1. Requisitos del proyecto

Partiendo de una serie de items necesarios para llevar a cabo el proyecto de forma satisfactoria, a continuación se expondrán un conjunto de escenarios y pruebas de aceptación para cada uno de los requisitos que se plantean en la hoja de ruta de este proyecto.

En primer lugar, ahondaremos en los requisitos imprescindibles para llevar a cabo el proyecto en su forma básica. Para finalizar, hemos añadido una serie de requisitos adicionales (avanzados) que nuestro proyecto también va a tener en cuenta.

De forma adicional, también se muestran para cada una de las historias y subhustorias de usuario una serie de tests de integración y aceptación

1.1. Requisitos básicos

1.1.1. Requisito básico 1 - Gestionar una ubicación de interés

El primer requisito básico consta de gestionar una ubicación de interés sobre la que se desea consultar información. En esencia de ello depende el fundamento básico de este aplicativo web.

1.1.1.1 Requisito básico 1, Historia de usuario 1

Como usuario quiero dar de alta una ubicación a partir de un topónimo, con el fin de tenerla disponible en el sistema.

Escenarios

Escenario	Ubicaciones previas	Topónimo válido	Ubicación repetida	Ubicaciones después	BBDD modificada
E1	0	Si	No	1	Si
E2	0	No	No	0	No
E3	1	Si	Si	1	No
E4	1	No	Si	1	No

Tabla 1: Escenarios Requisito básico 1, Historia de Usuario 1

Análisis

El escenario E1 podría subdividirse en al menos tres pruebas de aceptación diferentes. Una usando topónimos válidos pero poco conocidos (Antártida), otra con topónimos válidos con una única representación (Madrid) y otra con topónimos válidos con múltiples representaciones (Castelló y Castellón).

El escenario E2 podría subdividirse en al menos tres pruebas de aceptación diferentes. Una usando un topónimo invlido formado por símbolos y otros caracteres especiales, otra con un topónimo invlido formado por caracteres normales y otra con un topónimo invalido formado por una pequeña variación de uno valido.

Pruebas de aceptación

	E1 (válido)	**E2 (inválido)**
Given	▪ No hay ubicaciones disponibles ▪ Un usuario registrado ▪ Un topónimo 'Castellon'	▪ No hay ubicaciones registradas ▪ Un usuario registrado ▪ Un topónimo 'INVALIDO'
When	Añadir una ubicación 'Castellón'	Añadir una ubicación 'INVALIDO'
Then	Hay una ubicación registrada 'Castellon'	No hay ubicaciones registradas

Tabla 2: Pruebas de aceptación, Requisito básico 1, Historia de Usuario 1

Tests de integración y aceptación

Los siguientes tests están en el repositorio `app` del proyecto y se encuentran en las rutas:

`src/test/java/app/test/acceptance/basic/requirement01/History01.java`

`src/test/java/app/test/integration/basic/requirement01/History01.java`

Fragmento de código 1: Test de aceptación del requisito basico 1 → HU01

```java
public class History01 extends SessionTest {
    @Test
    public void valid() {
        // Given
        var name = "Castellon de la Plana";
        Mockito.reset(spy.accountManager);

        // When
        var response = client.location.addLocation(name);

        // Then
        Mockito.verify(spy.accountManager).saveAccount(any());
        response.statusCode(HttpStatus.OK.value());
        var state = client.location.getLocations();
        state.body("", hasSize(1));
        state.body("name", hasItem(name));
    }

    @Test
    public void invalid() {
        // Given
        var name = "INVALIDO";
        Mockito.reset(spy.accountManager);

        // When
        var response = client.location.addLocation(name);

        // Then
        Mockito.verify(spy.accountManager, never()).saveAccount(any());
```

```
45          response.statusCode(HttpStatus.NOT_FOUND.value());
46          var state = client.location.getLocations();
47          state.body("", hasSize(0));
48      }
49  }
```

Fragmento de código 2: Test de integración del requisito basico 1 → HU01

```
18  public class History01 extends SessionTest {
19      @Test
20      public void valid() {
21          // Given
22          var name = "Castellon";
23          var locationMock = new LocationModel(name, 39.980, -0.033);
24          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
25          Mockito.reset(spy.accountManager);
26
27          // When
28          var response = client.location.addLocation(name);
29
30          // Then
31          Mockito.verify(spy.accountManager).saveAccount(any());
32          response.statusCode(HttpStatus.OK.value());
33          var state = client.location.getLocations();
34          state.body("", hasSize(1));
35          state.body("name", hasItem(name));
36      }
37
38      @Test
39      public void invalid() {
40          // Given
41          var name = "INVALIDO";
42          Mockito.doThrow(new MissingError()).when(spy.queryManager).getData(name
                );
43          Mockito.reset(spy.accountManager);
44
45          // When
46          var response = client.location.addLocation(name);
47
48          // Then
49          Mockito.verify(spy.accountManager, never()).saveAccount(any());
50          response.statusCode(HttpStatus.NOT_FOUND.value());
51          var state = client.location.getLocations();
52          state.body("", hasSize(0));
53      }
54  }
```

1.1.1.2 Requisito básico 1, Historia de usuario 2

Como usuario quiero dar de alta una ubicación a partir de unas coordenadas geográficas, con el fin de tenerla disponible en el sistema.

Escenarios

Escenario	Ubicaciones previas	Coord. válidas	Ubicación repetida	Ubicaciones después	BBDD modificada
E1	0	Si	No	1	Si
E2	0	No	No	0	No
E3	1	Si	Si	1	No
E4	1	No	Si	1	No

Tabla 3: Escenarios Requisito básico 1, Historia de Usuario 2

Análisis

El escenario E1 podría subdividirse en al menos tres pruebas de aceptación diferentes. Una usando unas coordenadas válidas pero en el límite sintáctico (90.0, 180.0), otra con unas coordenadas válidas en la frontera de dos ciudades (Castellón y Benicassim) y otra otra unas coordenadas válidas una ubicación concreta (Castellon).

El escenario E2 podría subdividirse en al menos tres pruebas de aceptación diferentes. Una usando unas coordenadas inválidas formadas por símbolos y otros caracteres especiales, otra con unas coordenadas inválidas lejos del límite sintáctico (180.0, 360.0) y otra con unas coordenadas inválidas cerca del límite sintáctico (90.1, 180.1).

Pruebas de aceptación

	E1 (válido)	E2 (inválido)
Given	▪ Un usuario registrado ▪ No hay ubicaciones registradas ▪ Unas coordenadas '39,0'	▪ Un usuario registrado ▪ No hay ubicaciones registradas ▪ Unas coordenadas '180,360'
When	Añadir unas coordenadas '39,0'	Añadir unas coordenadas '180,360'
Then	Hay una ubicación registrada con las coordenadas '39,0'	No hay ubicaciones registradas

Tabla 4: Pruebas de aceptación, Requisito básico 1, Historia de Usuario 2

Tests de integración y aceptación

Los siguientes tests están en el repositorio `app` del proyecto y se encuentran en las rutas:

`src/test/java/app/test/acceptance/basic/requirement01/History02.java`

`src/test/java/app/test/integration/basic/requirement01/History02.java`

Fragmento de código 3: Test de aceptación del requisito basico 1 → HU02

```
public class History02 extends SessionTest {
    @Test
    public void valid() {
        // Given
        var coords = "39.978,-0.033";
        Mockito.reset(spy.accountManager);

        // When
        var response = client.location.addLocation(coords);

        // Then
        Mockito.verify(spy.accountManager).saveAccount(any());
        response.statusCode(HttpStatus.OK.value());
        var state = client.location.getLocations();
        state.body("", hasSize(1));
        state.body("coords", hasItem(coords));
    }

    @Test
    public void invalid() {
        // Given
        var coords = "180.0,360.0";
        Mockito.reset(spy.accountManager);

        // When
        var response = client.location.addLocation(coords);

        // Then
        Mockito.verify(spy.accountManager, never()).saveAccount(any());
        response.statusCode(HttpStatus.NOT_FOUND.value());
        var state = client.location.getLocations();
        state.body("", hasSize(0));
    }
}
```

Fragmento de código 4: Test de integración del requisito basico 1 → HU02

```
public class History02 extends SessionTest {
    @Test
    public void valid() {
        // Given
        var name = "NAME";
        var locationMock = new LocationModel(name, 39.980, -0.033);
        var coords = locationMock.getCoords();
        Mockito.doReturn(locationMock).when(spy.queryManager).getData(coords);
        Mockito.reset(spy.accountManager);

        // When
        var response = client.location.addLocation(coords);

        // Then
        Mockito.verify(spy.accountManager).saveAccount(any());
        response.statusCode(HttpStatus.OK.value());
        var state = client.location.getLocations();
        state.body("", hasSize(1));
```

```
36            state.body("coords", hasItem(coords));
37        }
38
39        @Test
40        public void invalid() {
41            // Given
42            var coords = "180.0,360.0";
43            Mockito.doThrow(new MissingError()).when(spy.queryManager).getData(
                  coords);
44            Mockito.reset(spy.accountManager);
45
46            // When
47            var response = client.location.addLocation(coords);
48
49            // Then
50            Mockito.verify(spy.accountManager, never()).saveAccount(any());
51            response.statusCode(HttpStatus.NOT_FOUND.value());
52            var state = client.location.getLocations();
53            state.body("", hasSize(0));
54        }
55    }
```

1.1.1.3 Requisito básico 1, Historia de usuario 3

Como usuario quiero validar el topónimo de una ubicación disponible en los servicios API activos, con el fin de evaluar su utilidad.

Escenarios

Escenario	Ubicaciones activas	Topónimo válido	Servicio disponible	Resultado
E1	1	Si	Si	Si
E2	0	Si	No	No
E3	0	No	Si	No
E4	0	No	No	No

Tabla 5: Escenarios Requisito básico 1, Historia de Usuario 3

Análisis

El escenario E1 podría subdividirse en al menos tres pruebas de aceptación diferentes. Una usando topónimos válidos disponibles pero poco conocidos (Antártida), otra con topónimos válidos disponibles conocidos (Castellón), y otra con topónimos válidos disponible que ya esté registrado.

El escenario E3 podría subdividirse en al menos tres pruebas de aceptación diferentes. Una usando un topónimo invlido formado por símbolos y otros caracteres especiales, otra con un topónimo invlido formado por caracteres normales y otra con un topónimo invalido formado por una pequeña variación de uno valido.

Pruebas de aceptación

	E1 (válido)	**E3 (inválido)**
Given	▪ Un usuario registrado ▪ La ubic. 'Castellón' activa ▪ El servicio del clima ▪ El topónimo 'Valencia'	▪ Un usuario registrado ▪ Ninguna ubicación activa ▪ El topónimo 'INVALIDO'
When	Se realiza una petición a la API para el topónimo 'Valencia'	Se realiza una petición a la API para ese topónimo
Then	La API devuelve información del clima para Valencia	La API devuelve un error

Tabla 6: Pruebas de aceptación, Requisito básico 1, Historia de Usuario 3

Tests de integración y aceptación

Los siguientes tests están en el repositorio `app` del proyecto y se encuentran en las rutas:

`src/test/java/app/test/acceptance/basic/requirement01/History03.java`

`src/test/java/app/test/integration/basic/requirement01/History03.java`

Fragmento de código 5: Test de aceptación del requisito basico 1 → HU03

```
14  public class History03 extends SessionTest {
15      @Test
16      public void valid() {
17          // Given
18          var name = "Castellon";
19          var type = ServiceType.WEATHER.name();
20          client.location.addLocation(name);
21          client.service.enableService(type);
22          name = "Valencia";
23
24          // When
25          var response = client.service.getServicesForLocation(name);
26
27          // Then
28          response.statusCode(HttpStatus.OK.value());
29          response.body(setupEnabledQuery(true, ""), hasSize(1));
30          response.body(setupEnabledQuery(true, "service.type"), hasItem(type));
31      }
32
33      @Test
34      public void invalid() {
35          // Given
36          var name = "INVALIDO";
37
38          // When
39          var response = client.service.getServicesForLocation(name);
40
```

```
41            // Then
42            response.statusCode(HttpStatus.NOT_FOUND.value());
43        }
44    }
```

Fragmento de código 6: Test de integración del requisito basico 1 → HU03

```
17  public class History03 extends SessionTest {
18      @Test
19      public void valid() {
20          // Given
21          var name = "Castellon";
22          var type = ServiceType.WEATHER.name();
23          var locationMock = new LocationModel(name, 39.980, -0.033);
24          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
25          client.location.addLocation(name);
26          client.service.enableService(type);
27
28          name = "Valencia";
29          locationMock = new LocationModel(name, 39.503, -0.405);
30          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
31          Mockito.doReturn(true).when(spy.weatherService).getData(locationMock);
32
33          // When
34          var response = client.service.getServicesForLocation(name);
35
36          // Then
37          response.statusCode(HttpStatus.OK.value());
38          response.body(setupEnabledQuery(true, ""), hasSize(1));
39          response.body(setupEnabledQuery(true, "service.type"), hasItem(type));
40          response.body(setupEnabledQuery(true, "data"), hasItem(true));
41      }
42
43      @Test
44      public void invalid() {
45          // Given
46          var name = "INVALIDO";
47          Mockito.doThrow(new MissingError()).when(spy.queryManager).getData(name
                );
48
49          // When
50          var response = client.service.getServicesForLocation(name);
51
52          // Then
53          response.statusCode(HttpStatus.NOT_FOUND.value());
54      }
55  }
```

1.1.1.4 Requisito básico 1, Historia de usuario 4

Como usuario quiero validar las coordenadas geográficas de una ubicación disponible en los servicios API activos, con el fin de evaluar su utilidad.

Escenarios

Escenario	Ubicaciones activas	Coordenadas válidas	Servicio disponible	Resultado
E1	1	Si	Si	Si
E2	0	Si	No	No
E3	0	No	Si	No
E4	0	No	No	No

Tabla 7: Escenarios Requisito básico 1, Historia de Usuario 4

Análisis

El escenario E1 podría subdividirse en al menos tres pruebas de aceptación diferentes. Una usando unas coordenadas válidas pero en el límite sintáctico (90.0, 180.0), otra con unas coordenadas válidas en la frontera de dos ciudades (Castellón y Benicassim) y otra otra unas coordenadas válidas una ubicación concreta (Castellon).

El escenario E3 podría subdividirse en al menos tres pruebas de aceptación diferentes. Una usando unas coordenadas inválidas formadas por símbolos y otros caracteres especiales, otra con unas coordenadas inválidas lejos del límite sintáctico (180.0, 360.0) y otra con unas coordenadas inválidas cerca del límite sintáctico (90.1, 180.1).

Pruebas de aceptación

	E1 (válido)	**E3 (inválido)**
Given	▪ Las coordenadas '39,0' ▪ EL servicio del clima ▪ Un usuario registrado ▪ La ubicación 'Valencia' activa	▪ Las coordenadas '180,360' ▪ El servicio del clima ▪ Un usuario registrado
When	Se realiza una petición a la API para esas coordenadas	Se realiza una petición a la API para para esas coordenadas
Then	La API devuelve información del clima para las coordenadas '39,0'	La API devuelve un error

Tabla 8: Pruebas de aceptación, Requisito básico 1, Historia de Usuario 4

Tests de integración y aceptación

Los siguientes tests están en el repositorio app del proyecto y se encuentran en las rutas:

- src/test/java/app/test/acceptance/basic/requirement01/History04.java
- src/test/java/app/test/integration/basic/requirement01/History04.java

Fragmento de código 7: Test de aceptación del requisito basico 1 → HU04

```java
public class History04 extends SessionTest {
    @Test
    public void valid() {
        // Given
        var name = "Valencia";
        var coords = "39.978,-0.033";
        var type = ServiceType.WEATHER.name();
        client.location.addLocation(name);
        client.service.enableService(type);

        // When
        var response = client.service.getServicesForLocation(coords);

        // Then
        response.statusCode(HttpStatus.OK.value());
        response.body(setupEnabledQuery(true, ""), hasSize(1));
        response.body(setupEnabledQuery(true, "service.type"), hasItem(type));
    }

    @Test
    public void invalid() {
        // Given
        var coords = "180,360";

        // When
        var response = client.service.getServicesForLocation(coords);

        // Then
        response.statusCode(HttpStatus.NOT_FOUND.value());
    }
}
```

Fragmento de código 8: Test de integración del requisito basico 1 → HU04

```java
public class History04 extends SessionTest {
    @Test
    public void valid() {
        // Given
        var name = "Valencia";
        var locationMock = new LocationModel(name, 39.503, -0.405);
        Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
        client.location.addLocation(name);

        name = "NAME";
        locationMock = new LocationModel(name, 39.980, -0.033);
```

```
28          var coords = locationMock.getCoords();
29          Mockito.doReturn(locationMock).when(spy.queryManager).getData(coords);
30          Mockito.doReturn(true).when(spy.weatherService).getData(locationMock);
31
32          var type = ServiceType.WEATHER.name();
33          client.service.enableService(type);
34
35          // When
36          var response = client.service.getServicesForLocation(coords);
37
38          // Then
39          response.statusCode(HttpStatus.OK.value());
40          response.body(setupEnabledQuery(true, ""), hasSize(1));
41          response.body(setupEnabledQuery(true, "service.type"), hasItem(type));
42      }
43
44      @Test
45      public void invalid() {
46          // Given
47          var coords = "180,360";
48          Mockito.doThrow(new MissingError()).when(spy.queryManager).getData(
                coords);
49
50          // When
51          var response = client.service.getServicesForLocation(coords);
52
53          // Then
54          response.statusCode(HttpStatus.NOT_FOUND.value());
55      }
56  }
```

1.1.1.5 Requisito básico 1, Historia de usuario 5

Como usuario quiero activar una ubicación disponible en el sistema, con el fin de recibir información relacionada con dicha ubicación

Escenarios

Escenario	Ubicaciones activas previas	Ubicaciones desactivadas previas	Ubicaciones activas después	Ubicaciones desactivadas después	BBDD modificada
E1	0	1	1	0	Si
E2	1	0	1	0	No

Tabla 9: Escenarios Requisito básico 1, Historia de Usuario 5

Análisis

El escenario E1 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería activar una ubicación que ya esté en el sistema pero estuviera desactivado.

El escenario E2 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería activar una ubicación que ya esté en el sistema pero estuviera activado.

