| **Precondiciones**:  El sistema tiene un *token* almacenado para refrescar.  El sistema se encuentra inicializado. | |
| **Descripción**:  El caso de uso comienza cuando la Aplicación móvil RP requiere un nuevo *token* debido a que su token actual está vencido y debe ser refrescado.  El sistema establecerá una comunicación con el *Token Endpoint* del OP para obtener un nuevo *token*. La respuesta obtenida será validada por el componente y los parámetros obtenidos (entre ellos el *token*) serán almacenados y retornados a la Aplicación móvil RP. | |
| **Flujo normal**:   1. <<Incluye>> "Obtener un *token* para un Usuario final autenticado" pero se genera la Token Request utilizando los Refresh Token Request Params. | |
| **Flujos Alternativos:** | |
| **Poscondiciones**:  El sistema obtiene un nuevo *token* válido y lo almacena. | |

### 2.3.6. Obtener Información de un Usuario Final Autenticado

Este caso de uso se corresponde con el requerimiento R03.

|  |  |
| --- | --- |
| **Nombre**: | Obtener información de un Usuario final autenticado. |
| **Objetivo**: Permitir a la Aplicación móvil RP obtener los datos que el Usuario final aceptó compartir con la Aplicación móvil RP. | |
| **Actores**:  Aplicación móvil RP. | |
| **Precondiciones**:  El sistema se encuentra inicializado.  El sistema tiene almacenado un *token*. | |
| **Descripción**:  El caso de uso comienza cuando la Aplicación móvil RP quiere obtener datos sobre un Usuario final.  El sistema establecerá una comunicación con el *UserInfo Endpoint* del OP para obtener la información. La respuesta obtenida será validada por el sistema y los parámetros obtenidos serán retornados a la Aplicación móvil RP. | |
| **Flujo normal**:   1. La Aplicación móvil RP solicita al sistema obtener información del Usuario final autenticado. 2. El sistema genera una *User Info Request* (utilizando los *User Info Request Params*) que es enviada al *UserInfo Endpoint* del OP. 3. El sistema obtiene una *User Info Response* generada por el *UserInfo Endpoint* del OP producto del pedido realizado. 4. El sistema valida la respuesta y los retorna a la Aplicación móvil RP en forma de *User Info Response Params*. 5. Fin CU. | |
| **Flujos Alternativos:**  2A. Los *User Info Request Params* utilizados en la *User Info Request* son inválidos.  2A.1 El sistema obtiene una *User Info Response* generada por el *User Info Endpoint* del OP que indica el error correspondiente.  2A.2 El sistema valida la respuesta y la retorna a la Aplicación móvil RP.  2A.3 Fin CU. | |
| **Poscondiciones**: No hay cambio de estado. | |

### 

### 2.3.7. Verificar Token Obtenido para Usuario Final Autenticado

Este caso de uso se corresponde con el requerimiento R05.

|  |  |
| --- | --- |
| **Nombre**: | Verificar *token* obtenida para un Usuario final autenticado. |
| **Objetivo**: Permitir a la Aplicación móvil RP verificar un *token* obtenido previamente y así generar una mayor seguridad en la comunicación con el OP. | |
| **Actores**:  Aplicación móvil RP. | |
| **Precondiciones**:  El sistema se encuentra inicializado.  El sistema tiene almacenado un *token*. | |
| **Descripción**:  El caso de uso comienza cuando la Aplicación móvil RP requiere validar un token.  El sistema establecerá una comunicación con el JWKS Endpoint del OP para obtener información útil para realizar dicha validación. La respuesta obtenida será validada por el sistema. Posteriormente se realiza la validación del token y el resultado es devuelto a la Aplicación móvil RP . | |
| **Flujo normal**:   1. La Aplicación móvil RP solicita al sistema validar un token para usuario final autenticado indicando al sistema el token correspondiente. 2. El sistema genera una *Validate Token Request* (utilizando los Validate Token Request Params) que es enviada al JWKS Endpoint del OP. 3. El sistema obtiene una *Validate Token* Response generada por el JWKS Endpoint del OP producto del pedido realizado. 4. El sistema valida la respuesta. 5. El sistema valida el token ingresado por la Aplicación móvil RP utilizando los *Token Response Params*. El resultado de la validación es retornado a la Aplicación móvil RP. 6. Fin CU. | |
| **Flujos Alternativos:** No hay | |
| **Poscondiciones**: No hay cambio en el estado. | |

### 2.3.8. Finalizar Sesión en OP para Usuario Final Autenticado

El RP solicita al componente SDK cerrar sesión del usuario final autenticado. El componente SDK realiza un *Logout Request* al OP utilizando los *Logout Request Params.* En caso de no existir errores, la sesión del usuario final se cerrará ante el OP y se devolverá un mensaje correspondiente al RP. El componente elimina los parámetros relacionados a la sesión de usuario previamente cerrada.

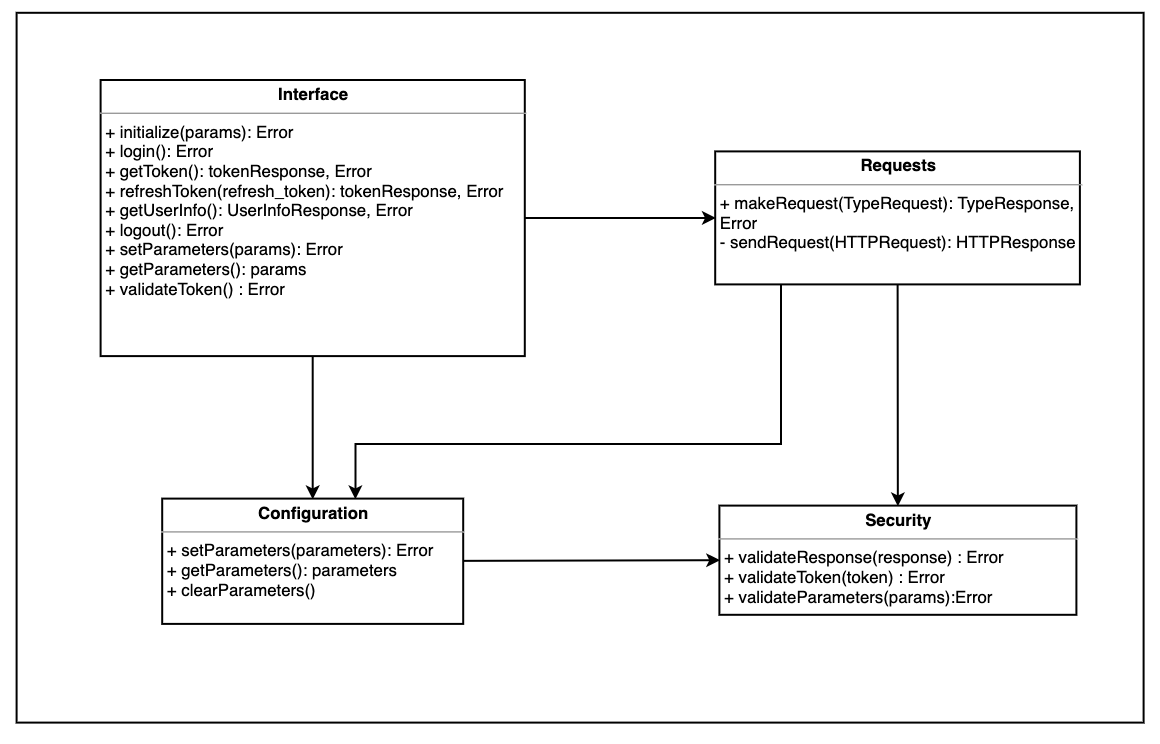
Este caso de uso se corresponde con el requerimiento R06.

