Plan de pruebas US 509051 Mostrar últimas gasolineras cargadas en modo sin conexión.

PRUEBAS DE ACEPTACIÓN

Las pruebas de aceptación se encuentran definidas en la tarjeta de scrumdesk “509051 - Mostrar últimas gasolineras cargadas en modo sin conexión”.

PRUEBAS DE UI

Son las mismas que las pruebas de aceptación, con excepción de que los resultados obtenidos de la API se simulan empleando los datos del ***Anexo 1***, esto con el fin de permitir la automatización de las pruebas a través de Espresso mediante la comprobación de datos estables, estos son, datos que no cambian con el tiempo como sucede con la respuesta de la API real.

Los casos de prueba de UI definidos según los datos del Anexo 1 son los siguientes:

ID	Contexto	Resultado
UI.1.a	Las gasolineras de la **Lista 1** están **precargadas**, **Se cargan** las gasolineras de la **Lista 2**, Aplicación con acceso a internet, API de datos funcionando correctamente, Base de datos funcionando correctamente, Aplicación no abierta en 2º plano.	Se muestra en un listado: CEPSA y REPSOL. Toast notificando 2 gasolineras cargadas.
UI.1.b	Las gasolineras de la **Lista 1** están **precargadas**, Aplicación **sin** acceso a internet, API de datos funcionando correctamente, Base de datos funcionando correctamente, Aplicación no abierta en 2º plano.	Se muestra en un listado: PETROPRIX y BALLENOIL. Toast notificando 2 gasolineras cargadas y el día 17/11/2024 siguiendo el formato indicado.
UI.2	Las gasolineras de la **Lista 1** están **precargadas**, Aplicación con acceso a internet, API de datos **no** responde, Base de datos funcionando correctamente, Aplicación no abierta en 2º plano.	Se muestra en un listado: PETROPRIX y BALLENOIL. Toast notificando 2 gasolineras cargadas y el día 17/11/2024 siguiendo el formato indicado.
UI.3	**No hay gasolineras precargadas**, Aplicación **sin** acceso a internet,	Se muestra un listado vacío.

	API de datos funcionando correctamente, Base de datos **sin datos**, Aplicación no abierta en 2º plano.	Toast notificando que no existen datos guardados.
UI.4	**No hay gasolineras precargadas**, Aplicación **sin** acceso a internet, API de datos funcionando correctamente, Base de datos **genera error de lectura**, Aplicación no abierta en 2º plano.	Se muestra un listado vacío. Toast notificando error al cargar los datos.
UI.5	**Se cargan** las gasolineras de la **Lista 2**, Aplicación con acceso a internet, API de datos funcionando correctamente, Base de datos **genera error de escritura**, Aplicación no abierta en 2º plano.	Se muestra en un listado: CEPSA y REPSOL. Toast notificando 2 gasolineras cargadas. Toast notificando error al registrar las nuevas gasolineras cargadas.

Tabla 1: Casos de prueba UI.

LISTA DE CLASES CON MÉTODOS A PROBAR CON DOCUMENTACIÓN

A continuación, se listan las clases con los métodos a probar por haber sido creados o modificados junto a un apartado de documentación para describir lo que realizan actualmente en el código del proyecto.

De la clase "*MainPresenter.java", el método siguiente*:

Métodos	Documentación
void load().	Se encarga de solicitar y recibir asíncronamente la respuesta de la API con las gasolineras a cargar, encargándose también de solicitar la actualización de la vista. Tiene dos comportamientos internos. Cuando la API retorna gasolineras: • Persiste las Gasolineras en la DDBB. • Inicializa la lista de Gasolineras. Y cuando encuentra algún tipo de error: • Carga las Gasolineras guardadas en la DDBB. • Lanza un mensaje de error si no hay guardadas. • Lanzar un mensaje de error genérico en cualquier otro caso de fallo.

Tabla 2: Documentación para los métodos de *ICallBack*.

De la clase "*IGasStationsDAO.java*", los métodos siguientes:

Métodos	Documentación

List<Gasolinera> getAll().	Obtiene todas las gasolineras.
Gasolinera getGasStationById(int id).	Obtiene una gasolinera por ID.
void addGasStation(Gasolinera gasolinera).	Añade una gasolinera.
void deleteAll().	Borra todas las gasolineras.

Tabla 3: Documentación para los métodos de *IGasStationsDAO*.

De la clase "*Gasolinera.java*", los métodos siguientes:

Métodos	**Documentación**
Gasolinera().	Crea la gasolinera con un id "" en vez de null.
Getters y Setters de los parámetros.	Implementación típica, excepto en el caso del Setter del id que no permite null.

Tabla 4: Documentación para los métodos de *Gasolinera*.