Pruebas de aceptación

	E1 (válido)	**E2 (inválido)**
Given	▪ Un usuario registrado ▪ Ninguna ubicación activa ▪ Una ubicación desactivada en el sistema 'Castellón'	▪ Un usuario registrado ▪ Una ubicación activada en el sistema 'Castellón' ▪ Ninguna ubica. desactivada
When	Se realiza la petición sobre 'Castellón'	Se realiza la petición sobre 'Castellón'
Then	▪ Una ubicación activada en el sistema 'Castellón' ▪ Ninguna ubic. desactivada	▪ Una ubicación activada en el sistema 'Castellón' ▪ Ninguna ubica. desactivada

Tabla 10: Pruebas de aceptación, Requisito básico 1, Historia de Usuario 5

Tests de integración y aceptación

Los siguientes tests están en el repositorio `app` del proyecto y se encuentran en las rutas:

`src/test/java/app/test/acceptance/basic/requirement01/History05.java`

`src/test/java/app/test/integration/basic/requirement01/History05.java`

Fragmento de código 9: Test de aceptación del requisito basico 1 → HU05

```
17 public class History05 extends SessionTest {
18     @Test
19     public void valid() {
20         // Given
21         var name = "Castellon de la Plana";
22         var location = client.location.addLocation(name);
23         var coords = location.extract().jsonPath().getString("coords");
24         client.location.removeLocation(coords);
25         Mockito.reset(spy.accountManager);
26
27         // When
28         var response = client.history.restoreLocation(coords);
29
30         // Then
31         Mockito.verify(spy.accountManager).saveAccount(any());
32         response.statusCode(HttpStatus.OK.value());
33         var statePlaces = client.location.getLocations();
34         var stateHistory = client.history.getLocations();
35         statePlaces.body("", hasSize(1));
```

```
36          statePlaces.body("name", hasItem(name));
37          stateHistory.body("", hasSize(0));
38      }
39
40      @Test
41      public void invalid() {
42          // Given
43          var name = "Castellon de la Plana";
44          var location = client.location.addLocation(name);
45          var coords = location.extract().jsonPath().getString("coords");
46          Mockito.reset(spy.accountManager);
47
48          // When
49          var response = client.history.restoreLocation(coords);
50
51          // Then
52          Mockito.verify(spy.accountManager, never()).saveAccount(any());
53          response.statusCode(HttpStatus.NOT_FOUND.value());
54          var statePlaces = client.location.getLocations();
55          var stateHistory = client.history.getLocations();
56          statePlaces.body("", hasSize(1));
57          statePlaces.body("name", hasItem(name));
58          stateHistory.body("", hasSize(0));
59      }
60  }
```

Fragmento de código 10: Test de integración del requisito basico 1 → HU05

```
17  public class History05 extends SessionTest {
18      @Test
19      public void valid() {
20          // Given
21          var name = "Castellon";
22          var locationMock = new LocationModel(name, 39.980, -0.033);
23          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
24          var location = client.location.addLocation(name);
25          var coords = location.extract().jsonPath().getString("coords");
26          client.location.removeLocation(coords);
27          Mockito.reset(spy.accountManager);
28
29          // When
30          var response = client.history.restoreLocation(coords);
31
32          // Then
33          Mockito.verify(spy.accountManager).saveAccount(any());
34          response.statusCode(HttpStatus.OK.value());
35          var statePlaces = client.location.getLocations();
36          var stateHistory = client.history.getLocations();
37          statePlaces.body("", hasSize(1));
38          statePlaces.body("name", hasItem(name));
39          stateHistory.body("", hasSize(0));
40      }
41
42      @Test
43      public void invalid() {
44          // Given
```

```
45          var name = "Castellon";
46          var locationMock = new LocationModel(name, 39.980, -0.033);
47          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
48          var location = client.location.addLocation(name);
49          var coords = location.extract().jsonPath().getString("coords");
50          Mockito.reset(spy.accountManager);
51
52          // When
53          var response = client.history.restoreLocation(coords);
54
55          // Then
56          Mockito.verify(spy.accountManager, never()).saveAccount(any());
57          response.statusCode(HttpStatus.NOT_FOUND.value());
58          var statePlaces = client.location.getLocations();
59          var stateHistory = client.history.getLocations();
60          statePlaces.body("", hasSize(1));
61          statePlaces.body("name", hasItem(name));
62          stateHistory.body("", hasSize(0));
63      }
64  }
```

1.1.1.6 Requisito básico 1, Historia de usuario 6

Como usuario quiero obtener las coordenadas geográficas de una ubicación a partir de su topónimo, con el fin de facilitar la obtención de información en múltiples fuentes públicas (API).

Escenarios

Escenario	Topónimo válido	Servicio disponible	Coordenadas
E1	Si	Si	Si
E2	No	Si	No
E3	Si	No	No
E4	No	No	No

Tabla 11: Escenarios Requisito básico 1, Historia de Usuario 6

Análisis

El escenario E1 podría subdividirse en al menos tres pruebas de aceptación diferentes. Una usando topónimos válidos pero poco conocidos (Antártida), otra con topónimos válidos con una única representación (Madrid) y otra con topónimos válidos con múltiples representaciones (Castelló y Castellón).

El escenario E2 podría subdividirse en al menos tres pruebas de aceptación diferentes. Una usando un topónimo invlido formado por símbolos y otros caracteres especiales, otra con un topónimo invlido formado por caracteres normales y otra con un topónimo invalido formado por una pequeña variación de uno valido.

Pruebas de aceptación

	E1 (válido)	**E2 (inválido)**
Given	▪ Un usuario registrado ▪ API de geocoding disponible ▪ El topónimo 'Castellón'	▪ Un usuario registrado ▪ API de geocoding disponible ▪ El topónimo 'INVALIDO'
When	Se intenta transformar ese topónimo en unas coordenadas	Se intenta transformar ese topónimo en unas coordenadas
Then	Devuelve '39,0'	No devuelve coordenadas

Tabla 12: Pruebas de aceptación, Requisito básico 1, Historia de Usuario 6

Tests de integración y aceptación

Los siguientes tests están en el repositorio `app` del proyecto y se encuentran en las rutas:

`src/test/java/app/test/acceptance/basic/requirement01/History06.java`

`src/test/java/app/test/integration/basic/requirement01/History06.java`

Fragmento de código 11: Test de aceptación del requisito basico 1 → HU06

```java
public class History06 extends SessionTest {
    @Test
    public void valid() {
        // Given
        var name = "Castellon";
        var coords = "39.970,-0.050";

        // When
        var response = client.query.query(name);

        // Then
        response.statusCode(HttpStatus.OK.value());
        response.body("", hasSize(1));
        response.body("coords", hasItem(coords));
    }

    @Test
    public void invalid() {
        // Given
        var name = "INVALIDO";

        // When
        var response = client.query.query(name);

        // Then
        response.statusCode(HttpStatus.OK.value());
        response.body("", hasSize(0));
    }
}
```

Fragmento de código 12: Test de integración del requisito basico 1 → HU06

```
18  public class History06 extends SessionTest {
19      @Test
20      public void valid() {
21          // Given
22          var name = "Castellon";
23          var locationMock = new LocationModel(name, 39.980, -0.033);
24          var coords = locationMock.getCoords();
25          Mockito.doReturn(List.of(locationMock)).when(spy.queryManager).
                getAllData(name);
26
27          // When
28          var response = client.query.query(name);
29
30          // Then
31          response.statusCode(HttpStatus.OK.value());
32          response.body("", hasSize(1));
33          response.body("coords", hasItem(coords));
34      }
35
36      @Test
37      public void invalid() {
38          // Given
39          var name = "INVALIDO";
40          Mockito.doReturn(Collections.emptyList()).when(spy.queryManager).
                getAllData(name);
41
42          // When
43          var response = client.query.query(name);
44
45          // Then
46          response.statusCode(HttpStatus.OK.value());
47          response.body("", hasSize(0));
48      }
49  }
```

1.1.1.7 Requisito básico 1, Historia de usuario 7

Como usuario quiero obtener el topónimo más próximo a las coordenadas geográficas de una ubicación, con el fin de facilitar la obtención de información en múltiples fuentes públicas (API).

Escenarios

Escenario	**Coordenadas válidas**	**Topónimo disponible**	**Servicio disponible**	**Topónimo más próximo**
E1	Si	Si	Si	Si
E2	Si	Si	No	No
E3	SI	No	Si	No
E4	Si	No	No	No
E5	No	No	No	No

Tabla 13: Escenarios Requisito básico 1, Historia de Usuario 7

Análisis

El escenario E1 podría subdividirse en al menos tres pruebas de aceptación diferentes. Una usando unas coordenadas válidas pero en el límite sintáctico (90.0, 180.0), otra con unas coordenadas válidas en la frontera de dos ciudades (Castellón y Benicassim) y otra otra unas coordenadas válidas una ubicación concreta (Castellon).

El escenario E3 podría subdividirse en al menos tres pruebas de aceptación diferentes. Una usando unas coordenadas inválidas formadas por símbolos y otros caracteres especiales, otra con unas coordenadas inválidas lejos del límite sintáctico (180.0, 360.0) y otra con unas coordenadas inválidas cerca del límite sintáctico (90.1, 180.1).

Pruebas de aceptación

	E1 (válido)	**E3 (inválido)**
Given	▪ Un usuario registrado ▪ API geocoding disponible ▪ Las coordenadad '39,0'	▪ Un usuario registrado ▪ API geocoding disponible ▪ Las coordenadas '31,-44'
When	Se realiza una petición a la API para esas coordenadas	Se realiza una petición a la API para esas coordenadas
Then	Devuelve 'Playa de olivia'	No devuelve una ubicación

Tabla 14: Pruebas de aceptación, Requisito básico 1, Historia de Usuario 7

Tests de integración y aceptación

Los siguientes tests están en el repositorio app del proyecto y se encuentran en las rutas:

- `src/test/java/app/test/acceptance/basic/requirement01/History07.java`
- `src/test/java/app/test/integration/basic/requirement01/History07.java`

Fragmento de código 13: Test de aceptación del requisito basico 1 → HU07

```
12  public class History07 extends SessionTest {
13      @Test
14      public void valid() {
15          // Given
16          var name = "Castelló de la Plana";
17          var coords = "39.980,-0.033";
18
19          // When
20          var response = client.query.query(coords);
21
22          // Then
23          response.statusCode(HttpStatus.OK.value());
24          response.body("", hasSize(1));
25          response.body("name", hasItem(name));
26      }
27
28      @Test
29      public void invalid() {
30          // Given
31          var coords = "180,360";
32
33          // When
34          var response = client.query.query(coords);
35
36          // Then
37          response.statusCode(HttpStatus.OK.value());
38          response.body("", hasSize(0));
39      }
40  }
```

Fragmento de código 14: Test de integración del requisito basico 1 → HU07

```
17  public class History07 extends SessionTest {
18      @Test
19      public void valid() {
20          // Given
21          var name = "Castellon";
22          var locationMock = new LocationModel(name, 39.980, -0.033);
23          var coords = locationMock.getCoords();
24          Mockito.doReturn(List.of(locationMock)).when(spy.queryManager).
                getAllData(coords);
25
26          // When
27          var response = client.query.query(coords);
28
```

```
29          // Then
30          response.statusCode(HttpStatus.OK.value());
31          response.body("", hasSize(1));
32          response.body("name", hasItem(name));
33      }
34
35      @Test
36      public void invalid() {
37          // Given
38          var coords = "180,360";
39          Mockito.doReturn(Collections.emptyList()).when(spy.queryManager).
                getAllData(coords);
40
41          // When
42          var response = client.query.query(coords);
43
44          // Then
45          response.statusCode(HttpStatus.OK.value());
46          response.body("", hasSize(0));
47      }
48  }
```

1.1.1.8 Requisito básico 1, Historia de usuario 8

Como usuario quiero asignar un alias a una ubicación, con el fin de personalizar el uso del sistema.

Escenarios

Escenario	Ubicaciones previas	Alias proporcionado	Alias resultante	BBDD modificada
E1	1	Si	Alias	Si
E2	1	No	Topónimo	No

Tabla 15: Escenarios Requisito básico 1, Historia de Usuario 8

Análisis

El escenario E1 podría subdividirse en al menos tres pruebas de aceptación diferentes. Una usando un alias vacío, otra un alias con caracteres en blanco y otra con un alias con caracteres normales.

El escenario E2 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería no proporcionar un alias y comprobar que se restablece el original.

Pruebas de aceptación

	E1 (válido)	**E2 (inválido)**
Given	▪ Un usuario registrado ▪ Una ubicación guardado: 'Castellón'	▪ Un usuario registrado ▪ Una ubicación guardada: 'Castellón'
When	Cuando actualizas el alias a 'CS'	No actualizas el alias
Then	Alias ahora es 'CS'	Alias sigue siendo igual que el por defecto, que es el topónimo, en este caso 'Castellon'

Tabla 16: Pruebas de aceptación, Requisito básico 1, Historia de Usuario 8

Tests de integración y aceptación

Los siguientes tests están en el repositorio `app` del proyecto y se encuentran en las rutas:

`src/test/java/app/test/acceptance/basic/requirement01/History08.java`

`src/test/java/app/test/integration/basic/requirement01/History08.java`

Fragmento de código 15: Test de aceptación del requisito basico 1 → HU08

```
15  public class History08 extends SessionTest {
16      @Test
17      public void valid() {
18          // Given
19          var alias = "CS";
20          var name = "Castellon";
21          var location = client.location.addLocation(name);
22          var coords = location.extract().jsonPath().getString("coords");
23          Mockito.reset(spy.accountManager);
24
25          // When
26          var response = client.location.updateLocation(coords, alias);
27
28          // Then
29          Mockito.verify(spy.accountManager).saveAccount(any());
30          response.statusCode(HttpStatus.OK.value());
31          var state = client.location.getLocations();
32          state.body("", hasSize(1));
33          state.body("alias", hasItem(alias));
34      }
35
36      @Test
37      public void invalid() {
38          // Given
39          var name = "Castellon de la Plana";
40          client.location.addLocation(name);
41          Mockito.reset(spy.accountManager);
42
```

```
43          // When
44          // No operation
45
46          // Then
47          Mockito.verify(spy.accountManager, never()).saveAccount(any());
48          var state = client.location.getLocations();
49          state.body("", hasSize(1));
50          state.body("alias", hasItem(name));
51      }
52  }
```

Fragmento de código 16: Test de integración del requisito basico 1 → HU08

```
16  public class History08 extends SessionTest {
17      @Test
18      public void valid() {
19          // Given
20          var alias = "CS";
21          var name = "Castellon";
22          var locationMock = new LocationModel(name, 39.980, -0.033);
23          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
24          var location = client.location.addLocation(name);
25          var coords = location.extract().jsonPath().getString("coords");
26          Mockito.reset(spy.accountManager);
27
28          // When
29          var response = client.location.updateLocation(coords, alias);
30
31          // Then
32          Mockito.verify(spy.accountManager).saveAccount(any());
33          response.statusCode(HttpStatus.OK.value());
34          var state = client.location.getLocations();
35          state.body("", hasSize(1));
36          state.body("alias", hasItem(alias));
37      }
38
39      @Test
40      public void invalid() {
41          // Given
42          var name = "Castellon";
43          var locationMock = new LocationModel(name, 39.980, -0.033);
44          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
45          client.location.addLocation(name);
46          Mockito.reset(spy.accountManager);
47
48          // When
49          // No operation
50
51          // Then
52          Mockito.verify(spy.accountManager, never()).saveAccount(any());
53          var state = client.location.getLocations();
54          state.body("", hasSize(1));
55          state.body("alias", hasItem(name));
56      }
57  }
```

1.1.1.9 Requisito básico 1, Historia de usuario 9

Como usuario quiero desactivar una ubicación activa, con el fin de reducir temporalmente la cantidad de información a consultar.

Escenarios

Escenario	Ubicaciones activas previas	Ubicaciones desactivadas previas	Ubicaciones activas después	Ubicaciones desactivadas después	BBDD modificada
E1	1	0	0	1	Si
E2	0	1	0	1	No

Tabla 17: Escenarios Requisito básico 1, Historia de Usuario 9

Análisis

El escenario E1 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería desactivar una ubicación que ya esté en el sistema pero estuviera activado.

El escenario E2 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería desactivar una ubicación que ya esté en el sistema pero estuviera desactivado.