|  |  |
| --- | --- |
| **Nombre**: | Finalizar sesión en OP para Usuario final autenticado |
| **Objetivo**: Permitir a la Aplicación móvil RP terminar la sesión de un Usuario final ante el OP. Permite al Usuario final comunicar al OP que su sesión en la aplicación móvil RP ha finalizado. | |
| **Actores**:  Aplicación móvil RP. | |
| **Precondiciones**:  El sistema se encuentra inicializado.  El sistema tiene un *token* almacenado. | |
| **Descripción**:  El caso de uso comienza cuando la Aplicación móvil RP desea cerrar la sesión de Usuario gub.uy de un Usuario final. El sistema establecerá una comunicación con el Logout Endpoint del OP para cerrar la sesión. La respuesta obtenida será validada por el componente. | |
| **Flujo normal**:   1. La Aplicación móvil RP desea cerrar la sesión de Usuario gub.uy de un Usuario final. 2. El sistema genera una Logout Request (utilizando los Authentication Request Params) y la envía al Authorization Endpoint del OP haciendo uso del navegador web del dispositivo móvil. 3. A través del navegador web del dispositivo móvil, el usuario final puede apreciar cómo su sesión es cerrada directamente en el OP. Una vez que finaliza su sesión en el OP, el usuario final es redirigido a la Aplicación móvil RP. 4. El sistema obtiene una Logout Response generada por el Authorization Endpoint del OP producto del cierre de la sesión del usuario final. 5. El sistema valida la respuesta y se indica a la Aplicación móvil RP la correctitud del cierre de la sesión del usuario final en el OP. 6. Fin CU. | |
| **Flujos Alternativos:** No hay. | |
| **Poscondiciones**:  El sistema elimina todo parámetro relacionado a la sesión de usuario final en la aplicación móvil RP. | |

# 3. Vista del Modelo de Diseño

## 3.1. Diseño de Clases

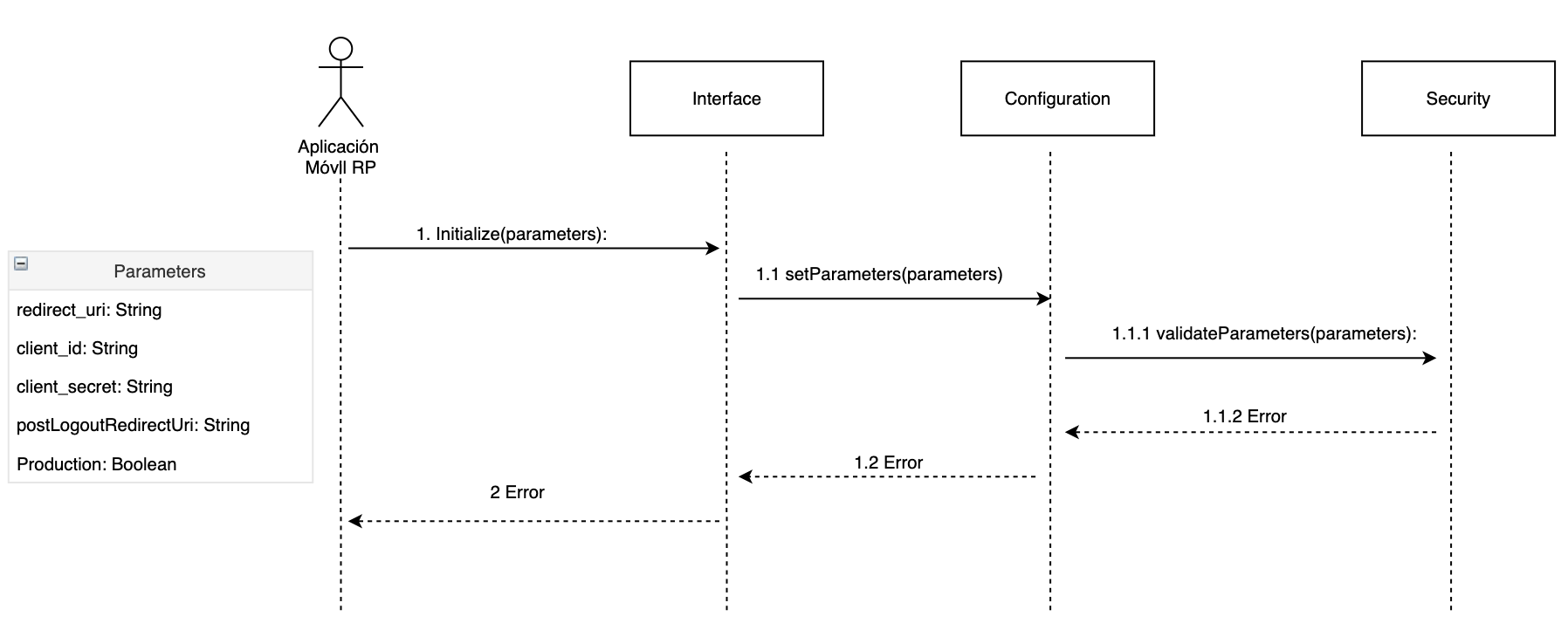
Se muestran las principales clases del sistema junto a las interfaces que implementan y las visibilidades. Los tipos de datos y clases más específicas se incluyen en los diagramas de diseño de los casos de uso (Sección 3.2). Se optó por este diseño debido a que favorece la definición de los módulos incrementando la mantenibilidad del componente. Este diseño también facilita la verificación de cada módulo y del sistema como un todo. Asimismo, modularizar el componente simplifica el trabajo de implementación y de integración. Los módulos junto a las clases que contienen, se encuentran descritos en la Sección 6.