De la clase "*MainView.java*", los métodos siguientes:

Métodos	**Documentación**
void showLoadCorrectFromLocalDB(int stations).	Muestra una notificación indicando que el número de gasolineras pasadas como parámetro se cargaron correctamente, junto a esta, la fecha de registro de los datos guardados.
IGasStationsDAO getGasolinerasDAO().	Obtiene la instancia de gasolinerasDAO para extracción de datos.
void updateLocalDBDateRegister()	Actualiza las Preferencias que almacenan la fecha en la que se actualizó la base de datos local para que almacene la fecha actual.
String getLocalDBDateRegister().	Recupera la fecha en la que se actualizó la base de datos local.

Tabla 5: Documentación para los métodos de *IMainContract*.

ESPECIFICACIÓN DE PRUEBAS

A continuación, se especifican casos de prueba unitaria o integración para dos métodos diferentes.

Prueba unitaria

De la clase "*MainPresenter.java*", empleando los datos del ***Anexo 1***, el método:

- void load()

Para ello, será necesario emplear el uso de objetos Mock para la interfaz IMainContract#View, además de objetos Mock para la interfaz de la DAO IGasStationsDAO.

Los casos de prueba se nombrarán bajo la nomenclatura UP1.X.

ID	**Entrada**	**Valor esperado**

UP1.a	**Persistencia**: No existen gasolineras precargadas. **Respuesta API:** {CEPSA, REPSOL}.	Se comprueba que IGasStationsDAO tiene {CEPSA, REPSOL} gasolineras. Se comprueba que se llama a view.updateLocalDBDateRegister() para actualizar la fecha de actualización. Se comprueba que se llama a view.showStations() con {CEPSA, REPSOL}. Se comprueba que se llama a view.showInfoMessage() con un mensaje notificando el número de gasolineras cargadas.
UP1.b	**Persistencia**: {PETROPRIX, BALLENOIL}. **Respuesta API:** {CEPSA, REPSOL}.	Se comprueba que IGasStationsDAO tiene {CEPSA, REPSOL} gasolineras. Se comprueba que se llama a view.updateLocalDBDateRegister() para actualizar la fecha de actualización. Se comprueba que se llama a view.showStations() con {CEPSA, REPSOL}. Se comprueba que se llama a view.showInfoMessage() con un mensaje notificando el número de gasolineras cargadas.
UP1.c	**Persistencia**: {PETROPRIX, BALLENOIL}. **Respuesta API:** Se lanza onFailure().	Se comprueba que IGasStationsDAO tiene {PETROPRIX, BALLENOIL} gasolineras. Se comprueba que **no** se llama a view.updateLocalDBDateRegister() para actualizar la fecha de actualización. Se comprueba que se llama a view.showStations() con {PETROPRIX, BALLENOIL}. Se comprueba que se llama a view.showInfoMessage() con un mensaje notificando el número de

		gasolineras cargadas y su fecha de actualización 17/11/2024 siguiendo el formato indicado.
UP1.d	**Persistencia**: No existen gasolineras precargadas. **Respuesta API:** Se lanza onFailure().	Se comprueba que IGasStationsDAO tiene {} gasolineras. Se comprueba que **no** se llama a view.updateLocalDBDateRegister() para actualizar la fecha de actualización. Se comprueba que se llama a view.showStations() con {}. Se comprueba que se llama a view.showInfoMessage() con un mensaje notificando que no existen datos guardados.
UP1.e	**Persistencia:** {PETROPRIX, BALLENOIL} (Error lectura en BBDD). **Respuesta API:** Se lanza onFailure().	**No** se lanza SQLiteException. Se comprueba que IGasStationsDAO tiene {} gasolineras. Se comprueba que **no** se llama a view.updateLocalDBDateRegister() para actualizar la fecha de actualización. Se comprueba que se llama a view.showStations() con {}. Se comprueba que se llama a view.showLoadError() con un mensaje notificando un error cargando las gasolineras.
UP1.f	**Persistencia:** {PETROPRIX, BALLENOIL}. **Respuesta API:** {CEPSA, REPSOL} (Error escritura en DDBB).	**No** se lanza SQLiteException. Se comprueba que IGasStationsDAO tiene {PETROPRIX, BALLENOIL} gasolineras. Se comprueba que **no** se llama a view.updateLocalDBDateRegister() para actualizar la fecha de actualización.