Pruebas de aceptación

	E1 (válido)	E2 (inválido)
Given	▪ Un usuario registrado ▪ Una ubicación activa en el sistema 'Castellón' ▪ Nunguna ubic. desactivada	▪ Un usuario registrado ▪ Ninguna ubic. activada ▪ Una ubic. desactivada en el sistema 'Castellón'
When	Se realiza la petición de desactivación sobre 'Castellón'	Se realiza la petición de desactivación sobre 'Castellón'
Then	▪ Ninguna ubic. activada ▪ Una ubic. desactivada en el sistema 'Castellón'	▪ Ninguna ubic. activada ▪ Una ubic. desactivada en el sistema 'Castellón'

Tabla 18: Pruebas de aceptación, Requisito básico 1, Historia de Usuario 9

Tests de integración y aceptación

Los siguientes tests están en el repositorio app del proyecto y se encuentran en las rutas:

- src/test/java/app/test/acceptance/basic/requirement01/History09.java
- src/test/java/app/test/integration/basic/requirement01/History09.java

Fragmento de código 17: Test de aceptación del requisito basico 1 → HU09

```java
public class History09 extends SessionTest {
    @Test
    public void valid() {
        // Given
        var name = "Castellon de la Plana";
        var location = client.location.addLocation(name);
        var coords = location.extract().jsonPath().getString("coords");
        Mockito.reset(spy.accountManager);

        // When
        var response = client.location.removeLocation(coords);

        // Then
        Mockito.verify(spy.accountManager).saveAccount(any());
        response.statusCode(HttpStatus.OK.value());
        var statePlaces = client.location.getLocations();
        var stateHistory = client.history.getLocations();
        statePlaces.body("", hasSize(0));
        stateHistory.body("", hasSize(1));
        stateHistory.body("name", hasItem(name));
    }

    @Test
    public void invalid() {
        // Given
        var name = "Castellon de la Plana";
        var location = client.location.addLocation(name);
        var coords = location.extract().jsonPath().getString("coords");
        client.location.removeLocation(coords);
        Mockito.reset(spy.accountManager);

        // When
        var response = client.location.removeLocation(coords);

        // Then
        Mockito.verify(spy.accountManager, never()).saveAccount(any());
        response.statusCode(HttpStatus.NOT_FOUND.value());
        var statePlaces = client.location.getLocations();
        var stateHistory = client.history.getLocations();
        statePlaces.body("", hasSize(0));
        stateHistory.body("", hasSize(1));
        stateHistory.body("name", hasItem(name));
    }
}
```

Fragmento de código 18: Test de integración del requisito basico 1 → HU09

```
16  public class History09 extends SessionTest {
17      @Test
18      public void valid() {
19          // Given
20          var name = "Castellon";
21          var locationMock = new LocationModel(name, 39.980, -0.033);
22          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
23          var location = client.location.addLocation(name);
24          var coords = location.extract().jsonPath().getString("coords");
25          Mockito.reset(spy.accountManager);
26
27          // When
28          var response = client.location.removeLocation(coords);
29
30          // Then
31          Mockito.verify(spy.accountManager).saveAccount(any());
32          response.statusCode(HttpStatus.OK.value());
33          var statePlaces = client.location.getLocations();
34          var stateHistory = client.history.getLocations();
35          statePlaces.body("", hasSize(0));
36          stateHistory.body("", hasSize(1));
37          stateHistory.body("name", hasItem(name));
38      }
39
40      @Test
41      public void invalid() {
42          // Given
43          var name = "Castellon";
44          var locationMock = new LocationModel(name, 39.980, -0.033);
45          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
46          var location = client.location.addLocation(name);
47          var coords = location.extract().jsonPath().getString("coords");
48          client.location.removeLocation(coords);
49          Mockito.reset(spy.accountManager);
50
51          // When
52          var response = client.location.removeLocation(coords);
53
54          // Then
55          Mockito.verify(spy.accountManager, never()).saveAccount(any());
56          response.statusCode(HttpStatus.NOT_FOUND.value());
57          var statePlaces = client.location.getLocations();
58          var stateHistory = client.history.getLocations();
59          statePlaces.body("", hasSize(0));
60          stateHistory.body("", hasSize(1));
61          stateHistory.body("name", hasItem(name));
62      }
63  }
```

1.1.1.10 Requisito básico 1, Historia de usuario 10

Como usuario quiero dar de baja una ubicación disponible, con el fin de eliminar información que ya no resulta de interés.

Escenarios

Escenario	Ubicaciones previas	Ubicación registrada	Ubicaciones después	BBDD modificada
E1	1	Si	0	Si
E2	1	No	1	No

Tabla 19: Escenarios Requisito básico 1, Historia de Usuario 10

Análisis

El escenario E1 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería dar de baja una ubicación que ya esté en el sistema.

El escenario E2 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería dar de baja una ubicación que ya no esté en el sistema.

Pruebas de aceptación

	E1 (válido)	E2 (inválido)
Given	▪ Un usuario registrado ▪ Una ubicación desact. 'Castellón'	▪ Un usuario registrado ▪ Una ubicación desact. 'Castellón'
When	Intenta dar de baja 'Castellon'	Intenta dar de baja 'Valencia'
Then	No hay ubicaciones desactivadas	Una ubicación desactivada 'Castellon'

Tabla 20: Pruebas de aceptación, Requisito básico 1, Historia de Usuario 10

Tests de integración y aceptación

Los siguientes tests están en el repositorio app del proyecto y se encuentran en las rutas:

- src/test/java/app/test/acceptance/basic/requirement01/History10.java
- src/test/java/app/test/integration/basic/requirement01/History10.java

Fragmento de código 19: Test de aceptación del requisito basico 1 → HU10

```java
public class History10 extends SessionTest {
    @Test
    public void valid() {
        // Given
        var name = "Castellon de la Plana";
        var location = client.location.addLocation(name);
        var coords = location.extract().jsonPath().getString("coords");
        client.location.removeLocation(coords);
        Mockito.reset(spy.accountManager);

        // When
        var response = client.history.removeLocation(coords);

        // Then
        Mockito.verify(spy.accountManager).saveAccount(any());
        response.statusCode(HttpStatus.OK.value());
        var statePlaces = client.location.getLocations();
        var stateHistory = client.history.getLocations();
        statePlaces.body("", hasSize(0));
        stateHistory.body("", hasSize(0));
    }

    @Test
    public void invalid() {
        // Given
        var name = "Castellon de la Plana";
        var location = client.location.addLocation(name);
        var coords = location.extract().jsonPath().getString("coords");
        client.location.removeLocation(coords);
        coords = "39.503,-0.405";
        Mockito.reset(spy.accountManager);

        // When
        var response = client.history.removeLocation(coords);

        // Then
        Mockito.verify(spy.accountManager, never()).saveAccount(any());
        response.statusCode(HttpStatus.NOT_FOUND.value());
        var statePlaces = client.location.getLocations();
        var stateHistory = client.history.getLocations();
        statePlaces.body("", hasSize(0));
        stateHistory.body("", hasSize(1));
        stateHistory.body("name", hasItem(name));
    }
}
```

Fragmento de código 20: Test de integración del requisito basico 1 → HU10

```
16  public class History10 extends SessionTest {
17      @Test
18      public void valid() {
19          // Given
20          var name = "Castellon";
21          var locationMock = new LocationModel(name, 39.980, -0.033);
22          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
23          var location = client.location.addLocation(name);
24          var coords = location.extract().jsonPath().getString("coords");
25          client.location.removeLocation(coords);
26          Mockito.reset(spy.accountManager);
27
28          // When
29          var response = client.history.removeLocation(coords);
30
31          // Then
32          Mockito.verify(spy.accountManager).saveAccount(any());
33          response.statusCode(HttpStatus.OK.value());
34          var statePlaces = client.location.getLocations();
35          var stateHistory = client.history.getLocations();
36          statePlaces.body("", hasSize(0));
37          stateHistory.body("", hasSize(0));
38      }
39
40      @Test
41      public void invalid() {
42          // Given
43          var name = "Castellon";
44          var locationMock = new LocationModel(name, 39.980, -0.033);
45          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
46          var location = client.location.addLocation(name);
47          var coords = location.extract().jsonPath().getString("coords");
48          client.location.removeLocation(coords);
49          coords = "39.503,-0.405";
50          Mockito.reset(spy.accountManager);
51
52          // When
53          var response = client.history.removeLocation(coords);
54
55          // Then
56          Mockito.verify(spy.accountManager, never()).saveAccount(any());
57          response.statusCode(HttpStatus.NOT_FOUND.value());
58          var statePlaces = client.location.getLocations();
59          var stateHistory = client.history.getLocations();
60          statePlaces.body("", hasSize(0));
61          stateHistory.body("", hasSize(1));
62          stateHistory.body("name", hasItem(name));
63      }
64  }
```

1.1.2. Requisito básico 2 - Gestionar hasta tres ubicaciones de interés

El segundo requisito básico consta de gestionar hasta tres ubicaciones de interés sobre el que se desea consultar información.

1.1.2.1 Requisito básico 2, Historia de usuario 1

Como usuario quiero consultar información de hasta tres ubicaciones simultáneamente, con el fin de estar al corriente de novedades en todas ellas.
Esta historia está dividida en subhistorias.

Requisito básico 2, Historia de usuario 1, Subhistoria 1

Como usuario quiero consultar información de hasta tres ubicaciones simultáneamente, con el fin de saber todos sus datos a la vez.

Escenarios

Escenario	**Cantidad de ubicaciones**	**Resultado**
E1	0	No
E2	2	Si

Tabla 21: Escenarios Requisito básico 2, Historia de Usuario 1, Subhistoria 1

Análisis

El escenario E1 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería tener al dos ubicaciones activas dadas de alta y está responder sus datos.

El escenario E2 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería no tener ninguna ubicación activa dada de alta y esta no responder.

Pruebas de aceptación

	E1 (válido)	**E2 (inválido)**
Given	▪ Un usuario registrado ▪ Tiene dos ubicaciones guardadas: ('Castellón', 'Alicante')	▪ Un usuario registrado ▪ No tiene ninguna ubicación activa
When	Cuando solicita a la API la información sobre sus ubicaciones guardadas	Cuando solicita a la API la información sobre sus ubicaciones activas
Then	La API deberá devolver 2 paquetes de información, uno por cada ubicación guardada	La API devolverá un paquete vacío

Tabla 22: Pruebas de aceptación, Requisito básico 2, Historia de Usuario 1, Subhistoria 1

Tests de integración y aceptación

Los siguientes tests están en el repositorio `app` del proyecto y se encuentran en las rutas:

`src/test/java/app/test/acceptance/basic/requirement02/history01/Subhistory01.java`

`src/test/java/app/test/integration/basic/requirement02/history01/Subhistory01.java`

Fragmento de código 21: Test de aceptación del requisito basico 2 → HU01 y SH01

```java
public class Subhistory01 extends SessionTest {
    @Test
    public void valid() {
        // Given
        var nameA = "Castellon de la Plana";
        client.location.addLocation(nameA);

        var nameB = "Alicante";
        client.location.addLocation(nameB);

        // When
        var response = client.location.getLocations();

        // Then
        response.statusCode(HttpStatus.OK.value());
        response.body("", hasSize(2));
        response.body("name", hasItems(nameA, nameB));
    }

    @Test
    public void invalid() {
        // Given
```

```
        // No locations

        // When
        var response = client.location.getLocations();

        // Then
        response.statusCode(HttpStatus.OK.value());
        response.body("", hasSize(0));
    }
}
```

Fragmento de código 22: Test de integración del requisito basico 2 → HU01 y SH01

```
public class Subhistory01 extends SessionTest {
    @Test
    public void valid() {
        // Given
        var nameA = "Castellon";
        var locationMockA = new LocationModel(nameA, 39.980, -0.033);
        Mockito.doReturn(locationMockA).when(spy.queryManager).getData(nameA);
        client.location.addLocation(nameA);

        var nameB = "Alicante";
        var locationMockB = new LocationModel(nameB, 38.53996, -0.50579);
        Mockito.doReturn(locationMockB).when(spy.queryManager).getData(nameB);
        client.location.addLocation(nameB);

        // When
        var response = client.location.getLocations();

        // Then
        response.statusCode(HttpStatus.OK.value());
        response.body("", hasSize(2));
        response.body("name", hasItems(nameA, nameB));
    }

    @Test
    public void invalid() {
        // Given
        // No locations

        // When
        var response = client.location.getLocations();

        // Then
        response.statusCode(HttpStatus.OK.value());
        response.body("", hasSize(0));
    }
}
```

Requisito básico 2, Historia de usuario 1, Subhistoria 2

Como usuario quiero consultar información de hasta tres servicios de una ubicación simultáneamente, con el fin de estar al corriente de novedades en todas ellas.

Escenarios

Escenario	Cantidad de ubicaciones	Cantidad de servicios	Resultado
E1	1	0	No
E2	1	2	Si
E3	0	0	No

Tabla 23: Escenarios Requisito básico 2, Historia de Usuario 1, Subhistoria 2

Análisis

El escenario E2 podría subdividirse en al menos dos pruebas de aceptación diferentes. Una naive que mientras exista respuesta de ambos asume que ha funcionado, otra sanity que realice lo mismo pero además comprueba el nombre de los campos y su tipado.

El escenario E3 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería no tener servicios, ni ubicaciones y este no responder.

Pruebas de aceptación

	E2 (válido)	E3 (inválido)
Given	▪ Un usuario registrado ▪ Una ubicación activa 'Castellón' ▪ Tiene dos servicios activos ('Clima', 'Eventos')	▪ Un usuario registrado. ▪ No tiene ninguna ubicación activa. ▪ No tiene ningún servicio activo.
When	Cuando solicita a la API la información sobre sus servicios	Cuando solicita a la API la información sobre sus servicios
Then	La API deberá devolver 2 paquetes de información, uno por cada servicio activo	La API devolverá un paquete vacío

Tabla 24: Pruebas de aceptación, Requisito básico 2, Historia de Usuario 1, Subhistoria 2

Tests de integración y aceptación

Los siguientes tests están en el repositorio app del proyecto y se encuentran en las rutas:

`src/test/java/app/test/acceptance/basic/requirement02/history01/Subhistory02.java`

`src/test/java/app/test/integration/basic/requirement02/history01/Subhistory02.java`

Fragmento de código 23: Test de aceptación del requisito basico 2 → HU01 y SH02

```
14  public class Subhistory02 extends SessionTest {
15      @Test
16      public void valid() {
17          // Given
18          var typeA = ServiceType.WEATHER.name();
19          client.service.enableService(typeA);
20
21          var typeB = ServiceType.EVENTS.name();
22          client.service.enableService(typeB);
23
24          var name = "Castellon";
25          var location = client.location.addLocation(name);
26          var coords = location.extract().jsonPath().getString("coords");
27
28          // When
29          var response = client.service.getServicesForLocation(coords);
30
31          // Then
32          response.statusCode(HttpStatus.OK.value());
33          response.body(setupEnabledQuery(true, ""), hasSize(2));
34          response.body(setupEnabledQuery(true, "service.type"), hasItems(typeA,
                typeB));
35      }
36
37      @Test
38      public void invalid() {
39          // Given
40          // No services
41          // No locations
42          var name = "INVALIDO";
43
44          // When
45          var response = client.service.getServicesForLocation(name);
46
47          // Then
48          response.statusCode(HttpStatus.NOT_FOUND.value());
49      }
50  }
```

Fragmento de código 24: Test de integración del requisito basico 2 → HU01 y SH02

```
17  public class Subhistory02 extends SessionTest {
```

```
18      @Test
19      public void valid() {
20          // Given
21          var typeA = ServiceType.WEATHER.name();
22          client.service.enableService(typeA);
23          Mockito.doReturn(true).when(spy.weatherService).getData(any());
24
25          var typeB = ServiceType.EVENTS.name();
26          client.service.enableService(typeB);
27          Mockito.doReturn(true).when(spy.eventsService).getData(any());
28
29          var name = "Castellon";
30          var locationMock = new LocationModel(name, 39.980, -0.033);
31          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
32          var location = client.location.addLocation(name);
33          var coords = location.extract().jsonPath().getString("coords");
34
35          // When
36          var response = client.service.getServicesForLocation(coords);
37
38          // Then
39          response.statusCode(HttpStatus.OK.value());
40          response.body(setupEnabledQuery(true, ""), hasSize(2));
41          response.body(setupEnabledQuery(true, "service.type"), hasItems(typeA,
                typeB));
42      }
43
44      @Test
45      public void invalid() {
46          // Given
47          // No services
48          // No locations
49          var name = "INVALIDO";
50
51          // When
52          var response = client.service.getServicesForLocation(name);
53
54          // Then
55          response.statusCode(HttpStatus.NOT_FOUND.value());
56      }
57  }
```

1.1.2.2 Requisito básico 2, Historia de usuario 2

Como usuario quiero poder elegir servicios de información (API) independientes para cada ubicación, con el doble fin de consultar sólo información de interés y contribuir a la gestión eficiente de recursos.
Esta historia está dividida en subhistorias.

Requisito básico 2, Historia de usuario 2, Subhistoria 1

Como usuario quiero poder activar servicios de información (API) independientes para cada ubicación, con el doble fin de consultar sólo información de interés y contribuir a la gestión eficiente de recursos.

Escenarios

Escenario	Cantidad de ubicaciones	Servicios disponibles	Servicio activo	Servicio pedido válido	Servicio activo después	BBDD modificada
E1	1	1	No	Si	Si	Si
E2	1	1	Si	Si	Si	No
E3	1	1	No	No	No	No

Tabla 25: Escenarios Requisito básico 2, Historia de Usuario 2, Subhistoria 1

Análisis

El escenario E1 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería tener una ubicación guardada sin servicios, un servicio disponible y activar este servicio en la ubicación.

El escenario E3 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería tener una ubicación guardada sin servicios, un servicio disponible y activar un servicio que no sea este en la ubicación.

Pruebas de aceptación

	E1 (válido)	E3 (inválido)
Given	▪ Un usuario registrado ▪ Una ubicación guardada → 'Castellón' ▪ Un servicio disponible 'Clima' ▪ La ubicación no tiene servicios	▪ Un usuarior registrado ▪ Una ubicación guardada → 'Castellón' ▪ Un servicio disponible 'Clima' ▪ La ubicación no tiene servicios
When	El usuario activa el servicio 'Clima' en 'Castellón'	El usuario activa el servicio 'INVALIDO' en 'Castellón'
Then	'Castellón' tiene el servicio 'Clima' activo	'Castellón' no tiene servicios activos

Tabla 26: Pruebas de aceptación, Requisito básico 2, Historia de Usuario 2, Subhistoria 1

Tests de integración y aceptación

Los siguientes tests están en el repositorio app del proyecto y se encuentran en las rutas:

`src/test/java/app/test/acceptance/basic/requirement02/history02/Subhistory01.java`

`src/test/java/app/test/integration/basic/requirement02/history02/Subhistory01.java`

Fragmento de código 25: Test de aceptación del requisito basico 2 → HU02 y SH01

```java
public class Subhistory01 extends SessionTest {
    @Test
    public void valid() {
        // Given
        var name = "Castellon";
        var type = ServiceType.WEATHER.name();
        var location = client.location.addLocation(name);
        var coords = location.extract().jsonPath().getString("coords");
        client.service.enableService(type);
        Mockito.reset(spy.accountManager);

        // When
        var response = client.service.enableServiceForLocation(coords, type);

        // Then
        Mockito.verify(spy.accountManager).saveAccount(any());
        response.statusCode(HttpStatus.OK.value());
        var status = client.service.getServicesForLocation(coords);
        status.body(setupEnabledQuery(true, ""), hasSize(1));
        status.body(setupEnabledQuery(true, "service.type"), hasItem(type));
    }

    @Test
    public void invalid() {
        // Given
        var name = "Castellon";
        var type = ServiceType.WEATHER.name();
        var location = client.location.addLocation(name);
        var coords = location.extract().jsonPath().getString("coords");
        client.service.enableService(type);
        Mockito.reset(spy.accountManager);
        type = "INVALIDO";

        // When
        var response = client.service.enableServiceForLocation(coords, type);

        // Then
        Mockito.verify(spy.accountManager, never()).saveAccount(any());
        response.statusCode(HttpStatus.BAD_REQUEST.value());
        var status = client.service.getServicesForLocation(coords);
        status.body(setupEnabledQuery(true, ""), hasSize(0));
    }
}
```

Fragmento de código 26: Test de integración del requisito basico 2 → HU02 y SH01

```
18  public class Subhistory01 extends SessionTest {
19      @Test
20      public void valid() {
21          // Given
22          var name = "Castellon";
23          var type = ServiceType.WEATHER.name();
24          var locationMock = new LocationModel(name, 39.980, -0.033);
25          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
26          Mockito.doReturn(true).when(spy.weatherService).getData(any());
27          var location = client.location.addLocation(name);
28          var coords = location.extract().jsonPath().getString("coords");
29          client.service.enableService(type);
30          Mockito.reset(spy.accountManager);
31
32          // When
33          var response = client.service.enableServiceForLocation(coords, type);
34
35          // Then
36          Mockito.verify(spy.accountManager).saveAccount(any());
37          response.statusCode(HttpStatus.OK.value());
38          var status = client.service.getServicesForLocation(coords);
39          status.body(setupEnabledQuery(true, ""), hasSize(1));
40          status.body(setupEnabledQuery(true, "service.type"), hasItem(type));
41      }
42
43      @Test
44      public void invalid() {
45          // Given
46          var name = "Castellon";
47          var type = ServiceType.WEATHER.name();
48          var locationMock = new LocationModel(name, 39.980, -0.033);
49          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
50          Mockito.doReturn(true).when(spy.weatherService).getData(any());
51          var location = client.location.addLocation(name);
52          var coords = location.extract().jsonPath().getString("coords");
53          client.service.enableService(type);
54          Mockito.reset(spy.accountManager);
55          type = "INVALIDO";
56
57          // When
58          var response = client.service.enableServiceForLocation(coords, type);
59
60          // Then
61          Mockito.verify(spy.accountManager, never()).saveAccount(any());
62          response.statusCode(HttpStatus.BAD_REQUEST.value());
63          var status = client.service.getServicesForLocation(coords);
64          status.body(setupEnabledQuery(true, ""), hasSize(0));
65      }
66  }
```

Requisito básico 2, Historia de usuario 2, Subhistoria 2

Como usuario quiero poder desactivar servicios de información (API) independientes para cada ubicación, con el doble fin de consultar sólo información de interés y contribuir a la gestión eficiente de recursos.