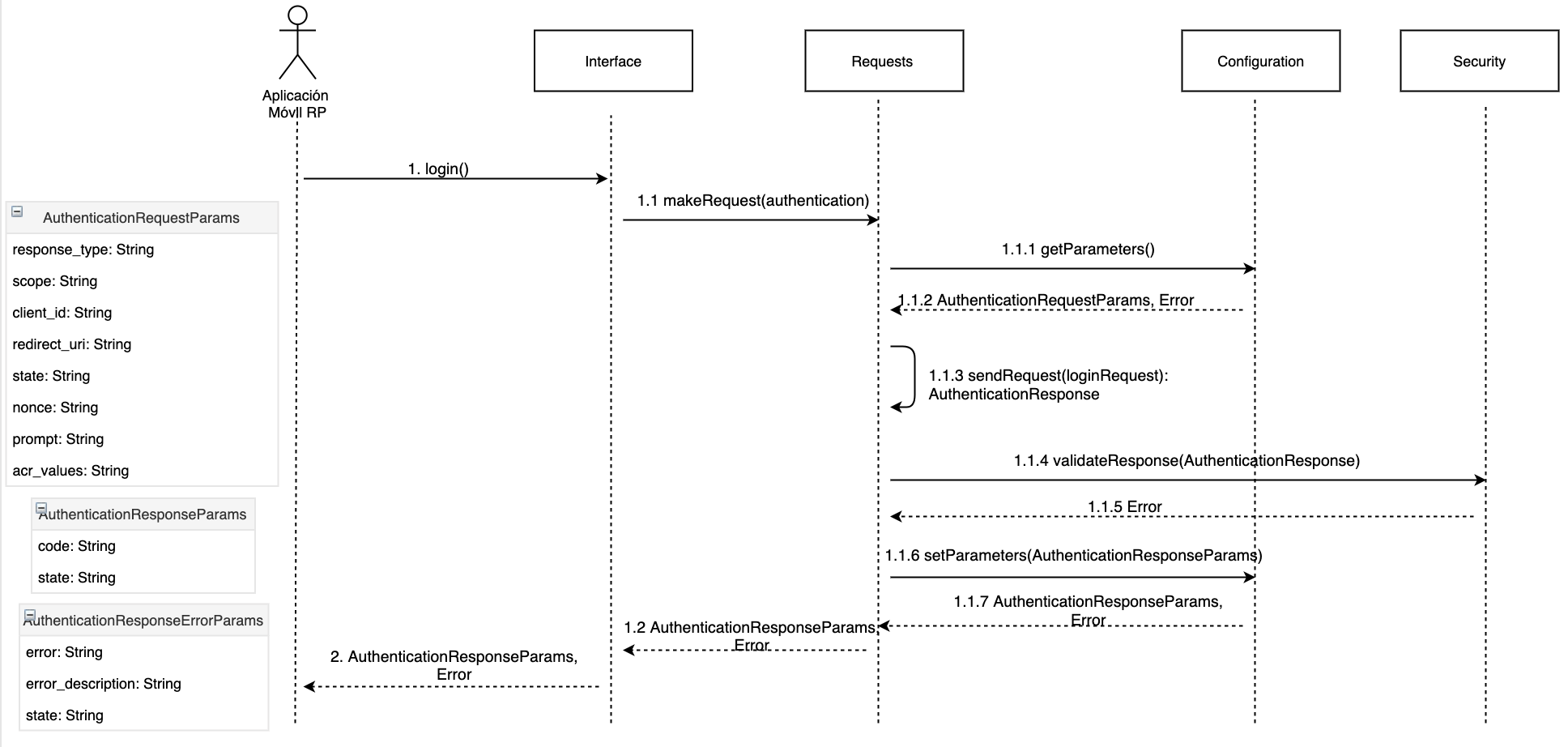
## 3.2. Diseño de Casos de Uso

En esta sección se presentan las interacciones e invocaciones entre entidades del componente para resolver los casos de uso. También se muestran los tipos de datos relevantes para cada uno.