		Se comprueba que se llama a view.showStations() con {PETROPRIX, BALLENOIL}.. Se comprueba que se llama a view.showInfoMessage () con un mensaje notificando el número de gasolineras cargadas y su fecha de actualización 17/11/2024 siguiendo el formato indicado. Se comprueba que se llama a view.showLoadError() con un mensaje notificando un error al actualizar la DDBB local.
UP1.g	**Persistencia:** No existen gasolineras precargadas. **Respuesta API:** {}.	Se comprueba que IGasStationsDAO tiene {} gasolineras. Se comprueba que se llama a view.updateLocalDBDateRegister() para actualizar la fecha de actualización. Se comprueba que se llama a view.showStations() con {}. Se comprueba que se llama a view.showInfoMessage() con un mensaje notificando 0 gasolineras cargadas.
UP1.h	**Persistencia:** {PETROPRIX, BALLENOIL}. **Respuesta API:** {}.	Se comprueba que IGasStationsDAO tiene {} gasolineras. Se comprueba que se llama a view.updateLocalDBDateRegister() para actualizar la fecha de actualización. Se comprueba que se llama a view.showStations() con {}. Se comprueba que se llama a view.showInfoMessage() con un mensaje notificando 0 gasolineras cargadas.

Tabla 6: Casos de prueba unitaria load().

Prueba de integración

Debemos probar la integración entre la nueva clase de dominio IGasStationsDAO y el Presenter modificado. Para ello, será necesario el uso de objetos Mock para la interfaz IMainContract#View, probando así la interacción entre IGasStationsDAO y el Presenter.

En este caso, aparte del método load(), el resto de métodos son Getters, Setters, un constructor, métodos del View y de la DAO; Y no hay que probar ninguno de estos. Así que para las pruebas de integración, se especifican los mismos casos de prueba que los descritos en la Tabla 6, pero siguiendo la nomenclatura IP2.X en vez de UP1.X.

Nótese que los apartados de este documento han sido redactados siguiendo las Normas de Evaluación para Procesos de Ingeniería Software - Planes de Prueba.

Anexo 1 – Listados de gasolineras con datos de interes

Lista 1: Listado de gasolineras almacenado inicialmente cuando la aplicación está **sin** conexión:

- **Fecha de carga de los datos (para todas las gasolineras):** 17/11/2024.
- **Rótulo (IDs de objetos gasolinera):** PETROPRIX y BALLENOIL.

Lista 2: Listado de gasolineras almacenado cuando la aplicación consigue conexión:

- **Fecha de carga de los datos (para todas las gasolineras):** Fecha actual.
- **Rótulo (IDs de objetos gasolinera):** CEPSA y REPSOL.

IMPLEMENTACION DE LAS PRUEBAS

Prueba de UI para el caso de UI.1.A (Pablo Landeras)

Después de programar y ejecutar los tests no se ha detectado ningún problema en el código analizado.

Prueba unitaria para el caso de UP.1 (Lucía Fernández Mancebo)

A la hora de programar y ejecutar los test definidos, se han encontrado los siguientes problemas en el código:

- UP1.d: No se llamaba al método showStations(). Esto se debía a que en el onFailure() no se llamaba al initialiceGasStationList() cuando la lista estaba vacía, sino cuando tenía elementos. Para solucionarlo, se llama a dicho método al inicio del try, con la lista de gasolineras (así se invoca tanto si la lista está vacía como si está llena).
- UP1.e: No se llamaba al método showStations() al lanzarse la excepción de lectura de BBDD, debido a que, no se valora el hecho de mostrar una lista vacía cuando se produce una excepción (en general), y por ello, tampoco se muestra el mensaje de error al cargar las gasolineras. Al final, se optó por invocar al método initialiceGasStationList() con una lista vacía en el catch para las excepciones.
- UP1.f: No se mostraba el mensaje de “Error al guardar en la Base de Datos”, no se capturaba en el onSuccess() la excepción de BBDD que retornase el mensaje de error. Para ello, se ha añadido un bloque catch que capture la excepción (y otras excepciones) y retorne el mensaje. Además de inicializar la lista de gasolineras retornadas por el servicio con un bloque finally.

Antes, se trataba la excepción de base de datos como una SQLException, que no se podía lanzar en la programación de los Mocks y se envolvía dentro de un RuntimeException para lanzarlo. Sin embargo, se ha encontrado la excepción SQLiteException, y dado que Room funciona con SQLite, se ha decidido cambiar esto en los test, verificando su correcto funcionamiento.

Prueba de integración para el caso de IP.2 (Lucía Fernández Mancebo)

A la hora de realizar las pruebas de integración, se han programado los mismos casos que en las pruebas unitarias a excepción de los fallos en la DAO. No ha habido problemas detectados en la ejecución de estas pruebas ya que se han realizado tras las unitarias, y se ha podido comprobar que todo funciona correctamente.

Arreglo de pruebas unitarias del PointsPresenter.java (Lucía Fernández Mancebo)

Se han arreglado para la clase PointsPresenterTest.java los errores de los Test Unitarios realizados. Se han arreglado los test que lanzaban excepciones para las pruebas del “Crear puntos de Interés por coordenadas”, y se ha añadido el método equals() para la clase Puntos de Interés.