Escenarios

Escenario	Cantidad de ubicaciones	Servicios disponibles	Servicio activo	Servicio pedido válido	Servicio activo después	BBDD modificada
E1	1	1	No	Si	No	No
E2	1	1	Si	Si	No	Si
E3	1	1	Si	No	Si	No

Tabla 27: Escenarios Requisito básico 2, Historia de Usuario 2, Subhistoria 2

Análisis

El escenario E1 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería tener una ubicación guardada con servicios, un servicio disponible y desactivar este servicio en la ubicación.

El escenario E3 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería tener una ubicación guardada con servicios, un servicio disponible y desactivar un servicio que no sea este en la ubicación.

Pruebas de aceptación

	E2 (válido)	**E3 (inválido)**
Given	▪ Un usuario registrado ▪ Una ubicación guardada 'Castellón' ▪ Un servicio disponible 'Clima' ▪ 'Castellón tiene el servicio 'Clima' activo	▪ Un usuario registrado ▪ Una ubicación guardada 'Castellón' ▪ Un servicio disponible 'Clima' ▪ 'Castellón' tiene el servicio 'Clima' activo
When	El usuario desactiva el servicio 'Clima' en 'Castellón'	El usuario desactiva el servicio 'INVALIDO' en 'Castellón'
Then	'Castellón' no tiene servicios activos	'Castellón' tiene el servicio 'Clima' activo

Tabla 28: Pruebas de aceptación, Requisito básico 2, Historia de Usuario 2, Subhistoria 2

Tests de integración y aceptación

Los siguientes tests están en el repositorio app del proyecto y se encuentran en las rutas:

`src/test/java/app/test/acceptance/basic/requirement02/history02/Subhistory02.java`

`src/test/java/app/test/integration/basic/requirement02/history02/Subhistory02.java`

Fragmento de código 27: Test de aceptación del requisito basico 2 → HU02 y SH02

```java
public class Subhistory02 extends SessionTest {
    @Test
    public void valid() {
        // Given
        var name = "Castellon";
        var type = ServiceType.WEATHER.name();
        client.service.enableService(type);
        var location = client.location.addLocation(name);
        var coords = location.extract().jsonPath().getString("coords");
        Mockito.reset(spy.accountManager);

        // When
        var response = client.service.disableServiceForLocation(coords, type);

        // Then
        Mockito.verify(spy.accountManager).saveAccount(any());
        response.statusCode(HttpStatus.OK.value());
        var status = client.service.getServicesForLocation(coords);
        status.body(setupEnabledQuery(true, ""), hasSize(0));
    }

    @Test
    public void invalid() {
        // Given
        var name = "Castellon";
        var typeA = ServiceType.WEATHER.name();
        client.service.enableService(typeA);
        var location = client.location.addLocation(name);
        var coords = location.extract().jsonPath().getString("coords");
        Mockito.reset(spy.accountManager);
        var typeB = "INVALIDO";

        // When
        var response = client.service.disableServiceForLocation(coords, typeB);

        // Then
        Mockito.verify(spy.accountManager, never()).saveAccount(any());
        response.statusCode(HttpStatus.BAD_REQUEST.value());
        var status = client.service.getServicesForLocation(coords);
        status.body(setupEnabledQuery(true, ""), hasSize(1));
        status.body(setupEnabledQuery(true, "service.type"), hasItem(typeA));
    }
}
```

Fragmento de código 28: Test de integración del requisito basico 2 → HU02 y SH02

```
18  public class Subhistory02 extends SessionTest {
19      @Test
20      public void valid() {
21          // Given
22          var name = "Castellon";
23          var type = ServiceType.WEATHER.name();
24          var locationMock = new LocationModel(name, 39.980, -0.033);
25          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
26          Mockito.doReturn(true).when(spy.weatherService).getData(any());
27          client.service.enableService(type);
28          var location = client.location.addLocation(name);
29          var coords = location.extract().jsonPath().getString("coords");
30          Mockito.reset(spy.accountManager);
31
32          // When
33          var response = client.service.disableServiceForLocation(coords, type);
34
35          // Then
36          Mockito.verify(spy.accountManager).saveAccount(any());
37          response.statusCode(HttpStatus.OK.value());
38          var status = client.service.getServicesForLocation(coords);
39          status.body(setupEnabledQuery(true, ""), hasSize(0));
40      }
41
42      @Test
43      public void invalid() {
44          // Given
45          var name = "Castellon";
46          var typeA = ServiceType.WEATHER.name();
47          var locationMock = new LocationModel(name, 39.980, -0.033);
48          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
49          Mockito.doReturn(true).when(spy.weatherService).getData(any());
50          client.service.enableService(typeA);
51          var location = client.location.addLocation(name);
52          var coords = location.extract().jsonPath().getString("coords");
53          Mockito.reset(spy.accountManager);
54          var typeB = "INVALIDO";
55
56          // When
57          var response = client.service.disableServiceForLocation(coords, typeB);
58
59          // Then
60          Mockito.verify(spy.accountManager, never()).saveAccount(any());
61          response.statusCode(HttpStatus.BAD_REQUEST.value());
62          var status = client.service.getServicesForLocation(coords);
63          status.body(setupEnabledQuery(true, ""), hasSize(1));
64          status.body(setupEnabledQuery(true, "service.type"), hasItem(typeA));
65      }
66  }
```

1.1.2.3 Requisito básico 2, Historia de usuario 3

Como usuario quiero consultar fácilmente la lista de ubicaciones activas.

Escenarios

Escenario	Cantidad de ubicaciones	Cantidad de ubicaciones activas	Resultado
E1	2	1	Si
E2	2	0	No

Tabla 29: Escenarios Requisito básico 2, Historia de Usuario 3

Análisis

El escenario E1 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería tener dos ubicaciones guardadas pero solo una activa y al consultar esta devolver solo la activa.

El escenario E2 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería tener dos ubicaciones guardadas pero ninguna activa y al consultar esta no devolver ninguna.

Pruebas de aceptación

	E1 (válido)	E2 (inválido)
Given	▪ Un usuario registrado ▪ Tiene dos ubicaciones guardadas: 'Castellón' y 'Valencia' ▪ De ellas una está activada 'Castellón'	▪ Un usuario registrado ▪ Tiene dos ubicaciones guardadas: 'Castellón' y 'Valencia' ▪ De ellas no hay ninguna activada
When	Cuando solicita a la API la información sobre sus ubicaciones activas	Cuando solicita a la API la información sobre sus ubicaciones activas
Then	▪ La API deberá devolver un paquete de información por cada ubicación activa ▪ En este caso: 'Castellón'	La API devolverá un paquete vacío

Tabla 30: Pruebas de aceptación, Requisito básico 2, Historia de Usuario 3

Tests de integración y aceptación

Los siguientes tests están en el repositorio app del proyecto y se encuentran en las rutas:

- src/test/java/app/test/acceptance/basic/requirement02/History03.java
- src/test/java/app/test/integration/basic/requirement02/History03.java

Fragmento de código 29: Test de aceptación del requisito basico 2 → HU03

```java
public class History03 extends SessionTest {
    @Test
    public void valid() {
        // Given
        var nameA = "Castellon de la Plana";
        client.location.addLocation(nameA);

        var nameB = "Alicante";
        client.location.addLocation(nameB);

        // When
        var response = client.location.getLocations();

        // Then
        response.statusCode(HttpStatus.OK.value());
        response.body("", hasSize(2));
        response.body("name", hasItems(nameA, nameB));
    }

    @Test
    public void invalid() {
        // Given
        // No locations

        // When
        var response = client.location.getLocations();

        // Then
        response.statusCode(HttpStatus.OK.value());
        response.body("", hasSize(0));
    }
}
```

Fragmento de código 30: Test de integración del requisito basico 2 → HU03

```java
public class History03 extends SessionTest {
    @Test
    public void valid() {
        // Given
        var nameA = "Castellon";
        var locationMockA = new LocationModel(nameA, 39.980, -0.033);
        Mockito.doReturn(locationMockA).when(spy.queryManager).getData(nameA);
        client.location.addLocation(nameA);

        var nameB = "Alicante";
```

```
25          var locationMockB = new LocationModel(nameB, 38.53996, -0.50579);
26          Mockito.doReturn(locationMockB).when(spy.queryManager).getData(nameB);
27          client.location.addLocation(nameB);
28
29          // When
30          var response = client.location.getLocations();
31
32          // Then
33          response.statusCode(HttpStatus.OK.value());
34          response.body("", hasSize(2));
35          response.body("name", hasItems(nameA, nameB));
36      }
37
38      @Test
39      public void invalid() {
40          // Given
41          // No locations
42
43          // When
44          var response = client.location.getLocations();
45
46          // Then
47          response.statusCode(HttpStatus.OK.value());
48          response.body("", hasSize(0));
49      }
50  }
```

1.1.2.4 Requisito básico 2, Historia de usuario 4

Como usuario quiero consultar fácilmente la información de cualquiera de las ubicaciones activas por separado.
Esta historia está dividida en subhistorias.

Requisito básico 2, Historia de usuario 4, Subhistoria 1

Como usuario quiero consultar fácilmente la información de cualquiera de las ubicaciones activas por separado.

Escenarios

Escenario	Cantidad de ubicaciones	Cantidad de ubicaciones activas	Resultado
E1	2	2	Si
E2	2	0	No

Tabla 31: Escenarios Requisito básico 2, Historia de Usuario 4, Subhistoria 1

Análisis

El escenario E1 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería tener dos ubicaciones guardadas y ambas activas, al consultar sobre una concreta esta devolver solo esa.

El escenario E2 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería tener dos ubicaciones guardadas pero ninguna activa y al consultar cualquiera esta devolver ninguna.

Pruebas de aceptación

	E1(válido)	**E1 (inválido)**
Given	▪ Un usuario registrado ▪ Tiene dos ubicaciones guardadas ('Castellón' y 'Valencia') ▪ De ellas todas están activadas ▪ ('Castellón' y 'Valencia')	▪ Un usuario registrado ▪ No tiene ninguna ubicación guardada ▪ ('Castellano' y 'Valencia') ▪ Ninguna de ellas está activada
When	Cuando solicita a la API la información sobre 'Castellón'	Cuando solicita a la API la información sobre 'Castellón'
Then	La API deberá devolver información sobre 'Castellón'	La API no devolverá información

Tabla 32: Pruebas de aceptación, Requisito básico 2, Historia de Usuario 4, Subhistoria 1

Tests de integración y aceptación

Los siguientes tests están en el repositorio `app` del proyecto y se encuentran en las rutas:

`src/test/java/app/test/acceptance/basic/requirement02/history04/Subhistory01.java`

`src/test/java/app/test/integration/basic/requirement02/history04/Subhistory01.java`

Fragmento de código 31: Test de aceptación del requisito basico 2 → HU04 y SH01

```
16 public class Subhistory01 extends SessionTest {
17     @Test
18     public void valid() {
19         // Given
20         var nameA = "Valencia";
21         client.location.addLocation(nameA);
22 
23         var nameB = "Castellon de la Plana";
24         var location = client.location.addLocation(nameB);
```

```
25          var coords = location.extract().jsonPath().getString("coords");
26
27          // When
28          var response = client.location.getLocations();
29
30          // Then
31          response.statusCode(HttpStatus.OK.value());
32          response.body("", hasSize(2));
33          response.body(setupCoordsQuery(coords, ""), equalTo(Map.of("name",
                nameB, "alias", nameB, "coords", coords)));
34      }
35
36      @Test
37      public void invalid() {
38          // Given
39          var nameA = "Valencia";
40          var location = client.location.addLocation(nameA);
41          var coords = location.extract().jsonPath().getString("coords");
42          client.location.removeLocation(coords);
43
44          var nameB = "Castellon de la Plana";
45          location = client.location.addLocation(nameB);
46          coords = location.extract().jsonPath().getString("coords");
47          client.location.removeLocation(coords);
48
49          // When
50          var response = client.location.getLocations();
51
52          // Then
53          response.statusCode(HttpStatus.OK.value());
54          response.body("", hasSize(0));
55      }
56  }
```

Fragmento de código 32: Test de integración del requisito basico 2 → HU04 y SH01

```
18  public class Subhistory01 extends SessionTest {
19      @Test
20      public void valid() {
21          // Given
22          var name = "Valencia";
23          var locationMock = new LocationModel(name, 39.503, -0.405);
24          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
25          client.location.addLocation(name);
26
27          name = "Castellon";
28          locationMock = new LocationModel(name, 39.980, -0.033);
29          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
30          var location = client.location.addLocation(name);
31          var coords = location.extract().jsonPath().getString("coords");
32
33          // When
34          var response = client.location.getLocations();
35
36          // Then
37          response.statusCode(HttpStatus.OK.value());
```

```
38          response.body("", hasSize(2));
39          response.body(setupCoordsQuery(coords, ""), equalTo(Map.of("name", name
                , "alias", name, "coords", coords)));
40      }
41
42      @Test
43      public void invalid() {
44          // Given
45          var name = "Valencia";
46          var locationMock = new LocationModel(name, 39.503, -0.405);
47          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
48          var location = client.location.addLocation(name);
49          var coords = location.extract().jsonPath().getString("coords");
50          client.location.removeLocation(coords);
51
52          name = "Castellon";
53          locationMock = new LocationModel(name, 39.980, -0.033);
54          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
55          location = client.location.addLocation(name);
56          coords = location.extract().jsonPath().getString("coords");
57          client.location.removeLocation(coords);
58
59          // When
60          var response = client.location.getLocations();
61
62          // Then
63          response.statusCode(HttpStatus.OK.value());
64          response.body("", hasSize(0));
65      }
66  }
```

Requisito básico 2, Historia de usuario 4, Subhistoria 2

Como usuario quiero consultar fácilmente la información del clima sobre un una ubicación activa.

Debido a que la historia original pide como requisitos que la acción se realiza sobre una ubicación activa (por lo tanto también registrada y válida) y que el servicio del clima puede responder ante cualquier coordenada sintácticamente válida (garantizado por el hecho de estar activa) es imposible crear un caso invalido.

Escenarios

Escenario	Cantidad de ubicaciones	Cantidad de ubicaciones activas	Cantidad de servicios activos	Ubicación reconocida por servicio	Resultado
E1	2	2	1	Si	Si
E2	2	1	1	Si	Si

Tabla 33: Escenarios Requisito básico 2, Historia de Usuario 4, Subhistoria 2

Análisis

El escenario E1 podría subdividirse en al menos dos pruebas de aceptación diferentes. Una *naive* que mientras exista respuesta de clima asume que ha funcionado, otra *sanity* que realice lo mismo pero además comprueba el nombre de los campos y su tipado.

El escenario E2 podría subdividirse en al menos dos pruebas de aceptación diferentes. Una *naive* que mientras exista respuesta de clima asume que ha funcionado, otra *sanity* que realice lo mismo pero además comprueba el nombre de los campos y su tipado.