### 3.2.1. Inicializar componente



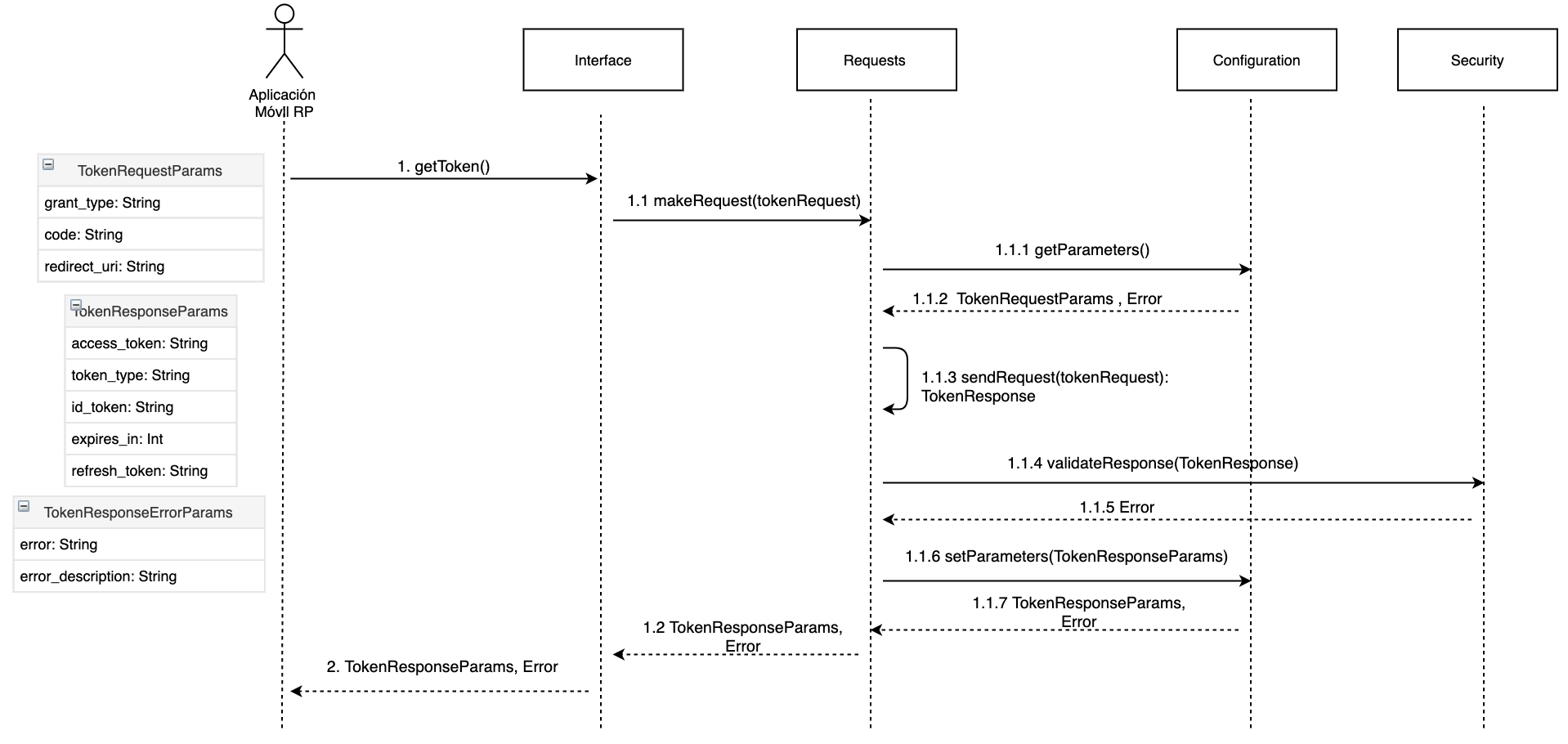
### 3.2.2. Autenticar usuario final



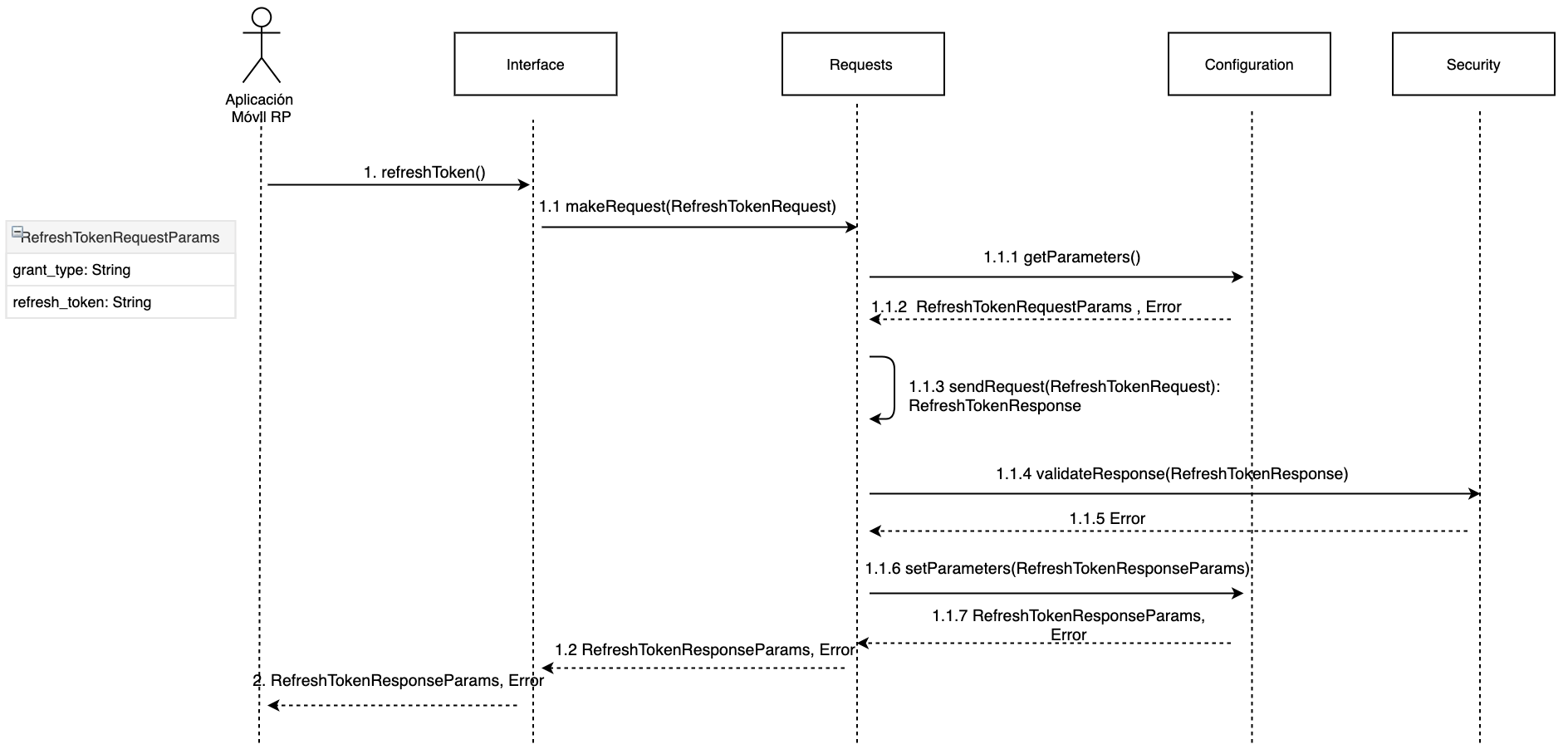