Pruebas de aceptación

	E1 (válido 1)	**E2 (válido 2)**
Given	▪ Un usuario registrado ▪ Tiene dos ubicaciones guardadas ('Castellón' y 'Valencia') ▪ De ellas todas están activadas ▪ API dispone de un servicio ('Clima')	▪ Un usuario registrado ▪ No tiene ninguna ubicación guardada ▪ ('Antartica' y 'Valencia') ▪ La ubicación de 'Antártica' esté activa ▪ API dispone de un servicio ('Clima')
When	Cuando solicita a la API la información sobre los servicios de la ubicación 'Antártica'	Cuando solicita a la API la información sobre los servicios de la ubicación 'Castellón'
Then	La API deberá devolviera información del clima de 'Antática'	La API devolverá un paquete vacío

Tabla 34: Pruebas de aceptación, Requisito básico 2, Historia de Usuario 4, Subhistoria 2

Tests de integración y aceptación

Los siguientes tests están en el repositorio `app` del proyecto y se encuentran en las rutas:

`src/test/java/app/test/acceptance/basic/requirement02/history04/Subhistory02.java`

`src/test/java/app/test/integration/basic/requirement02/history04/Subhistory02.java`

Fragmento de código 33: Test de aceptación del requisito basico 2 → HU04 y SH02

```
15 public class Subhistory02 extends SessionTest {
16     @Test
17     public void valid1() {
18         // Given
19         var type = ServiceType.WEATHER.name();
20         client.service.enableService(type);
```

```
21
22          var name = "Castellon";
23          client.location.addLocation(name);
24
25          name = "Madrid";
26          var location = client.location.addLocation(name);
27          var coords = location.extract().jsonPath().getString("coords");
28
29          // When
30          var response = client.service.getServicesForLocation(coords);
31
32          // Then
33          response.statusCode(HttpStatus.OK.value());
34          response.body("", hasSize(1));
35          response.body(setupServiceQuery(type, "data.temp"), instanceOf(Number.
                class));
36          response.body(setupServiceQuery(type, "data.rain"), instanceOf(Number.
                class));
37          response.body(setupServiceQuery(type, "data.wind"), instanceOf(Number.
                class));
38          response.body(setupServiceQuery(type, "data.icon"), instanceOf(String.
                class));
39          response.body(setupServiceQuery(type, "data.description"), instanceOf(
                String.class));
40      }
41
42      @Test
43      public void valid2() {
44          // Given
45          var type = ServiceType.WEATHER.name();
46          client.service.enableService(type);
47
48          var name = "Madrid";
49          var location = client.location.addLocation(name);
50          var coords = location.extract().jsonPath().getString("coords");
51          client.location.removeLocation(coords);
52
53          var alias = "Antarctica";
54          coords = "-78.159,16.406";
55          location = client.location.addLocation(coords, alias);
56          coords = location.extract().jsonPath().getString("coords");
57
58          // When
59          var response = client.service.getServicesForLocation(coords);
60
61          // Then
62          response.statusCode(HttpStatus.OK.value());
63          response.body("", hasSize(1));
64          response.body(setupServiceQuery(type, "data.temp"), instanceOf(Number.
                class));
65          response.body(setupServiceQuery(type, "data.rain"), instanceOf(Number.
                class));
66          response.body(setupServiceQuery(type, "data.wind"), instanceOf(Number.
                class));
67          response.body(setupServiceQuery(type, "data.icon"), instanceOf(String.
                class));
68          response.body(setupServiceQuery(type, "data.description"), instanceOf(
```

```
                String.class));
69      }
70  }
```

Fragmento de código 34: Test de integración del requisito basico 2 → HU04 y SH02

```
17  public class Subhistory02 extends SessionTest {
18      @Test
19      public void valid1() {
20          // Given
21          var type = ServiceType.WEATHER.name();
22          client.service.enableService(type);
23
24          var name = "Valencia";
25          var locationMock = new LocationModel(name, 39.503, -0.405);
26          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
27          client.location.addLocation(name);
28
29          name = "Castellon";
30          locationMock = new LocationModel(name, 39.980, -0.033);
31          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
32          var location = client.location.addLocation(name);
33          var coords = location.extract().jsonPath().getString("coords");
34          Mockito.doReturn(true).when(spy.weatherService).getData(locationMock);
35
36          // When
37          var response = client.service.getServicesForLocation(coords);
38
39          // Then
40          response.statusCode(HttpStatus.OK.value());
41          response.body("", hasSize(1));
42          response.body(setupServiceQuery(type, "data"), equalTo(true));
43      }
44
45      @Test
46      public void valid2() {
47          // Given
48          var type = ServiceType.WEATHER.name();
49          client.service.enableService(type);
50
51          var name = "Valencia";
52          var locationMock = new LocationModel(name, 39.503, -0.405);
53          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
54          var location = client.location.addLocation(name);
55          var coords = location.extract().jsonPath().getString("coords");
56          client.location.removeLocation(coords);
57
58          var alias = "Antarctica";
59          locationMock = new LocationModel(name, -78.159, 16.406);
60          coords = locationMock.getCoords();
61          Mockito.doReturn(locationMock).when(spy.queryManager).getData(coords);
62          location = client.location.addLocation(coords, alias);
63          coords = location.extract().jsonPath().getString("coords");
64          Mockito.doReturn(true).when(spy.weatherService).getData(locationMock);
65
66          // When
```

```
67          var response = client.service.getServicesForLocation(coords);
68
69          // Then
70          response.statusCode(HttpStatus.OK.value());
71          response.body("", hasSize(1));
72          response.body(setupServiceQuery(type, "data"), equalTo(true));
73      }
74  }
```

Requisito básico 2, Historia de usuario 4, Subhistoria 3

Como usuario quiero consultar fácilmente la información de los eventos sobre un una ubicación activa

Escenarios

Escenario	Cantidad de ubicaciones	Cantidad de ubicaciones activas	Cantidad de servicios activos	Ubicación reconocida por servicio	Resultado
E1	2	2	1	Si	Si
E2	2	0	1	No	No

Tabla 35: Escenarios Requisito básico 2, Historia de Usuario 4, Subhistoria 3

Análisis

El escenario E1 podría subdividirse en al menos dos pruebas de aceptación diferentes. Una *naive* que mientras exista respuesta de eventos asume que ha funcionado, otra *sanity* que realice lo mismo pero además comprueba el nombre de los campos y su tipado.

El escenario E2 podría subdividirse en al menos dos pruebas de aceptación diferentes. Una que consulte una ubicación no reconocida y ésta no responda, y otra que realice la prueba desconectado de la red y esta tampoco responda.

Pruebas de aceptación

	E1 (válido)	**E2 (inválido)**
Given	▪ Un usuario registrado ▪ Tiene tres ubicaciones guardadas ('Castellón' y 'Valencia') ▪ De ellas todas están activadas ▪ API dispone de un servicio ('Eventos')	▪ Un usuario registrado ▪ Tiene tres ubicaciones guardadas ('Antartica' y 'Valencia') ▪ La ubicación 'Antártica' está activa ▪ API dispone de un servicio ('Eventos')
When	Cuando solicita a la API la información sobre los servicios de la ubicación 'Castellón'	Cuando solicita a la API la información sobre los servicios de la ubicación 'Antártica'
Then	La API deberá devolviera información de los eventos de 'Castellón'	La API devolverá un paquete vacío

Tabla 36: Pruebas de aceptación, Requisito básico 2, Historia de Usuario 4, Subhistoria 3

Tests de integración y aceptación

Los siguientes tests están en el repositorio `app` del proyecto y se encuentran en las rutas:

`src/test/java/app/test/acceptance/basic/requirement02/history04/Subhistory03.java`

`src/test/java/app/test/integration/basic/requirement02/history04/Subhistory03.java`

Fragmento de código 35: Test de aceptación del requisito basico 2 → HU04 y SH03

```java
public class Subhistory03 extends SessionTest {
    @Test
    public void valid() {
        // Given
        var type = ServiceType.EVENTS.name();
        client.service.enableService(type);

        var name = "Castellon";
        client.location.addLocation(name);

        name = "Madrid";
        var location = client.location.addLocation(name);
        var coords = location.extract().jsonPath().getString("coords");

        // When
        var response = client.service.getServicesForLocation(coords);

        // Then
        response.statusCode(HttpStatus.OK.value());
        response.body("", hasSize(1));
```

```
            response.body(setupServiceQuery(type, "data"), hasSize(greaterThan(0)))
                ;
            response.body(setupServiceQuery(type, "data.title"), everyItem(
                instanceOf(String.class)));
            response.body(setupServiceQuery(type, "data.date"), everyItem(
                instanceOf(String.class)));
            response.body(setupServiceQuery(type, "data.url"), everyItem(instanceOf
                (String.class)));
            response.body(setupServiceQuery(type, "data.author"), everyItem(
                instanceOf(String.class)));
            response.body(setupServiceQuery(type, "data.image"), everyItem(
                instanceOf(String.class)));
            response.body(setupServiceQuery(type, "data.price"), everyItem(anyOf(
                instanceOf(Number.class), equalTo(false))));
            response.body(setupServiceQuery(type, "data.location"), everyItem(
                instanceOf(String.class)));
        }

        @Test
        public void invalid() {
            // Given
            var type = ServiceType.EVENTS.name();
            client.service.enableService(type);

            var name = "Madrid";
            client.location.addLocation(name);

            var alias = "Antarctica";
            var coords = "-78.159,16.406";
            var location = client.location.addLocation(coords, alias);
            coords = location.extract().jsonPath().getString("coords");

            // When
            var response = client.service.getServicesForLocation(coords);

            // Then
            response.statusCode(HttpStatus.OK.value());
            response.body("", hasSize(1));
            response.body(setupServiceQuery(type, "data"), hasSize(0));
        }
    }
```

Fragmento de código 36: Test de integración del requisito basico 2 → HU04 y SH03

```
    public class Subhistory03 extends SessionTest {
        @Test
        public void valid() {
            // Given
            var type = ServiceType.EVENTS.name();
            client.service.enableService(type);

            var name = "Valencia";
            var locationMock = new LocationModel(name, 39.503, -0.405);
            Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
            client.location.addLocation(name);

```

```
31            name = "Castellon";
32            locationMock = new LocationModel(name, 39.980, -0.033);
33            Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
34            var location = client.location.addLocation(name);
35            var coords = location.extract().jsonPath().getString("coords");
36            Mockito.doReturn(true).when(spy.eventsService).getData(locationMock);
37
38            // When
39            var response = client.service.getServicesForLocation(coords);
40
41            // Then
42            response.statusCode(HttpStatus.OK.value());
43            response.body("", hasSize(1));
44            response.body(setupServiceQuery(type, "data"), equalTo(true));
45        }
46
47        @Test
48        public void invalid() {
49            // Given
50            var type = ServiceType.EVENTS.name();
51            client.service.enableService(type);
52
53            var name = "Valencia";
54            var locationMock = new LocationModel(name, 39.503, -0.405);
55            Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
56            client.location.addLocation(name);
57
58            var alias = "Antarctica";
59            locationMock = new LocationModel(name, -78.159, 16.406);
60            var coords = locationMock.getCoords();
61            Mockito.doReturn(locationMock).when(spy.queryManager).getData(coords);
62            var location = client.location.addLocation(coords, alias);
63            coords = location.extract().jsonPath().getString("coords");
64            Mockito.doReturn(Collections.emptyList()).when(spy.eventsService).
                  getData(locationMock);
65
66            // When
67            var response = client.service.getServicesForLocation(coords);
68
69            // Then
70            response.statusCode(HttpStatus.OK.value());
71            response.body("", hasSize(1));
72            response.body(setupServiceQuery(type, "data"), hasSize(0));
73        }
74    }
```

Requisito básico 2, Historia de usuario 4, Subhistoria 4

Como usuario quiero consultar fácilmente la información de las noticias sobre un una ubicación activa

Escenarios

Escenario	Cantidad de ubicaciones	Cantidad de ubicaciones activas	Cantidad de servicios activos	Ubicación reconocida por servicio	Resultado
E1	2	**2**	**1**	**Si**	**Si**
E2	2	1	1	No	No

Tabla 37: Escenarios Requisito básico 2, Historia de Usuario 4, Subhistoria 4

Análisis

El escenario E1 podría subdividirse en al menos dos pruebas de aceptación diferentes. Una *naive* que mientras exista respuesta de noticias asume que ha funcionado, otra *sanity* que realice lo mismo pero además comprueba el nombre de los campos y su tipado.

El escenario E2 podría subdividirse en al menos dos pruebas de aceptación diferentes. Una que consulte una ubicación no reconocida y ésta no responda, y otra que realice la prueba desconectado de la red y esta tampoco responda.

Pruebas de aceptación

	E1 (válido)	**E2 (inválido)**
Given	▪ Un usuario registrado ▪ Tiene dos ubicaciones guardadas ('Castellón' y 'Valencia') ▪ De ellas todas están activadas ▪ API dispone de un servicio ('Noticias')	▪ Un usuario registrado ▪ Tiene tres ubicaciones guardadas ▪ La ubicación 'Antártica' está activa ▪ API dispone de un servicio ('Noticias')
When	Cuando solicita a la API la información sobre los servicios de la ubicación 'Castellón'	Cuando solicita a la API la información sobre los servicios de la ubicación 'Antártica'
Then	La API deberá devolviera información de las noticias de 'Castellón'	La API devolverá un paquete vacío

Tabla 38: Pruebas de aceptación, Requisito básico 2, Historia de Usuario 4, Subhistoria 4

Tests de integración y aceptación

Los siguientes tests están en el repositorio app del proyecto y se encuentran en las rutas:

- src/test/java/app/test/acceptance/basic/requirement02/history04/Subhistory04.java
- src/test/java/app/test/integration/basic/requirement02/history04/Subhistory04.java

Fragmento de código 37: Test de aceptación del requisito basico 2 → HU04 y SH04

```
20  public class Subhistory04 extends SessionTest {
21      @Test
22      public void valid() {
23          // Given
24          var type = ServiceType.NEWS.name();
25          client.service.enableService(type);
26
27          var name = "Castellon";
28          client.location.addLocation(name);
29
30          name = "Madrid";
31          var location = client.location.addLocation(name);
32          var coords = location.extract().jsonPath().getString("coords");
33
34          // When
35          var response = client.service.getServicesForLocation(coords);
36
37          // Then
38          response.statusCode(HttpStatus.OK.value());
39          response.body("", hasSize(1));
40          response.body(setupServiceQuery(type, "data"), hasSize(greaterThan(0))
                ;
41          response.body(setupServiceQuery(type, "data.title"), everyItem(
                instanceOf(String.class)));
42          response.body(setupServiceQuery(type, "data.description"), everyItem(
                instanceOf(String.class)));
43          response.body(setupServiceQuery(type, "data.url"), everyItem(instanceOf
                (String.class)));
44          response.body(setupServiceQuery(type, "data.author"), everyItem(
                instanceOf(String.class)));
45          response.body(setupServiceQuery(type, "data.image"), everyItem(anyOf(
                instanceOf(String.class), equalTo(false))));
46      }
47
48      @Test
49      public void invalid() {
50          // Given
51          var type = ServiceType.NEWS.name();
52          client.service.enableService(type);
53
54          var name = "Madrid";
55          client.location.addLocation(name);
56
```

```
57          var alias = "Antarctica";
58          var coords = "-78.159,16.406";
59          var location = client.location.addLocation(coords, alias);
60          coords = location.extract().jsonPath().getString("coords");
61
62          // When
63          var response = client.service.getServicesForLocation(coords);
64
65          // Then
66          response.statusCode(HttpStatus.OK.value());
67          response.body("", hasSize(1));
68          response.body(setupServiceQuery(type, "data"), hasSize(0));
69      }
70  }
```

Fragmento de código 38: Test de integración del requisito basico 2 → HU04 y SH04

```
19  public class Subhistory04 extends SessionTest {
20      @Test
21      public void valid() {
22          // Given
23          var type = ServiceType.NEWS.name();
24          client.service.enableService(type);
25
26          var name = "Valencia";
27          var locationMock = new LocationModel(name, 39.503, -0.405);
28          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
29          client.location.addLocation(name);
30
31          name = "Castellon";
32          locationMock = new LocationModel(name, 39.980, -0.033);
33          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
34          var location = client.location.addLocation(name);
35          var coords = location.extract().jsonPath().getString("coords");
36          Mockito.doReturn(true).when(spy.newsService).getData(locationMock);
37
38          // When
39          var response = client.service.getServicesForLocation(coords);
40
41          // Then
42          response.statusCode(HttpStatus.OK.value());
43          response.body("", hasSize(1));
44          response.body(setupServiceQuery(type, "data"), equalTo(true));
45      }
46
47      @Test
48      public void invalid() {
49          // Given
50          var type = ServiceType.NEWS.name();
51          client.service.enableService(type);
52
53          var name = "Valencia";
54          var locationMock = new LocationModel(name, 39.503, -0.405);
55          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
56          client.location.addLocation(name);
57
```

```
58          var alias = "Antarctica";
59          locationMock = new LocationModel(name, -78.159, 16.406);
60          var coords = locationMock.getCoords();
61          Mockito.doReturn(locationMock).when(spy.queryManager).getData(coords);
62          var location = client.location.addLocation(coords, alias);
63          coords = location.extract().jsonPath().getString("coords");
64          Mockito.doReturn(Collections.emptyList()).when(spy.newsService).getData
                (locationMock);
65
66          // When
67          var response = client.service.getServicesForLocation(coords);
68
69          // Then
70          response.statusCode(HttpStatus.OK.value());
71          response.body("", hasSize(1));
72          response.body(setupServiceQuery(type, "data"), hasSize(0));
73      }
74  }
```

1.1.2.5 Requisito básico 2, Historia de usuario 5

Como usuario quiero consultar el histórico de ubicaciones, con el fin de facilitar la reactivación de alguna en caso de necesidad.

Escenarios

Escenario	Cantidad de ubicaciones	Cantidad de ubicaciones activas	Resultado
E1	2	1	Si
E2	2	0	No

Tabla 39: Escenarios Requisito básico 2, Historia de Usuario 5

Análisis

El escenario E1 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería consultar el histórico y comprobar que hay solo una ubicación.

El escenario E2 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería consultar el histórico y comprobar que no hay ninguna ubicación.

Pruebas de aceptación

	E1 (válido)	E2 (inválido)
Given	▪ Un usuario registrado ▪ Tiene dos ubicaciones guardadas ('Castellón' y 'Valencia') ▪ De ellas una está activadas ('Castellón')	▪ Un usuario registrado ▪ No tiene tres ubicaciones guardadas ▪ ('Castellano' y 'Valencia') ▪ Todas ellas activadas ('Castellón', 'Valencia')
When	Cuando solicita a la API la información sobre sus ubicaciones desactivadas	Cuando solicita a la API la información sobre sus ubicaciones desactivadas
Then	La API deberá devolver un paquete de información por cada ubicación desactivada. / En este caso: ('Valencia')	La API devolverá un paquete vacío

Tabla 40: Pruebas de aceptación, Requisito básico 2, Historia de Usuario 5

Tests de integración y aceptación

Los siguientes tests están en el repositorio `app` del proyecto y se encuentran en las rutas:

`src/test/java/app/test/acceptance/basic/requirement02/History05.java`

`src/test/java/app/test/integration/basic/requirement02/History05.java`

Fragmento de código 39: Test de aceptación del requisito basico 2 → HU05

```
13  public class History05 extends SessionTest {
14      @Test
15      public void valid() {
16          // Given
17          var name = "Castellon";
18          client.location.addLocation(name);
19
20          name = "Valencia";
21          var location = client.location.addLocation(name);
22          var coords = location.extract().jsonPath().getString("coords");
23          client.location.removeLocation(coords);
24
25          // When
26          var response = client.history.getLocations();
27
28          // Then
29          response.statusCode(HttpStatus.OK.value());
30          response.body("", hasSize(1));
31          response.body("name", hasItem(name));
32      }
33
```

```
34      @Test
35      public void invalid() {
36          // Given
37          var name = "Castellon";
38          client.location.addLocation(name);
39
40          name = "Valencia";
41          client.location.addLocation(name);
42
43          // When
44          var response = client.history.getLocations();
45
46          // Then
47          response.statusCode(HttpStatus.OK.value());
48          response.body("", hasSize(0));
49      }
50  }
```

Fragmento de código 40: Test de integración del requisito basico 2 → HU05

```
15  public class History05 extends SessionTest {
16      @Test
17      public void valid() {
18          // Given
19          var name = "Castellon";
20          var locationMock = new LocationModel(name, 39.980, -0.033);
21          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
22          client.location.addLocation(name);
23
24          name = "Valencia";
25          locationMock = new LocationModel(name, 39.503, -0.405);
26          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
27          var location = client.location.addLocation(name);
28          var coords = location.extract().jsonPath().getString("coords");
29          client.location.removeLocation(coords);
30
31          // When
32          var response = client.history.getLocations();
33
34          // Then
35          response.statusCode(HttpStatus.OK.value());
36          response.body("", hasSize(1));
37          response.body("name", hasItem(name));
38      }
39
40      @Test
41      public void invalid() {
42          // Given
43          var name = "Castellon";
44          var locationMock = new LocationModel(name, 39.980, -0.033);
45          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
46          client.location.addLocation(name);
47
48          name = "Valencia";
49          locationMock = new LocationModel(name, 39.503, -0.405);
50          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
```

```
51            client.location.addLocation(name);
52
53            // When
54            var response = client.history.getLocations();
55
56            // Then
57            response.statusCode(HttpStatus.OK.value());
58            response.body("", hasSize(0));
59        }
60    }
```

1.1.3. Requisito básico 3 - Activar servicios API a partir de una lista

El tercer requisito básico de esta aplicación trata de seleccionar (activar) servicios API a partir de un listado de servicios externos disponibles.

1.1.3.1 Requisito básico 3, Historia de usuario 1

Como usuario quiero consultar la lista de servicios de información disponibles (API), con el fin de elegir (activar) aquellos de interés.

Escenarios

Escenario	**Servicios disponibles**	**Respuesta**
E1	1	Si
E2	0	No

Tabla 41: Escenarios Requisito básico 3, Historia de Usuario 1

Análisis

El escenario E1 podría subdividirse en al menos dos pruebas de aceptación diferentes. Una *naive* que mientras exista asume que el servicio está disponible, otra *sanity* que realice una petición de prueba para ver si el servicio funciona como se espera.

El escenario E2 podría subdividirse en al menos dos pruebas de aceptación diferentes. Una *naive* que mientras no exista asume que el servicio está no disponible, otra *sanity* que realice una petición de prueba para ver si el servicio funciona mientras está desconectado de la red.

Pruebas de aceptación

	E1 (válido)	**E2 (inválido)**
Given	▪ Un usuario registrado ▪ Un servicio disponible ('Clima')	▪ Un usuario registrado ▪ Ningún servicio disponible
When	Un usuario solicita una lista de los servicios disponibles	Un usuario solicita una lista de los servicios disponibles
Then	Devuelve un listado con los servicios disponibles ('Clima')	Devuelve un listado vacío

Tabla 42: Pruebas de aceptación, Requisito básico 3, Historia de Usuario 1

Tests de integración y aceptación

Los siguientes tests están en el repositorio `app` del proyecto y se encuentran en las rutas:

`src/test/java/app/test/acceptance/basic/requirement03/History01.java`

`src/test/java/app/test/integration/basic/requirement03/History01.java`

Fragmento de código 41: Test de aceptación del requisito basico 3 → HU01

```
14  public class History01 extends SessionTest {
15      @Test
16      public void valid() {
17          // Given
18          var type = ServiceType.WEATHER.name();
19          client.service.enableAllServices();
20          client.service.disableService(type);
21
22          // When
23          var response = client.service.getServices();
24
25          // Then
26          response.statusCode(HttpStatus.OK.value());
27          response.body(setupEnabledQuery(false, ""), hasSize(1));
28          response.body(setupEnabledQuery(false, "service.type"), hasItem(type));
29      }
30
31      @Test
32      public void invalid() {
33          // Given
34          client.service.enableAllServices();
35
36          // When
37          var response = client.service.getServices();
38
39          // Then
40          response.statusCode(HttpStatus.OK.value());
41          response.body(setupEnabledQuery(false, ""), hasSize(0));
42      }
43  }
```

Fragmento de código 42: Test de integración del requisito basico 3 → HU01

```
14  public class History01 extends SessionTest {
15      @Test
16      public void valid() {
17          // Given
18          var type = ServiceType.WEATHER.name();
19          client.service.enableAllServices();
20          client.service.disableService(type);
21
22          // When
23          var response = client.service.getServices();
24
25          // Then
26          response.statusCode(HttpStatus.OK.value());
27          response.body(setupEnabledQuery(false, ""), hasSize(1));
28          response.body(setupEnabledQuery(false, "service.type"), hasItem(type));
29      }
30
31      @Test
32      public void invalid() {
33          // Given
34          client.service.enableAllServices();
35
```

```
36          // When
37          var response = client.service.getServices();
38
39          // Then
40          response.statusCode(HttpStatus.OK.value());
41          response.body(setupEnabledQuery(false, ""), hasSize(0));
42      }
43  }
```

1.1.3.2 Requisito básico 3, Historia de usuario 2

Como usuario quiero activar un servicio de información (API), entre aquellos disponibles.

Escenarios

Escenario	Servicios disponibles	Servicios activados previas	Servicios activados después	BBDD modificada
E1	1	0	1	Si
E2	1	0	0	No

Tabla 43: Escenarios Requisito básico 3, Historia de Usuario 2

Análisis

El escenario E1 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería activar un servicio desactivado y este activarse.

El escenario E2 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería intentar activar un servicio invalido y este no activarse.

Pruebas de aceptación

	E1(válido)	E2 (inválido)
Given	▪ Un usuario registrado ▪ Un servicio disponible 'Clima' ▪ Ningún servicio activado	▪ Un usuario registrado ▪ Un servicio disponible 'Clima' ▪ Ningún servicio activado
When	Una usuario solicita activar el servicio 'Clima'	Una usuario solicita activar el servicio 'INVALIDO'
Then	Un servicio activado 'Clima'	Ningún servicio activado

Tabla 44: Pruebas de aceptación, Requisito básico 3, Historia de Usuario 2

Tests de integración y aceptación

Los siguientes tests están en el repositorio app del proyecto y se encuentran en las rutas:

- src/test/java/app/test/acceptance/basic/requirement03/History02.java
- src/test/java/app/test/integration/basic/requirement03/History02.java

Fragmento de código 43: Test de aceptación del requisito basico 3 → HU02

```
17  public class History02 extends SessionTest {
18      @Test
19      public void valid() {
20          // Given
21          var type = ServiceType.WEATHER.name();
22          Mockito.reset(spy.accountManager);
23
24          // When
25          var response = client.service.enableService(type);
26
27          // Then
28          Mockito.verify(spy.accountManager).saveAccount(any());
29          response.statusCode(HttpStatus.OK.value());
30          var state = client.service.getServices();
31          state.body(setupEnabledQuery(true, ""), hasSize(1));
32          state.body(setupEnabledQuery(true, "service.type"), hasItem(type));
33      }
34
35      @Test
36      public void invalid() {
37          // Given
38          var type = "INVALIDO";
39          Mockito.reset(spy.accountManager);
40
41          // When
42          var response = client.service.enableService(type);
43
44          // Then
45          Mockito.verify(spy.accountManager, never()).saveAccount(any());
46          response.statusCode(HttpStatus.BAD_REQUEST.value());
47          var state = client.service.getServices();
48          state.body(setupEnabledQuery(true, ""), hasSize(0));
49      }
50  }
```

Fragmento de código 44: Test de integración del requisito basico 3 → HU02

```
17  public class History02 extends SessionTest {
18      @Test
19      public void valid() {
20          // Given
21          var type = ServiceType.WEATHER.name();
22          Mockito.reset(spy.accountManager);
23
24          // When
```

```
25          var response = client.service.enableService(type);
26
27          // Then
28          Mockito.verify(spy.accountManager).saveAccount(any());
29          response.statusCode(HttpStatus.OK.value());
30          var state = client.service.getServices();
31          state.body(setupEnabledQuery(true, ""), hasSize(1));
32          state.body(setupEnabledQuery(true, "service.type"), hasItem(type));
33      }
34
35      @Test
36      public void invalid() {
37          // Given
38          var type = "INVALIDO";
39          Mockito.reset(spy.accountManager);
40
41          // When
42          var response = client.service.enableService(type);
43
44          // Then
45          Mockito.verify(spy.accountManager, never()).saveAccount(any());
46          response.statusCode(HttpStatus.BAD_REQUEST.value());
47          var state = client.service.getServices();
48          state.body(setupEnabledQuery(true, ""), hasSize(0));
49      }
50  }
```

1.1.3.3 Requisito básico 3, Historia de usuario 3

Como usuario quiero conocer una breve descripción de cada fuente de información disponible (e.g. perfil de información, frecuencia de actualización, etc.), para poder tomar decisiones fundamentadas.

Escenarios

Escenario	Servicios disponibles	API disponibles	Servicio valido	Respuesta
E1	1	**Si**	**Si**	**Si**
E2	1	Si	No	No

Tabla 45: Escenarios Requisito básico 3, Historia de Usuario 3

Análisis

El escenario E1 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería consultar un servicio válido disponible y comprobar que devuelve datos.

El escenario E2 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros

del escenario sería consultar un servicio inválido y comprobar que no devuelve datos.

Pruebas de aceptación

	E1 (válido)	**E2 (inválido)**
Given	▪ Un usuario registrado ▪ Un servicio disponible 'Clima'	▪ Un usuario registrado ▪ Un servicio disponible 'Clima'
When	Se solicita la información sobre el servicio 'Clima'	Se solicita la información sobre el servicio 'INVALIDO'
Then	Devuelve nombre y descripción del servicio del 'Clima'	No devuelve información relevante

Tabla 46: Pruebas de aceptación, Requisito básico 3, Historia de Usuario 3

Tests de integración y aceptación

Los siguientes tests están en el repositorio `app` del proyecto y se encuentran en las rutas:

`src/test/java/app/test/acceptance/basic/requirement03/History03.java`

`src/test/java/app/test/integration/basic/requirement03/History03.java`

Fragmento de código 45: Test de aceptación del requisito basico 3 → HU03

```
16  public class History03 extends SessionTest {
17      @Test
18      public void valid() {
19          // Given
20          var type = ServiceType.WEATHER.name();
21
22          // When
23          var response = client.service.getServices();
24
25          // Then
26          response.statusCode(HttpStatus.OK.value());
27          response.body(setupServiceQuery(type, "service"), equalTo(Map.of("type"
                , type, "name", spy.weatherService.getName(), "description", spy.
                weatherService.getDescription())));
28      }
29
30      @Test
31      public void invalid() {
32          // Given
33          var type = "INVALIDO";
34
35          // When
36          var response = client.service.getServices();
37
38          // Then
39          response.statusCode(HttpStatus.OK.value());
40          response.body(setupServiceQuery(type, "service"), nullValue());
```

```
    }
}
```

Fragmento de código 46: Test de integración del requisito basico 3 → HU03

```
public class History03 extends SessionTest {
    @Test
    public void valid() {
        // Given
        var type = ServiceType.WEATHER.name();

        // When
        var response = client.service.getServices();

        // Then
        response.statusCode(HttpStatus.OK.value());
        response.body(setupServiceQuery(type, "service"), equalTo(Map.of("type"
            , type, "name", spy.weatherService.getName(), "description", spy.
            weatherService.getDescription())));
    }

    @Test
    public void invalid() {
        // Given
        var type = "INVALIDO";

        // When
        var response = client.service.getServices();

        // Then
        response.statusCode(HttpStatus.OK.value());
        response.body(setupServiceQuery(type, "service"), nullValue());
    }
}
```

1.1.3.4 Requisito básico 3, Historia de usuario 4

Como usuario quiero desactivar un servicio de información que haya dejado de interesar, con el fin de evitar interfaces de usuario sobrecargadas.

Escenarios

Escenario	Servicios disponibles	Servicios activados previas	Servicios activados después	BBDD modificada
E1	1	**1**	**0**	**Si**
E2	1	1	1	No

Tabla 47: Escenarios Requisito básico 3, Historia de Usuario 4

Análisis

El escenario E1 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería desactivar un servicio activado y este desactivarse.

El escenario E2 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería intentar desactivar un servicio inválido y este no cambiar el estado.

Pruebas de aceptación

	E1(válido)	**E2 (inválido)**
Given	▪ Un usuario registrado ▪ Un servicio disponible 'Clima' ▪ Un servicio activado 'Clima'	▪ Un usuario registrado ▪ Un servicio disponible 'Clima' ▪ Un servicio activado 'Clima'
When	Una usuario solicita desactivar el servicio 'Clima'	Una usuario solicita desactivar el servicio 'INVALIDO'
Then	No hay ningún servicio activo	No se ha cambiado ningún estado

Tabla 48: Pruebas de aceptación, Requisito básico 3, Historia de Usuario 4

Tests de integración y aceptación

Los siguientes tests están en el repositorio `app` del proyecto y se encuentran en las rutas:

`src/test/java/app/test/acceptance/basic/requirement03/History04.java`

`src/test/java/app/test/integration/basic/requirement03/History04.java`

Fragmento de código 47: Test de aceptación del requisito basico 3 → HU04

```
17  public class History04 extends SessionTest {
18      @Test
19      public void valid() {
20          // Given
21          var type = ServiceType.WEATHER.name();
22          client.service.enableService(type);
23          Mockito.reset(spy.accountManager);
24
25          // When
26          var response = client.service.disableService(type);
27
28          // Then
29          Mockito.verify(spy.accountManager).saveAccount(any());
30          response.statusCode(HttpStatus.OK.value());
31          var state = client.service.getServices();
32          state.body(setupEnabledQuery(true, ""), hasSize(0));
33      }
34
```

```
35      @Test
36      public void invalid() {
37          // Given
38          var typeA = ServiceType.WEATHER.name();
39          client.service.enableService(typeA);
40          var typeB = "INVALIDO";
41          Mockito.reset(spy.accountManager);
42
43          // When
44          var response = client.service.enableService(typeB);
45
46          // Then
47          Mockito.verify(spy.accountManager, never()).saveAccount(any());
48          response.statusCode(HttpStatus.BAD_REQUEST.value());
49          var state = client.service.getServices();
50          state.body(setupEnabledQuery(true, ""), hasSize(1));
51          state.body(setupEnabledQuery(true, "service.type"), hasItem(typeA));
52      }
53  }
```

Fragmento de código 48: Test de integración del requisito basico 3 → HU04

```
17  public class History04 extends SessionTest {
18      @Test
19      public void valid() {
20          // Given
21          var type = ServiceType.WEATHER.name();
22          client.service.enableService(type);
23          Mockito.reset(spy.accountManager);
24
25          // When
26          var response = client.service.disableService(type);
27
28          // Then
29          Mockito.verify(spy.accountManager).saveAccount(any());
30          response.statusCode(HttpStatus.OK.value());
31          var state = client.service.getServices();
32          state.body(setupEnabledQuery(true, ""), hasSize(0));
33      }
34
35      @Test
36      public void invalid() {
37          // Given
38          var typeA = ServiceType.WEATHER.name();
39          client.service.enableService(typeA);
40          var typeB = "INVALIDO";
41          Mockito.reset(spy.accountManager);
42
43          // When
44          var response = client.service.enableService(typeB);
45
46          // Then
47          Mockito.verify(spy.accountManager, never()).saveAccount(any());
48          response.statusCode(HttpStatus.BAD_REQUEST.value());
49          var state = client.service.getServices();
50          state.body(setupEnabledQuery(true, ""), hasSize(1));
```

```
            state.body(setupEnabledQuery(true, "service.type"), hasItem(typeA));
        }
    }
```

1.1.4. Requisito básico 4 - Recuperar el último estado de la aplicación

El cuarto y último requisito básico de esta aplicación consta de recuperar el último estado de la aplicación (e.g. ubicaciones, servicios, suscripciones, etc.) cada vez que se inicializa.

Debido a que utilizamos una base de datos empotrada, en vez de remota, no hay fallos de conexión; por lo tanto tampoco hay casos inválidos, tan solo puede fallar las abstracciones o otras clases que dependan de ella.

1.1.4.1 Requisito básico 4, Historia de usuario 1

Como usuario quiero que cada vez que inicie la aplicación, sus contenidos y aspecto sean idénticos a los que había la última vez que se cerró, con el fin de evitar reconfigurarla en cada uso.

Escenarios

Escenarios	**Primer inicio**	**Configuración matenida**	**BBDD modificada**
E1	Si	No	Si
E2	No	Si	No

Tabla 49: Escenarios Requisito básico 4, Historia de Usuario 1

Análisis

El escenario E1 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería crear una cuenta y esta crearse modificando la base de datos.

El escenario E2 podría subdividirse en al menos dos pruebas de aceptación diferentes. Una usando un usuario registrado sin ningún dato adicional, otra usando un usuario registrado con múltiples datos adicionales (como ubicaciones, histórico, servicios...).