### 

### 

### 3.2.3. Obtener token para usuario final autenticado



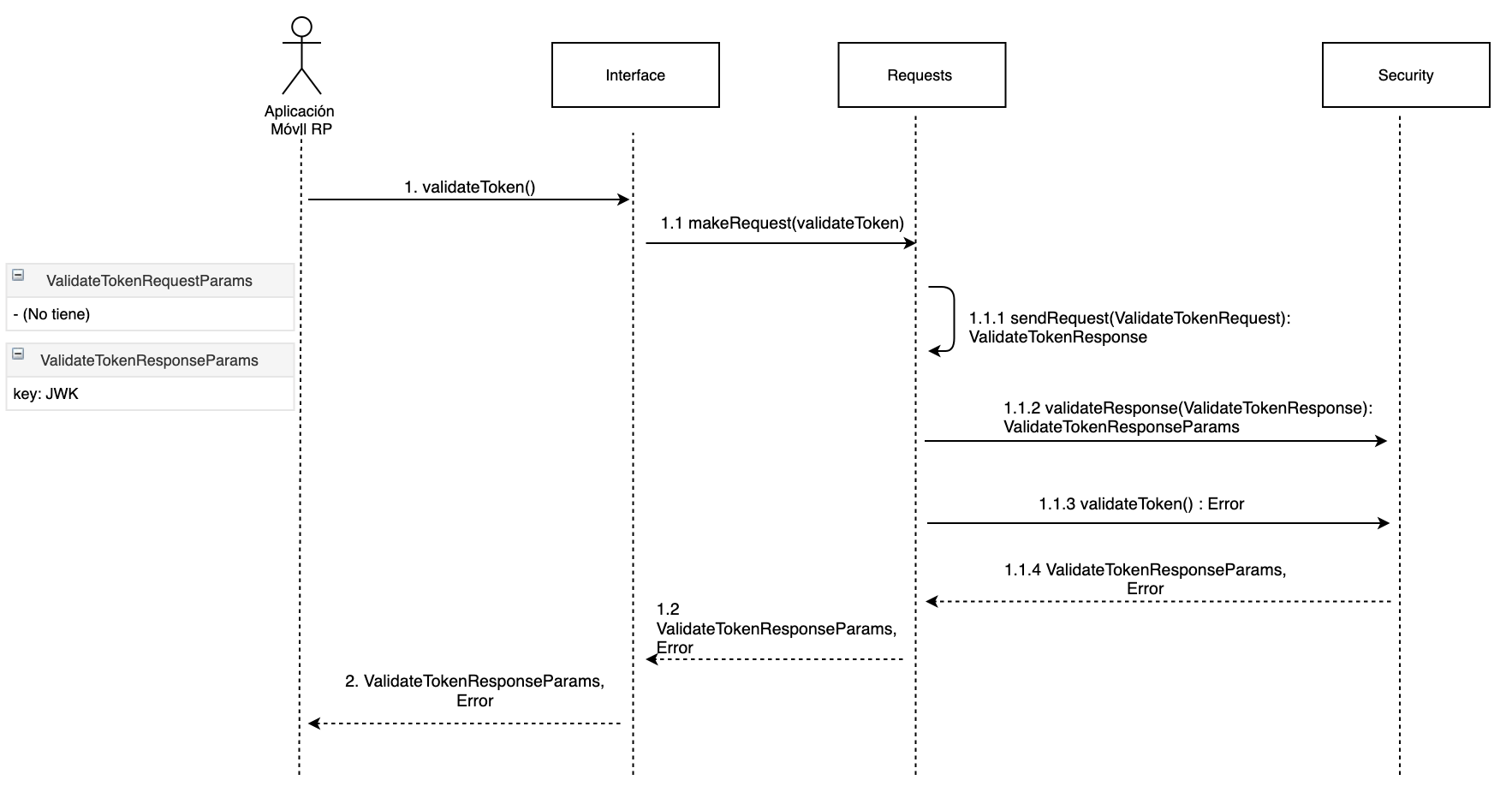
### 3.2.4. Refrescar token para usuario final autenticado