Pruebas de aceptación

	E1 (válido)	**E2 (válido)**
Given	▪ Un usuario no registrado ▪ Ninguna ubiación guardada ▪ Ningún servicio añadido	▪ Un usuario registrado ▪ Ninguna ubicación guardada ▪ Ningún servicio añadido
When	Se crea la cuenta	Accedemos a la sesión
Then	El usuarios se ha añadido a la base de datos	El usuario sigue igual en la base de datos

Tabla 50: Pruebas de aceptación, Requisito básico 4, Historia de Usuario 1

Tests de integración y aceptación

Los siguientes tests están en el repositorio app del proyecto y se encuentran en las rutas:

- src/test/java/app/test/acceptance/basic/requirement04/History01.java
- src/test/java/app/test/integration/basic/requirement04/History01.java

Fragmento de código 49: Test de aceptación del requisito basico 4 → HU01

```
14  public class History01 extends BaseTest {
15      @Test
16      public void valid1(TestInfo info) {
17          // Given
18          // No account
19          var id = getId(info);
20          Mockito.reset(spy.accountManager);
21
22          // When
23          var response = client.account.register(id, id);
24
25          // Then
26          Mockito.verify(spy.accountManager).saveAccount(any());
27          response.statusCode(HttpStatus.OK.value());
28      }
29
30      @Test
31      public void valid2(TestInfo info) {
32          // Given
33          var id = getId(info);
34          client.account.register(id, id);
35          client.session.logout();
36          Mockito.reset(spy.accountManager);
37
38          // When
39          var response = client.session.login(id, id);
40
41          // Then
42          Mockito.verify(spy.accountManager, never()).saveAccount(any());
43          response.statusCode(HttpStatus.OK.value());
44      }
45  }
```

Fragmento de código 50: Test de integración del requisito basico 4 → HU01

```
14  public class History01 extends BaseTest {
15      @Test
16      public void valid1(TestInfo info) {
17          // Given
18          // No account
19          var id = getId(info);
20          Mockito.reset(spy.accountManager);
21
22          // When
23          var response = client.account.register(id, id);
```

```
24
25          // Then
26          Mockito.verify(spy.accountManager).saveAccount(any());
27          response.statusCode(HttpStatus.OK.value());
28      }
29
30      @Test
31      public void valid2(TestInfo info) {
32          // Given
33          var id = getId(info);
34          client.account.register(id, id);
35          client.session.logout();
36          Mockito.reset(spy.accountManager);
37
38          // When
39          var response = client.session.login(id, id);
40
41          // Then
42          Mockito.verify(spy.accountManager, never()).saveAccount(any());
43          response.statusCode(HttpStatus.OK.value());
44      }
45  }
```

1.1.4.2 Requisito básico 4, Historia de usuario 2

Como usuario quiero que la historia anterior se cumpla aunque el cierre de la aplicación haya sido involuntario (e.g. un corte de luz).

Debido a lo mencionado en el requisito y que esta historia requiere que se guarde después de cada modificación, hemos optado por apuntar lo en cada historia como una columna adicional; si la sus aspectos se han guardados y se restaurarán la próxima vez que se inicie la aplicación, esta está marcada con un 'si'.

1.2. Requisitos avanzados

Además de los requisitos que la guía del proyecto nos ha ofrecido, como grupo hemos pensado que sería interesante dotar de funcionalidades extra a nuestra aplicación mediante algunos requisitos adicionales (de caracter avanzado). Es por ello por lo que consideramos que los siguientes requisitos avanzados son en parte una mejora de la aplicación tanto en usabilidad como en funcionalidad de la misma.

1.2.1. Requisito avanzado 1 - Crear cuentas en la aplicación

Permitir a los usuarios crear cuentas en la aplicación, estas servirán a modo de identificación y mantendrán la información de configuración y ubicaciones guardadas de los usuarios de la aplicación.

1.2.1.1 Requisito avanzado 1, Historia de usuario 1

Como usuario quiero poder crear unas credenciales únicas que sirvan para identificarse en la aplicación.

Escenarios

Escenarios	Usuario válido	Contraseña válida	Cuenta creada	BBDD modificada
E1	Si	Si	Si	Si
E2	Si	No	No	No
E3	No	Si	No	No
E4	No	No	No	No

Tabla 51: Escenarios Requisito avanzado 1, Historia de Usuario 1

Análisis

El escenario E1 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería crear una cuenta con un usuario y contraseña válida, y este registrándose.

El escenario E3 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería crear una cuenta con un usuario ya existente y una contraseña válida, y esté dar error al intentarlo.

Pruebas de aceptación

	E1 (válido)	E3 (inválido)
Given	▪ Usuario no tiene la cuenta con los datos. ▪ Usuario: `id` y Contraseña: `id`	▪ Usuario tiene cuenta con los datos. ▪ Usuario: `id` y Contraseña: `id`
When	Un usuario intenta crear una cuenta	Un usuario intenta crear una cuenta
Then	Se crea una cuenta en el sistema con esas credenciales	No se puede crear la cuenta porque el usuario ya se encuentra registrado en el sistema

Tabla 52: Pruebas de aceptación, Requisito avanzado 1, Historia de Usuario 1

Tests de integración y aceptación

Los siguientes tests están en el repositorio `app` del proyecto y se encuentran en las rutas:

`src/test/java/app/test/acceptance/advanced/requirement01/History01.java`

`src/test/java/app/test/integration/advanced/requirement01/History01.java`

Fragmento de código 51: Test de aceptación del requisito avanzado 1 → HU01

```java
public class History01 extends BaseTest {
    @Override
    @AfterEach
    public void afterEach(TestInfo info) {
        super.afterEach(info);
        client.account.deregister();
    }

    @Test
    public void valid(TestInfo info) {
        // Given
        // No account
        var id = getId(info);
        Mockito.reset(spy.accountManager);

        // When
        var response = client.account.register(id, id);

        // Then
        Mockito.verify(spy.accountManager).saveAccount(any());
        response.statusCode(HttpStatus.OK.value());
        client.session.logout();
        var state = client.session.login(id, id);
        state.statusCode(HttpStatus.OK.value());
    }

    @Test
    public void invalid(TestInfo info) {
```

```
43          // Given
44          var id = getId(info);
45          client.account.register(id, id);
46          client.session.logout();
47          Mockito.reset(spy.accountManager);
48
49          // When
50          var response = client.account.register(id, id);
51
52          // Then
53          Mockito.verify(spy.accountManager, never()).saveAccount(any());
54          response.statusCode(HttpStatus.CONFLICT.value());
55          client.session.logout();
56          var state = client.session.login(id, id);
57          state.statusCode(HttpStatus.OK.value());
58      }
59  }
```

Fragmento de código 52: Test de integración del requisito avanzado 1 → HU01

```
15  public class History01 extends BaseTest {
16      @Override
17      @AfterEach
18      public void afterEach(TestInfo info) {
19          super.afterEach(info);
20          client.account.deregister();
21      }
22
23      @Test
24      public void valid(TestInfo info) {
25          // Given
26          // No account
27          var id = getId(info);
28          Mockito.reset(spy.accountManager);
29
30          // When
31          var response = client.account.register(id, id);
32
33          // Then
34          Mockito.verify(spy.accountManager).saveAccount(any());
35          response.statusCode(HttpStatus.OK.value());
36          client.session.logout();
37          var state = client.session.login(id, id);
38          state.statusCode(HttpStatus.OK.value());
39      }
40
41      @Test
42      public void invalid(TestInfo info) {
43          // Given
44          var id = getId(info);
45          client.account.register(id, id);
46          client.session.logout();
47          Mockito.reset(spy.accountManager);
48
49          // When
50          var response = client.account.register(id, id);
```

```
51
52          // Then
53          Mockito.verify(spy.accountManager, never()).saveAccount(any());
54          response.statusCode(HttpStatus.CONFLICT.value());
55          client.session.logout();
56          var state = client.session.login(id, id);
57          state.statusCode(HttpStatus.OK.value());
58      }
59  }
```

1.2.1.2 Requisito avanzado 1, Historia de usuario 2

Como usuario quiero poder eliminar unas credenciales únicas para que ya no estén disponibles para iniciar esa sesión.

Escenarios

Escenarios	Usuario y con una sesión	Cuenta eliminada	BBDD modificada
E1	Si	Si	Si
E2	No	No	No

Tabla 53: Escenarios Requisito avanzado 1, Historia de Usuario 2

Análisis

El escenario E1 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería eliminar una cuenta teniendo una sesión iniciada, y este darse de baja.

El escenario E2 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería eliminar una cuenta sin tener sesión iniciada, y este dar error al intentarlo.

Pruebas de aceptación

	E1 (válido)	E2 (inválido)
Given	▪ Un usuario con sesión ▪ Unas credenciales correspondientes a una cuenta válida ▪ Usuario: `id` y Contraseña: `id`	▪ Un usuario sin sesión
When	El usuario intenta borrar su cuenta de usuario	El usuario intenta borrar su cuenta de usuario
Then	La cuenta es eliminada y esas credenciales ya no corresponden con una sesión de usuario	El proceso de borrado no se llevará a cabo

Tabla 54: Pruebas de aceptación, Requisito avanzado 1, Historia de Usuario 2

Tests de integración y aceptación

Los siguientes tests están en el repositorio `app` del proyecto y se encuentran en las rutas:

`src/test/java/app/test/acceptance/advanced/requirement01/History02.java`

`src/test/java/app/test/integration/advanced/requirement01/History02.java`

Fragmento de código 53: Test de aceptación del requisito avanzado 1 → HU02

```
14  public class History02 extends BaseTest {
15      @Test
16      public void valid(TestInfo info) {
17          // Given
18          var id = getId(info);
19          client.account.register(id, id);
20          Mockito.reset(spy.accountManager);
21
22          // When
23          var response = client.account.deregister();
24
25          // Then
26          Mockito.verify(spy.accountManager).deleteAccount(any());
27          response.statusCode(HttpStatus.OK.value());
28          var state = client.session.login(id, id);
29          state.statusCode(HttpStatus.UNAUTHORIZED.value());
30      }
31
32      @Test
33      public void invalid(TestInfo info) {
34          // Given
35          var id = getId(info);
36          client.account.register(id, id);
37          client.session.logout();
38          Mockito.reset(spy.accountManager);
39
```

```
        // When
        var response = client.account.deregister();

        // Then
        Mockito.verify(spy.accountManager, never()).deleteAccount(any());
        response.statusCode(HttpStatus.UNAUTHORIZED.value());
        var state = client.session.login(id, id);
        state.statusCode(HttpStatus.OK.value());
    }
}
```

Fragmento de código 54: Test de integración del requisito avanzado 1 → HU02

```
public class History02 extends BaseTest {
    @Test
    public void valid(TestInfo info) {
        // Given
        var id = getId(info);
        client.account.register(id, id);
        Mockito.reset(spy.accountManager);

        // When
        var response = client.account.deregister();

        // Then
        Mockito.verify(spy.accountManager).deleteAccount(any());
        response.statusCode(HttpStatus.OK.value());
        var state = client.session.login(id, id);
        state.statusCode(HttpStatus.UNAUTHORIZED.value());
    }

    @Test
    public void invalid(TestInfo info) {
        // Given
        var id = getId(info);
        client.account.register(id, id);
        client.session.logout();
        Mockito.reset(spy.accountManager);

        // When
        var response = client.account.deregister();

        // Then
        Mockito.verify(spy.accountManager, never()).deleteAccount(any());
        response.statusCode(HttpStatus.UNAUTHORIZED.value());
        var state = client.session.login(id, id);
        state.statusCode(HttpStatus.OK.value());
    }
}
```

1.2.1.3 Requisito avanzado 1, Historia de usuario 3

Como usuario quiero poder cambiar la contraseña de mi cuenta.

Escenarios

Escenarios	**Usuario con una sesión**	**Nueva contraseña válida**	**Cambio de contraseña efectuado**	**BBDD modificada**
E1	Si	Si	Si	Si
E2	Si	No	No	No
E3	No	Si	No	No
E4	No	No	No	No

Tabla 55: Escenarios Requisito avanzado 1, Historia de Usuario 3

Análisis

El escenario E1 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería teniendo sesión y una nueva contraseña válida, está cambiarse.

El escenario E3 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería sin tener sesión ni una contraseña válida, esta no cambiar ningún estado del sistema.

Pruebas de aceptación

	E1 (válido)	**E3 (inválido)**
Given	▪ Un usuario con sesión ▪ Unas credenciales correspondientes a una cuenta válida. ▪ Usuario: `id` y contraseña: `id` ▪ Una nueva contraseña válida: `idNuevo`	▪ Un usuario sin sesión
When	El usuario intenta actualizar su contraseña	El usuario intenta actualizar su contraseña
Then	La contraseña es actualizada con éxito	La contrasenya no será actualizada

Tabla 56: Pruebas de aceptación, Requisito avanzado 1, Historia de Usuario 3

Tests de integración y aceptación

Los siguientes tests están en el repositorio `app` del proyecto y se encuentran en las rutas:

- `src/test/java/app/test/acceptance/advanced/requirement01/History03.java`
- `src/test/java/app/test/integration/advanced/requirement01/History03.java`

Fragmento de código 55: Test de aceptación del requisito avanzado 1 → HU03

```java
public class History03 extends BaseTest {
    @Override
    @AfterEach
    public void afterEach(TestInfo info) {
        super.afterEach(info);
        client.account.deregister();
    }

    @Test
    public void valid(TestInfo info) {
        // Given
        var id = getId(info);
        var newId = id + "Nuevo";
        client.account.register(id, id);
        Mockito.reset(spy.accountManager);

        // When
        var response = client.account.updateAccount(newId);

        // Then
        Mockito.verify(spy.accountManager).saveAccount(any());
        response.statusCode(HttpStatus.OK.value());
        client.session.logout();
        var state = client.session.login(id, newId);
        state.statusCode(HttpStatus.OK.value());
    }

    @Test
    public void invalid(TestInfo info) {
        // Given
        var id = getId(info);
        client.account.register(id, id);
        client.session.logout();
        Mockito.reset(spy.accountManager);

        // When
        var response = client.account.updateAccount(id);

        // Then
        Mockito.verify(spy.accountManager, never()).saveAccount(any());
        response.statusCode(HttpStatus.UNAUTHORIZED.value());
        var state = client.session.login(id, id);
        state.statusCode(HttpStatus.OK.value());
    }
}
```

Fragmento de código 56: Test de integración del requisito avanzado 1 → HU03

```
15  public class History03 extends BaseTest {
16      @Override
17      @AfterEach
18      public void afterEach(TestInfo info) {
19          super.afterEach(info);
20          client.account.deregister();
21      }
22
23      @Test
24      public void valid(TestInfo info) {
25          // Given
26          var id = getId(info);
27          var newId = id + "Nuevo";
28          client.account.register(id, id);
29          Mockito.reset(spy.accountManager);
30
31          // When
32          var response = client.account.updateAccount(newId);
33
34          // Then
35          Mockito.verify(spy.accountManager).saveAccount(any());
36          response.statusCode(HttpStatus.OK.value());
37          client.session.logout();
38          var state = client.session.login(id, newId);
39          state.statusCode(HttpStatus.OK.value());
40      }
41
42      @Test
43      public void invalid(TestInfo info) {
44          // Given
45          var id = getId(info);
46          client.account.register(id, id);
47          client.session.logout();
48          Mockito.reset(spy.accountManager);
49
50          // When
51          var response = client.account.updateAccount(id);
52
53          // Then
54          Mockito.verify(spy.accountManager, never()).saveAccount(any());
55          response.statusCode(HttpStatus.UNAUTHORIZED.value());
56          var state = client.session.login(id, id);
57          state.statusCode(HttpStatus.OK.value());
58      }
59  }
```

1.2.1.4 Requisito avanzado 1, Historia de usuario 4

Como usuario quiero poder iniciar sesión con unas credenciales únicas para que se me identifique temporalmente en la aplicación.

Escenarios

Escenarios	Credenciales válidas	Sesión iniciada anteriormente	Sesión iniciada posteriormente
E1	Si	No	Si
E2	No	No	No
E3	Si	Si	Si

Tabla 57: Escenarios Requisito avanzado 1, Historia de Usuario 4

Análisis

El escenario E1 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería iniciar sesión con un usuario y contraseña válida, y esté crear una sesión.

El escenario E2 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería iniciar sesión con un usuario y contraseña inválidas, y esté dar error al intentarlo.

Pruebas de aceptación

	E1 (válido)	E2 (inválido)
Given	▪ Unas credenciales correspondientes a una cuenta válida ▪ Usuario: `id` y Contraseña: `id`	▪ Unas credenciales que no corresponden a ninguna cuenta válida ▪ Usuario: `idNuevo` y Contraseña: `idNuevo`
When	Cuando el usuario intenta iniciar sesión	Cuando el usuario intenta iniciar sesión
Then	Tiene acceso a los recursos propios de una sesión	No inicia sesion y no tiene acceso a recursos que dependan de una sesión y no tiene acceso a recursos que dependan de una sesión.

Tabla 58: Pruebas de aceptación, Requisito avanzado 1, Historia de Usuario 4

Tests de integración y aceptación

Los siguientes tests están en el repositorio `app` del proyecto y se encuentran en las rutas:

- `src/test/java/app/test/acceptance/advanced/requirement01/History04.java`
- `src/test/java/app/test/integration/advanced/requirement01/History04.java`

Fragmento de código 57: Test de aceptación del requisito avanzado 1 → HU04

```java
public class History04 extends BaseTest {
    @Override
    @AfterEach
    public void afterEach(TestInfo info) {
        super.afterEach(info);
        client.account.deregister();
    }

    @Test
    public void valid(TestInfo info) {
        // Given
        var id = getId(info);
        client.account.register(id, id);
        client.session.logout();

        // When
        var response = client.session.login(id, id);

        // Then
        response.statusCode(HttpStatus.OK.value());
        var state = client.session.getAccount();
        state.statusCode(HttpStatus.OK.value());
    }

    @Test
    public void invalid(TestInfo info) {
        // Given
        var id = getId(info);
        var idNew = id + "Nuevo";
        client.account.register(id, id);
        client.session.logout();

        // When
        var response = client.session.login(idNew, idNew);

        // Then
        response.statusCode(HttpStatus.UNAUTHORIZED.value());
        var state = client.session.getAccount();
        state.statusCode(HttpStatus.UNAUTHORIZED.value());
        state = client.session.login(id, id);
        state.statusCode(HttpStatus.OK.value());
    }
}
```

Fragmento de código 58: Test de integración del requisito avanzado 1 → HU04

```java
public class History04 extends BaseTest {
    @Override
    @AfterEach
    public void afterEach(TestInfo info) {
        super.afterEach(info);
        client.account.deregister();
    }

    @Test
    public void valid(TestInfo info) {
        // Given
        var id = getId(info);
        client.account.register(id, id);
        client.session.logout();

        // When
        var response = client.session.login(id, id);

        // Then
        response.statusCode(HttpStatus.OK.value());
        var state = client.session.getAccount();
        state.statusCode(HttpStatus.OK.value());
    }

    @Test
    public void invalid(TestInfo info) {
        // Given
        var id = getId(info);
        var idNew = id + "Nuevo";
        client.account.register(id, id);
        client.session.logout();

        // When
        var response = client.session.login(idNew, idNew);

        // Then
        response.statusCode(HttpStatus.UNAUTHORIZED.value());
        var state = client.session.getAccount();
        state.statusCode(HttpStatus.UNAUTHORIZED.value());
        state = client.session.login(id, id);
        state.statusCode(HttpStatus.OK.value());
    }
}
```

1.2.1.5 Requisito avanzado 1, Historia de usuario 5

Como usuario quiero poder cerrar sesión con unas credenciales únicas para que no me identifique temporalmente la aplicación.

Escenarios

Escenarios	Usuario con una sesión	Sesión iniciada anteriormente	Sesión iniciada posteriormente
E1	Si	Si	No
E2	No	No	No
E3	Si	Si	Si

Tabla 59: Escenarios Requisito avanzado 1, Historia de Usuario 5

Análisis

El escenario E1 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería cerrar sesión teniendo una sesión iniciada, y este eliminarse.

El escenario E2 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería cerrar sesión sin tener sesión iniciada, y esta seguir sin existir.

Pruebas de aceptación

	E1 (válido)	E2 (inválido)
Given	▪ Un usuario con sesión ▪ Unas credenciales correspondientes a una cuenta válida ▪ Usuario: usuario y Contraseña: contraseña	▪ Un usuario sin sesión
When	Cuando el usuario intenta iniciar sesión	Cuando el usuario intenta iniciar sesión
Then	Ya no podrá acceder a los recursos dependientes de sesión como ubicaciones guardados	La sesión seguirá sin existir

Tabla 60: Pruebas de aceptación, Requisito avanzado 1, Historia de Usuario 5

Tests de integración y aceptación

Los siguientes tests están en el repositorio `app` del proyecto y se encuentran en las rutas:

- `src/test/java/app/test/acceptance/advanced/requirement01/History05.java`
- `src/test/java/app/test/integration/advanced/requirement01/History05.java`

Fragmento de código 59: Test de aceptación del requisito avanzado 1 → HU05

```java
public class History05 extends BaseTest {
    @Override
    @AfterEach
    public void afterEach(TestInfo info) {
        super.afterEach(info);
        client.account.deregister();
    }

    @Test
    public void valid(TestInfo info) {
        // Given
        var id = getId(info);
        client.account.register(id, id);

        // When
        var response = client.session.logout();

        // Then
        response.statusCode(HttpStatus.OK.value());
        var state = client.session.getAccount();
        state.statusCode(HttpStatus.UNAUTHORIZED.value());
        state = client.session.login(id, id);
        response.statusCode(HttpStatus.OK.value());
    }

    @Test
    public void invalid(TestInfo info) {
        // Given
        var id = getId(info);
        client.account.register(id, id);
        client.session.logout();

        // When
        var response = client.session.logout();

        // Then
        response.statusCode(HttpStatus.OK.value());
        var state = client.session.getAccount();
        state.statusCode(HttpStatus.UNAUTHORIZED.value());
        state = client.session.login(id, id);
        response.statusCode(HttpStatus.OK.value());
    }
}
```

Fragmento de código 60: Test de integración del requisito avanzado 1 → HU05

```java
public class History05 extends BaseTest {
    @Override
    @AfterEach
    public void afterEach(TestInfo info) {
        super.afterEach(info);
        client.account.deregister();
    }

    @Test
    public void valid(TestInfo info) {
        // Given
        var id = getId(info);
        client.account.register(id, id);

        // When
        var response = client.session.logout();

        // Then
        response.statusCode(HttpStatus.OK.value());
        var state = client.session.getAccount();
        state.statusCode(HttpStatus.UNAUTHORIZED.value());
        state = client.session.login(id, id);
        response.statusCode(HttpStatus.OK.value());
    }

    @Test
    public void invalid(TestInfo info) {
        // Given
        var id = getId(info);
        client.account.register(id, id);
        client.session.logout();

        // When
        var response = client.session.logout();

        // Then
        response.statusCode(HttpStatus.OK.value());
        var state = client.session.getAccount();
        state.statusCode(HttpStatus.UNAUTHORIZED.value());
        state = client.session.login(id, id);
        response.statusCode(HttpStatus.OK.value());
    }
}
```

1.2.2. Requisito avanzado 2 - Modo invitado

Para permitir un uso más fluido de la aplicación para los usuarios que naveguen por la red, la aplicación proporcionará un modo invitado. Aquellos usuarios que entren a la aplicación y no tengan una sesión iniciada podrán utilizar la aplicación con normalidad pero sus cambios no se quedarán registrados para la próxima vez que inicie sesión.

1.2.2.1 Requisito avanzado 2, Historia de usuario 1

Como usuario quiero poder realizar todas las acciones sin tener que registrarme para poder probar la aplicación sin tener que dar mis datos.

Los test descritos en los requisitos 1, 2 y 3 deberán pasar también con una sesión de invitado, exceptuando el aspecto de la modificación de la base de datos. Sin embargo, a modo de simplificación, para no duplicar el trabajo, solo se implementara la historia 1 del requisito 1.

Tests de integración y aceptación

Los siguientes tests están en el repositorio `app` del proyecto y se encuentran en las rutas:

`src/test/java/app/test/acceptance/advanced/requirement02/History01.java`

`src/test/java/app/test/integration/advanced/requirement02/History01.java`

Fragmento de código 61: Test de aceptación del requisito avanzado 2 → HU01

```
16  public class History01 extends SessionTest {
17      @Test
18      public void valid() {
19          // Given
20          var name = "Castellon de la Plana";
21          client.session.loginAsGuest();
22          Mockito.reset(spy.accountManager);
23
24          // When
25          var response = client.location.addLocation(name);
26
27          // Then
28          Mockito.verify(spy.accountManager, never()).saveAccount(any());
29          response.statusCode(HttpStatus.OK.value());
30          var state = client.location.getLocations();
31          state.body("", hasSize(1));
32          state.body("name", hasItem(name));
33      }
34
35      @Test
36      public void invalid() {
37          // Given
38          var name = "INVALIDO";
39          client.session.loginAsGuest();
40          Mockito.reset(spy.accountManager);
41
```

```
42          // When
43          var response = client.location.addLocation(name);
44
45          // Then
46          Mockito.verify(spy.accountManager, never()).saveAccount(any());
47          response.statusCode(HttpStatus.NOT_FOUND.value());
48          var state = client.location.getLocations();
49          state.body("", hasSize(0));
50      }
51  }
```

Fragmento de código 62: Test de integración del requisito avanzado 2 → HU01

```
18  public class History01 extends BaseTest {
19      @Test
20      public void valid() {
21          // Given
22          var name = "Castellon";
23          var locationMock = new LocationModel(name, 39.980, -0.033);
24          Mockito.doReturn(locationMock).when(spy.queryManager).getData(name);
25          client.session.loginAsGuest();
26          Mockito.reset(spy.accountManager);
27
28          // When
29          var response = client.location.addLocation(name);
30
31          // Then
32          Mockito.verify(spy.accountManager, never()).saveAccount(any());
33          response.statusCode(HttpStatus.OK.value());
34          var state = client.location.getLocations();
35          state.body("", hasSize(1));
36          state.body("name", hasItem(name));
37      }
38
39      @Test
40      public void invalid() {
41          // Given
42          var name = "INVALIDO";
43          Mockito.doThrow(new MissingError()).when(spy.queryManager).getData(name
                );
44          client.session.loginAsGuest();
45          Mockito.reset(spy.accountManager);
46
47          // When
48          var response = client.location.addLocation(name);
49
50          // Then
51          Mockito.verify(spy.accountManager, never()).saveAccount(any());
52          response.statusCode(HttpStatus.NOT_FOUND.value());
53          var state = client.location.getLocations();
54          state.body("", hasSize(0));
55      }
56  }
```

1.2.2.2 Requisito avanzado 2, Historia de usuario 2

Como usuario quiero poder transformar una cuenta de invitado a una permanente para no necesitar recrear los ajustes de una en la otra.

Escenarios

Escenarios	Cantidad de servicios	Usuario válido	Contraseña válida	Cuenta creada	BBDD modificada
E1	1	Si	Si	Si	Si
E2	0	Si	No	No	No
E3	1	No	Si	No	No
E4	0	No	No	No	No

Tabla 61: Escenarios Requisito avanzado 2, Historia de Usuario 2

Análisis

El escenario E1 podría subdividirse en al menos tres pruebas de aceptación diferentes. Una con un servicio y ninguna ubicación activa, otra con un servicio y una ubicación activa, y otra con un servicio y múltiples ubicaciones.

El escenario E3 podría subdividirse en al menos tres pruebas de aceptación diferentes. Una con un servicio y ninguna ubicación activa, otra con un servicio y una ubicación activa, y otra con un servicio y múltiples ubicaciones.

Pruebas de aceptación

	E1 (válido)	E3 (inválido)
Given	▪ Usuario no tiene cuenta con los datos: ▪ Usuario: id ▪ Contraseña: id ▪ Tiene un servicio activo ('Clima')	▪ Usuario tiene cuenta con los datos: ▪ Usuario: id ▪ Contraseña: id ▪ Tiene un servicio activo ('Clima')
When	Un usuario intenta crear una cuenta	Un usuario intenta crear una cuenta
Then	Se crea una cuenta en el sistema con esas credenciales y tiene un servicio activo ('Clima')	No se puede crear la cuenta porque el usuario ya se encuentra registrado en el sistema

Tabla 62: Pruebas de aceptación, Requisito avanzado 2, Historia de Usuario 2

Tests de integración y aceptación

Los siguientes tests están en el repositorio `app` del proyecto y se encuentran en las rutas:

`src/test/java/app/test/acceptance/advanced/requirement02/History02.java`

`src/test/java/app/test/integration/advanced/requirement02/History02.java`

Fragmento de código 63: Test de aceptación del requisito avanzado 2 → HU02

```
19  public class History02 extends BaseTest {
20      @Override
21      @AfterEach
22      public void afterEach(TestInfo info) {
23          super.afterEach(info);
24          client.account.deregister();
25      }
26
27      @Test
28      public void valid(TestInfo info) {
29          // Given
30          var id = getId(info);
31          var type = ServiceType.WEATHER.name();
32          client.session.loginAsGuest();
33          client.service.disableAllServices();
34          client.service.enableService(type);
35          Mockito.reset(spy.accountManager);
36
37          // When
38          var response = client.account.register(id, id);
39
40          // Then
41          Mockito.verify(spy.accountManager).saveAccount(any());
42          response.statusCode(HttpStatus.OK.value());
43          var state = client.service.getServices();
44          state.body(setupEnabledQuery(true, ""), hasSize(1));
45          state.body(setupEnabledQuery(true, "service.type"), hasItem(type));
46          client.session.logout();
47
48          state = client.session.login(id, id);
49          state.statusCode(HttpStatus.OK.value());
50          state = client.service.getServices();
51          state.body(setupEnabledQuery(true, ""), hasSize(1));
52          state.body(setupEnabledQuery(true, "service.type"), hasItem(type));
53      }
54
55      @Test
56      public void invalid(TestInfo info) {
57          // Given
58          var id = getId(info);
59          var type = ServiceType.WEATHER.name();
60          client.account.register(id, id);
61          client.service.disableAllServices();
62          client.session.logout();
63
64          client.session.loginAsGuest();
65          client.service.disableAllServices();
66          client.service.enableService(type);
67          Mockito.reset(spy.accountManager);
68
69          // When
70          var response = client.account.register(id, id);
71
72          // Then
73          Mockito.verify(spy.accountManager, never()).saveAccount(any());
```

```
74            response.statusCode(HttpStatus.CONFLICT.value());
75            var state = client.service.getServices();
76            state.body(setupEnabledQuery(true, ""), hasSize(1));
77            state.body(setupEnabledQuery(true, "service.type"), hasItem(type));
78            client.session.logout();
79
80            state = client.session.login(id, id);
81            state.statusCode(HttpStatus.OK.value());
82            state = client.service.getServices();
83            state.body(setupEnabledQuery(true, ""), hasSize(0));
84        }
85    }
```

Fragmento de código 64: Test de integración del requisito avanzado 2 → HU02

```
19  public class History02 extends BaseTest {
20      @Override
21      @AfterEach
22      public void afterEach(TestInfo info) {
23          super.afterEach(info);
24          client.account.deregister();
25      }
26
27      @Test
28      public void valid(TestInfo info) {
29          // Given
30          var id = getId(info);
31          var type = ServiceType.WEATHER.name();
32          client.session.loginAsGuest();
33          client.service.disableAllServices();
34          client.service.enableService(type);
35          Mockito.reset(spy.accountManager);
36
37          // When
38          var response = client.account.register(id, id);
39
40          // Then
41          Mockito.verify(spy.accountManager).saveAccount(any());
42          response.statusCode(HttpStatus.OK.value());
43          var state = client.service.getServices();
44          state.body(setupEnabledQuery(true, ""), hasSize(1));
45          state.body(setupEnabledQuery(true, "service.type"), hasItem(type));
46          client.session.logout();
47
48          state = client.session.login(id, id);
49          state.statusCode(HttpStatus.OK.value());
50          state = client.service.getServices();
51          state.body(setupEnabledQuery(true, ""), hasSize(1));
52          state.body(setupEnabledQuery(true, "service.type"), hasItem(type));
53      }
54
55      @Test
56      public void invalid(TestInfo info) {
57          // Given
58          var id = getId(info);
59          var type = ServiceType.WEATHER.name();
```

```
60            client.account.register(id, id);
61            client.service.disableAllServices();
62            client.session.logout();
63
64            client.session.loginAsGuest();
65            client.service.disableAllServices();
66            client.service.enableService(type);
67            Mockito.reset(spy.accountManager);
68
69            // When
70            var response = client.account.register(id, id);
71
72            // Then
73            Mockito.verify(spy.accountManager, never()).saveAccount(any());
74            response.statusCode(HttpStatus.CONFLICT.value());
75            var state = client.service.getServices();
76            state.body(setupEnabledQuery(true, ""), hasSize(1));
77            state.body(setupEnabledQuery(true, "service.type"), hasItem(type));
78            client.session.logout();
79
80            state = client.session.login(id, id);
81            state.statusCode(HttpStatus.OK.value());
82            state = client.service.getServices();
83            state.body(setupEnabledQuery(true, ""), hasSize(0));
84        }
85    }
```

1.2.3. Requisito avanzado 3 - Autocompletado de texto

Para ofrecer una experiencia de usuario más fluida, la aplicación deberá ser capaz de ofrecer sugerencias de búsqueda una vez introducidos los primeros caracteres. Del total de sugerencias ofrecidas por la barra de búsqueda el usuario deberá ser capaz de elegir la que más se adapte a la búsqueda que pretende realizar.

1.2.3.1 Requisito avanzado 3, Historia de usuario 1

Como usuario quiero recibir sugerencias de autocompletado correspondientes a ubicaciones para evitar potenciales búsquedas de ubicaciones inexistentes.

Escenarios

Escenarios	**Más de cuatro caracteres**	**Sugerencia de ubicación**	**Sugerencia correcta**
E1	Si	Si	Si
E2	Si	No	No
E3	No	No	No

Tabla 63: Escenarios Requisito avanzado 3, Historia de Usuario 1

Análisis

El escenario E1 podría subdividirse en al menos tres pruebas de aceptación diferentes. Una con un prefijo válido de cuatro caracteres, otra con un prefijo válido de más de cuatro caracteres pero incompleto, y otra con el nombre completo de la ubicación.

El escenario E2 solo se podría implementar de una manera, ya que al no haber ambigüedad o estado externo hay poca variación. La única prueba dentro de los parámetros del escenario sería usar una ubicación inválida y esta no responder ninguna sugerencia.

Pruebas de aceptación

	E1 (valido)	**E2 (invalido)**
Given	Dados los caracteres: 'cast'	Dados los caracteres: 'INVALIDO'
When	Solicita una sugerencia de autocompletado	Solicita una sugerencia de autocompletado
Then	Devuelve una ubicación válida: 'Castellon'	No devuelve ningúna ubicación

Tabla 64: Pruebas de aceptación, Requisito avanzado 3, Historia de Usuario 1

Tests de integración y aceptación

Los siguientes tests están en el repositorio `app` del proyecto y se encuentran en las rutas:

- `src/test/java/app/test/acceptance/advanced/requirement03/History01.java`
- `src/test/java/app/test/integration/advanced/requirement03/History01.java`

Fragmento de código 65: Test de aceptación del requisito avanzado 3 → HU01

```
12  public class History01 extends SessionTest {
13      @Test
14      public void valid() {
15          // Given
16          var namePartial = "cast";
17          var name = "Castellon de la Plana";
18
19          // When
20          var response = client.query.query(namePartial);
21
22          // Then
23          response.statusCode(HttpStatus.OK.value());
24          response.body("", hasSize(1));
25          response.body("name", hasItem(name));
26      }
27
28      @Test
29      public void invalid() {
30          // Given
31          var name = "INVALIDO";
32
33          // When
34          var response = client.query.query(name);
35
36          // Then
37          response.statusCode(HttpStatus.OK.value());
38          response.body("", hasSize(0));
39      }
40  }
```

Fragmento de código 66: Test de integración del requisito avanzado 3 → HU01

```
17  public class History01 extends SessionTest {
18      @Test
19      public void valid() {
20          // Given
21          var namePartial = "cast";
22          var name = "Castellon";
23          var locationMock = new LocationModel(name, 39.980, -0.033);
24          Mockito.doReturn(List.of(locationMock)).when(spy.queryManager).
                getAllData(namePartial);
25
26          // When
27          var response = client.query.query(namePartial);
28
```

```
29          // Then
30          response.statusCode(HttpStatus.OK.value());
31          response.body("", hasSize(1));
32          response.body("name", hasItem(name));
33      }
34
35      @Test
36      public void invalid() {
37          // Given
38          var name = "INVALIDO";
39          Mockito.doReturn(Collections.emptyList()).when(spy.queryManager).
                getAllData(name);
40
41          // When
42          var response = client.query.query(name);
43
44          // Then
45          response.statusCode(HttpStatus.OK.value());
46          response.body("", hasSize(0));
47      }
48  }
```