### 3.2.5. Obtener Información de Usuario Final Autenticado

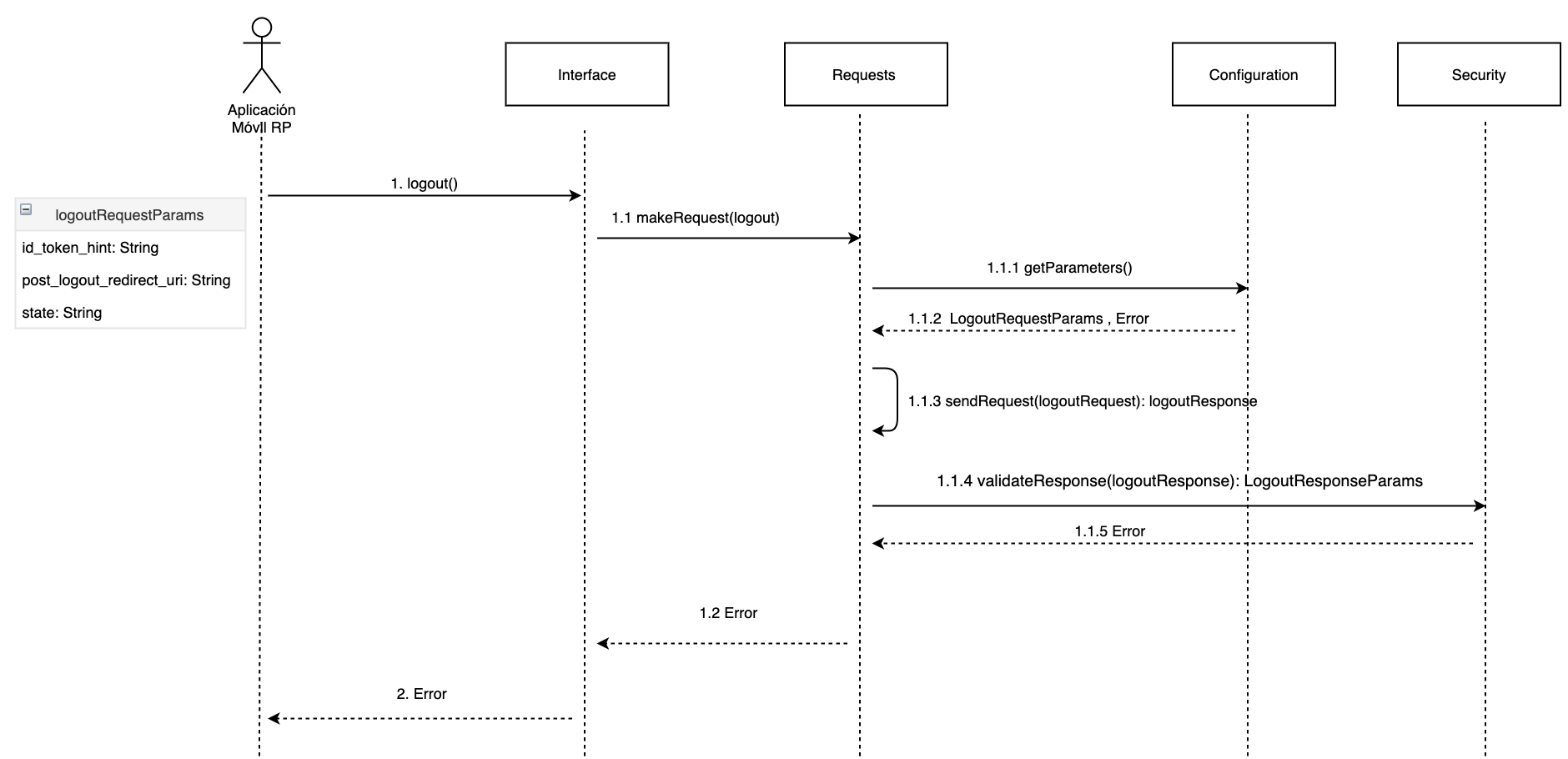
### 

### 3.2.6. Verificar Token Obtenido para Usuario Final Autenticado



### 

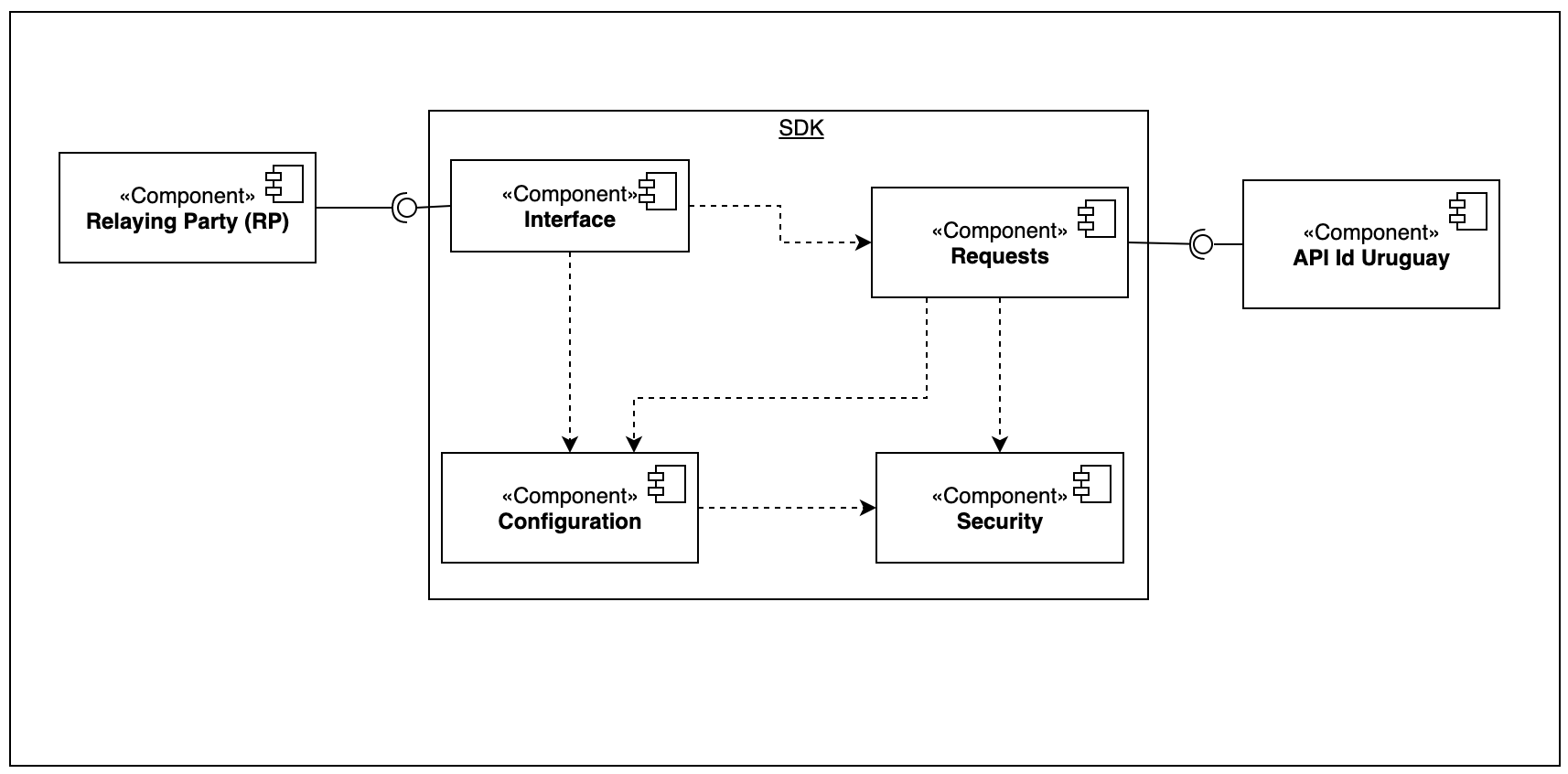
### 3.2.7. Finalizar Sesión en OP para Usuario Final Autenticado



# 4. Trazabilidad desde el Modelo de Diseño al Modelo de Implementación

Debido a que los componentes participantes de cada caso de uso (explicitados en la sección de Modelo de diseño) son los mismos componentes que se presentan en la sección Modelo de Implementación, no se cree oportuno realizar diagramas que establezcan la trazabilidad entre dichos modelos, pues la correspondencia de los componentes es directa.

# 5. Vista del Modelo de Implementación



## 5.1. Componentes

### 5.1.1. Interface

|  |
| --- |
|  |
| *Figura 1. Componente Interface* |

El componente *Interface* expone las funciones que estarán disponibles para la Aplicación móvil RP al utilizar el componente SDK. La Aplicación móvil RP solamente podrá hacer uso del componente SDK a través de las funciones provistas por este módulo. A su vez este módulo es encargado de realizar el procesamiento de errores y exponerlos a la Aplicación móvil RP. La firma de las funciones en alto nivel se encuentran en las interfaces definidas en la *figura 1*.

#### 5.1.1.1. Descripción de Procedimientos

* *initialize(params):Error* - Recibe los parámetros (utilizados para comunicarse con el OP) para inicializar el SDK. Estos parámetros son validados y seteados en el componente Configuration. Como respuesta se obtiene un tipo Error que indicará el resultado de la operación (por ejemplo código *gubuy\_no\_error* representa un resultado correcto, y otros valores representan resultados incorrectos). Luego de la invocación a esta función se inicializan todos los componentes del SDK y se establecen los valores por defecto en el componente *Configuration*. No debe ser posible invocar ninguna función de Interface sin antes haber invocado esta función.
* *login(): AuthenticationResponseParams, Error* - Devuelve un dato de tipo *AuthenticationResponseParams* y un *Error*. En caso de que *Error* contenga código *gubuy\_no\_error*, el *AuthenticationResponseParams* será válido. En caso contrario en *Error* se encuentra el código de error con su descripción y el *AuthenticationResponseParams* no será válido.
* *getToken(): TokenResponseParams, Error* - Devuelve un dato de tipo *TokenResponseParams* y *Error*. En caso de que *Error* contenga código *gubuy\_no\_error*, el *TokenResponseParams* será válido. En caso contrario en Error se encuentra el código de error con su descripción y el *TokenResponseParams* no será válido.
* *refreshToken(refresh\_token): TokenResponseParams, Error* - Devuelve un dato de tipo *TokenResponseParams* y *Error*. En caso de que *Error* contenga código *gubuy\_no\_error*, el *TokenResponseParams* será válido. En caso contrario en Error se encuentra el código de error con su descripción y el *TokenResponseParams* no será válido.
* *getUserInfo(): userInfoResponseParams, Error* - Devuelve un dato de tipo *userInfoResponseParams* y *Error*. En caso de que Error contenga código *gubuy\_no\_error*, el *userInfoResponseParams* será válido. En caso contrario en Error se encuentra el código de error con su descripción y el *userInfoResponseParams* no será válido.
* *logout(): Error* - Finaliza la sesión del usuario final autenticado en el OP. Devuelve un dato de tipo *Error* y en caso que sea código *gubuy\_no\_error* la operación habrá sido exitosa, en caso contrario se tendrá la descripción correspondiente al error ocurrido.
* *setParameters(parameters) : Error* - Recibe los parámetros en un tipo Params para ser establecidos en el componente *Configuration*. Se actualizan los parámetros siempre y cuando todos los parámetros proporcionados sean válidos.
* *getParameters() : Parameters* - Se retornan todos los parámetros que se encuentran configurados en el SDK.

### 5.1.2. Configuration

|  |
| --- |
|  |
| *Figura 2. Componente Configuration* |

Para facilitar el uso del componente SDK por parte de la Aplicación móvil RP, se decidió que el SDK mantenga algunos parámetros repetitivos en su memoria. Además se mantienen las constantes utilizadas por todos los componentes del SDK y las URL de los endpoints de la API Id Uruguay. Esto evita que la Aplicación móvil RP deba proporcionar al SDK parámetros que son evidentes, simplificando el uso del mismo. El componente *Configuration* es el encargado de mantener los parámetros del SDK, el mismo posee una interfaz en la cual se puede actualizarlos u obtenerlos. Las funciones expuestas por este componente se encuentran definidas en la *figura 2*.

#### 5.1.2.1. Descripción de procedimientos

* *setParameters(parameters): Error* - Recibe los parámetros para ser establecidos en el componente. En caso que al menos un parámetro no sea válido, no se actualiza ningún parámetro en el componente y se devuelve un *Error*, en caso contrario se devuelve un *Error* con código *gubuy\_no\_error*, lo cual indica que se actualizaron los parámetros.
* *getParameters(): Params* - Devuelve los parámetros que se encuentren seteados en el componente.
* *clearParameters():* - Se eliminan los valores de todos los parámetros del SDK excepto client\_id, client\_secret, redirect\_uri y post\_logout\_redirect\_uri.

### 5.1.3. Requests

|  |
| --- |
|  |
| *Figura 3. Componente Requests* |

El componente SDK deberá comunicarse con la API REST de ID Uruguay mediante pedidos HTTP. El componente *Requests* es el encargado de realizar estos pedidos a la API y obtener las respuestas. También provee una interfaz por la cual se puede solicitar al componente que realice una solicitud a la API y también es posible indicarle que tipo de solicitud es la requerida. Los distintos tipos de solicitudes se encuentran representados en la clase *TypeRequest*, en la *figura 3*. Adicionalmente en el componente Requests, se realizará un pre-procesamiento de las respuestas obtenidas de la API. Las funciones expuestas por este componente se encuentran definidas en la *figura 3*.

#### 5.1.3.1. Descripción de procedimientos

* *makeRequest(TypeRequest): TypeResponse, Error* - Realiza una request al OP según el parámetro *TypeRequest*. La respuesta es validada por el componente y en caso de éxito, se retorna los parámetros correspondientes en un tipo *TypeResponse*. Los parámetros contenidos dentro del *TypeResponse*, se corresponden directamente con los parámetros del *TypeRequest*. En caso de ocurrir un error, se devuelve el código y descripción en un tipo *Error*.
* sendRequest(HTTPRequest): HTTPResponse - Realiza un http request y recibe el http response correspondiente. Es llamado desde la función makeRequest. Con el fin de agregar robustez al componente, se realizan hasta 5 reintentos por cada llamada a la API.

### 5.1.4. Security

|  |
| --- |
|  |
| *Figura 4. Componente Security* |

El componente *Security* es responsable de la validación de tokens intercambiados en las solicitudes realizadas por parte del SDK. La validación de los parámetros implica chequear que no están vacíos y siempre que sea posible verificar su validez (con en el caso del *id\_token*). La validación de las responses, implica chequear que sus parámetros coincidan con los especificados en la API de Id Uruguay. Por ejemplo, en caso que se utilice el parámetro *State* en una *request*, verificar que la response contenga el mismo valor. En el caso particular del *userInfoResponse* debe verificarse que el valor del campo *sub* coincida con el valor del campo *sub* del *id\_token* asociado al *access\_token* utilizado para realizar el *userInfoRequest*.

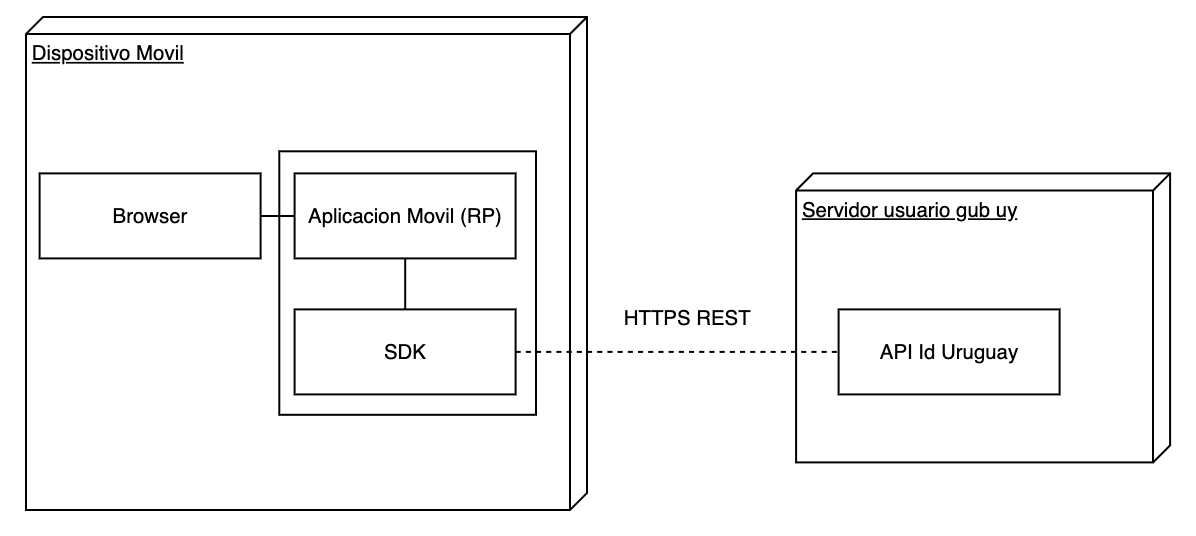
Se optó por un componente por separado debido a la sencillez de agregar controles de seguridad en un futuro en caso de que se requieran. Las funciones expuestas por este componente se encuentran definidas en la *figura 4*.

#### 5.1.4.1. Descripción de procedimientos

* *validateResponse(response): Error* - Se encarga de validar la respuesta obtenida desde el OP ante una request realizada por el componente Requests. En caso de éxito devolverá un tipo *Error* con código *gubuy\_no\_error*, en caso contrario se devolverá un código y descripción acorde al error ocurrido.
* *validateToken(token) : Error* - Se encarga de validar el token pasado como parámetro utilizando la información de validación obtenida del OP. En caso de éxito devolverá un tipo Error con código *gubuy\_no\_error*, en caso contrario se devolverá un código y descripción acorde al error ocurrido.
* *validateParams(params) : Error* - Se encarga de validar los parámetros. En caso de éxito devolverá un tipo Error con código *gubuy\_no\_error*, en caso contrario se devolverá un código y descripción acorde al error ocurrido.

# 6. Vista del Modelo de Distribución

## 6.1. Diagrama de Distribución



## 6.2. Nodos

### 6.2.1. Dispositivo Móvil

El dispositivo móvil donde se ejecuta la aplicación integrada con el componente SDK. Este nodo debe contar con una instalación de un navegador web.

### 6.2.2. Servidor Usuario gub.uy

Servidor de autenticación y autorización con el cual la RP se comunica, por medio del SDK, para acceder a información del usuario final. Dicho servidor implementa los siguientes endpoints: A*uthentication, Token, Userinfo, JWKS y Logout.*

## 6.3. Conexiones

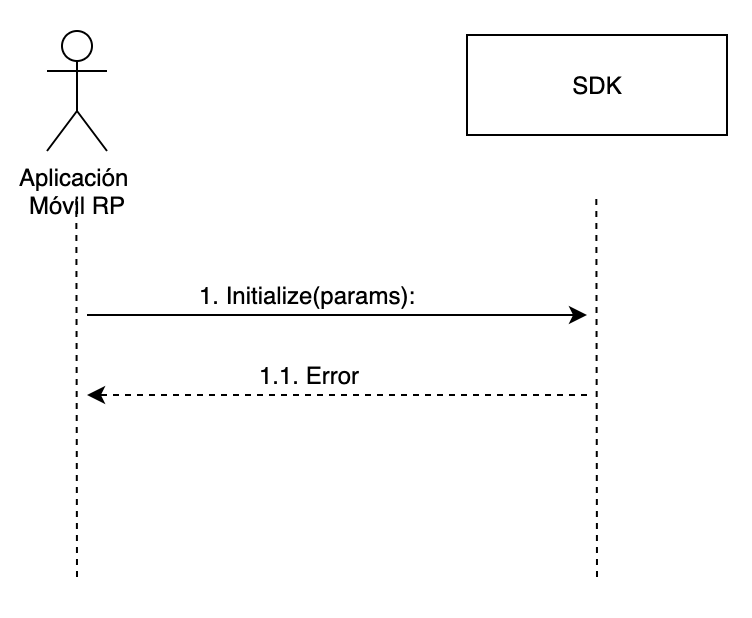
### 6.3.1. HTTPS

Para establecer las comunicaciones entre el dispositivo móvil y el servidor de autenticación y autorización se utilizan llamadas a la API REST de ID Uruguay utilizando el protocolo HTTPS.

# 7. Vista del Modelo de Procesos

En esta sección se presenta una vista de Procesos de los casos de uso definidos en la sección 2.2. El objetivo de esta sección es lograr un mejor entendimiento del usuario sobre la interacción del componente SDK con los componentes externos y sus invocaciones.

## 7.1. Inicializar componente



## 7.2. Autenticar usuario final

|  |
| --- |
|  |
| *Figura 5. Vista de proceso para Autenticar usuario final* |

## 7.3. Obtener Token para Usuario Final Autenticado

|  |
| --- |
|  |
| *Figura 6. Vista de proceso para Obtener Token para Usuario Final Autenticado* |

## 7.4. Obtener Información de Usuario Final Autenticado

|  |
| --- |
|  |
| *Figura 7. Vista de proceso para Obtener Información de Usuario Final Autenticado* |

## 7.5. Verificar Token Obtenido para Usuario Final Autenticado

|  |
| --- |
|  |
| *Figura 8. Vista de proceso para Verificar Token Obtenido para Usuario Final Autenticado* |

### 

## 7.6. Finalizar Sesión en OP para Usuario Final Autenticado

|  |
| --- |
|  |
| *Figura 9. Vista de proceso para Finalizar Sesión en OP para Usuario Final Autenticado* |

# 8. Restricciones del diseño respecto al lenguaje Javascript

## 8.1 Restricciones generales del paradigma basado en prototipos

Si bien la arquitectura planteada considera conceptos generales de programación orientada a objetos, con el fin de poder ser implementable en distintos lenguajes, se deben tener en consideración algunas restricciones de diseño impuestas por el uso de Javascript y React Native. Javascript es un lenguaje con orientación a objetos basada en prototipos (*prototype-based*)[[1]](#footnote-0), con lo que difiere de los lenguajes que siguen el paradigma de orientación a objetos tradicional. No se cuenta con un artefacto de clase (existe la palabra reservada *class* en la especificación ES6, pero esta es simplemente una función “especial”[[2]](#footnote-1)) ni de interfaz, con lo que la arquitectura se debe modificar levemente. La estructura definida es la misma, y se definen módulos (carpetas) que corresponden a los distintos componentes definidos (*Interfaces, Requests, Configuration,* y *Security*) en lugar de clases o interfaces. Se sigue un estilo de programación más acorde al paradigma funcional, donde las funcionalidades de cada componente se implementan como funciones expuestas por un módulo en lugar de métodos de una clase. Esto es consistente con el enfoque declarativo de React y con el flujo de programación en Javascript.

## 8.2 Programación asíncrona

Otro aspecto a considerar del lenguaje de implementación es la programación asíncrona. La función de *fetch* de Javascript, utilizada para hacer pedidos HTTP, es asíncrona[[3]](#footnote-2). Esto quiere decir que devuelve una *Promise*, que se resolverá si el pedido es exitoso o rechazará en caso de error. Cualquier función que llame a *fetch t*ambién devolverá una *Promise*, siendo por ende una función asíncrona. Como las funciones expuestas por el SDK utilizan *fetch* para realizar pedidos, estas deberán ser funciones asíncronas. Esto no agrega demasiada complejidad para un desarrollador de Javascript, ya que las funciones asíncronas se pueden llamar casi igual que una función normal, pero utilizando *async/await* o *Promise.then()[[4]](#footnote-3)*.

1. <https://developer.mozilla.org/en-US/docs/Web/JavaScript> [↑](#footnote-ref-0)
2. [https://developer.mozilla.rg/en-US/odocs/Web/JavaScript/Reference/Classes](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes) [↑](#footnote-ref-1)
3. <https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API> [↑](#footnote-ref-2)
4. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function> [↑](#footnote-ref-3)