Mateo Rada Arias A00368693

Kennet Sánchez Roldan A00369521xd

Enunciado

El Proyecto consiste en una aplicación la cual sirva como administrador financiero, la que pretende facilitar el manejo del dinero, incluyendo gastos e ingresos clasificándolos según su naturaleza, mediante la visualización de estos en las distintas fechas que se hayan registrado. El programa llevara el nombre de Budget4u.

Requerimientos funcionales

Req1. Administrar los ingresos que digite el usuario con fecha, cantidad, naturaleza y descripción. No se podrán ingresar ingresos menores que 1 o nulos. Los ingresos se dividirán en distintas naturalezas (regular, irregular, préstamo) los cuales tendrán distinta información según la naturaleza del ingreso.

> Req.1.1. Agregar un nuevo ingreso de naturaleza elegida por el usuario. Verificando que todos los datos estén completos y sean válidos.
>
> Req1.2. Eliminar un ingreso ya existente.
>
> Req1.3. Editar un ingreso y/o su información.
>
> Req1.4. Guardar de manera persistente la información de los ingresos.
>
> Req1.5. Permitir la visualización grafica de todos los ingresos seleccionados por criterios elegidos por el usuario (fecha, naturaleza, etc.).
>
> Req1.6. Crear un ingreso de tipo préstamo SI Y SOLO SI el usuario ya ha registrado un prestamista. El monto de este ingreso se verá descontado en la fecha del cobro ingresada por el usuario.

Req2. Administrar los gastos que digite el usuario con fecha, cantidad, naturaleza y descripción. No se podrán ingresar gastos menores que 1 o nulos. Los gastos se dividirán en distintas naturalezas (hogar, ocio, extraordinarios, ordinarios) los cuales tendrán distinta información según la naturaleza del gasto.

> Req.2.1. Agregar un nuevo gasto de naturaleza elegida por el usuario. Verificando que todos los datos estén completos y sean válidos.
>
> Req2.2. Eliminar un gasto ya existente.
>
> Req2.3. Editar un gasto y/o su información.
>
> Req2.4. Guardar de manera persistente la información de los gastos.

Req2.5. Permitir la visualización grafica de todos los gastos seleccionados por criterios elegidos por el usuario (fecha, naturaleza, etc.).

Req3. Visualizar el balance de dinero dependiendo de los gastos e ingresos digitados por el usuario.

Req3.1. Visualizar el balance esperado para los próximos meses dependiendo de los gastos e ingresos registrados por el usuario.

Req3.2. Visualizar el balance actual del usuario.

Req3.3. Visualizar el balance entre una fecha definida por el usuario

Req3.4. Visualizar la hora constantemente en la esquina inferior derecha.

Req4. Administrar la información del usuario actual. Un usuario no podrá modificar ni acceder a la información de otro usuario. Cada usuario tendrá nickname, contraseña y tipo. Los tipos de usuarios serán: empleado, estudiante o regular.

Req4.1. Registrar un nuevo usuario con Nickname, contraseña y tipo. No se podrá registrar un usuario con el mismo Nickname que otro.

Req4.2. Permitir al usuario eliminar su cuenta con toda la información que posea en ella.

Req5. Administrar los prestamistas ingresados por el usuario. Los usuarios tendrán sus prestamistas aparte y no se verán influenciados por los prestamistas de otros usuarios. Los prestamistas permitirán al usuario registrar ingresos de tipo préstamo. Cada prestamista tendrá nombre, teléfono de contacto y email.

Req6. Generar un balance mensual con el monto, ingresos totales y gastos totales de todo el mes.

Requerimientos no funcionales

1. Verificar que el usuario se encuentre en el arreglo de usuarios, de ser así, que la contraseña concuerde con la ingresada por el usuario.
2. Los ingresos y gastos deberán ser guardados en un árbol binario de búsqueda, dichos arboles serán guardados en la información de cada usuario.
3. El programa será capaz de ordenar los ingresos mediante el método de ordenamiento burbuja
4. El programa será capaz de ordenar los gastos mediante el método de ordenamiento de inserción.
5. El programa avisará al usuario de las excepciones atrapadas por el programa con el fin de que corrija su error.
6. La hora mantendrá constantemente actualizada en una esquina de la pantalla mediante un hilo aparte.
7. El programa no tendrá limite para guardar ingresos, costos o usuarios.

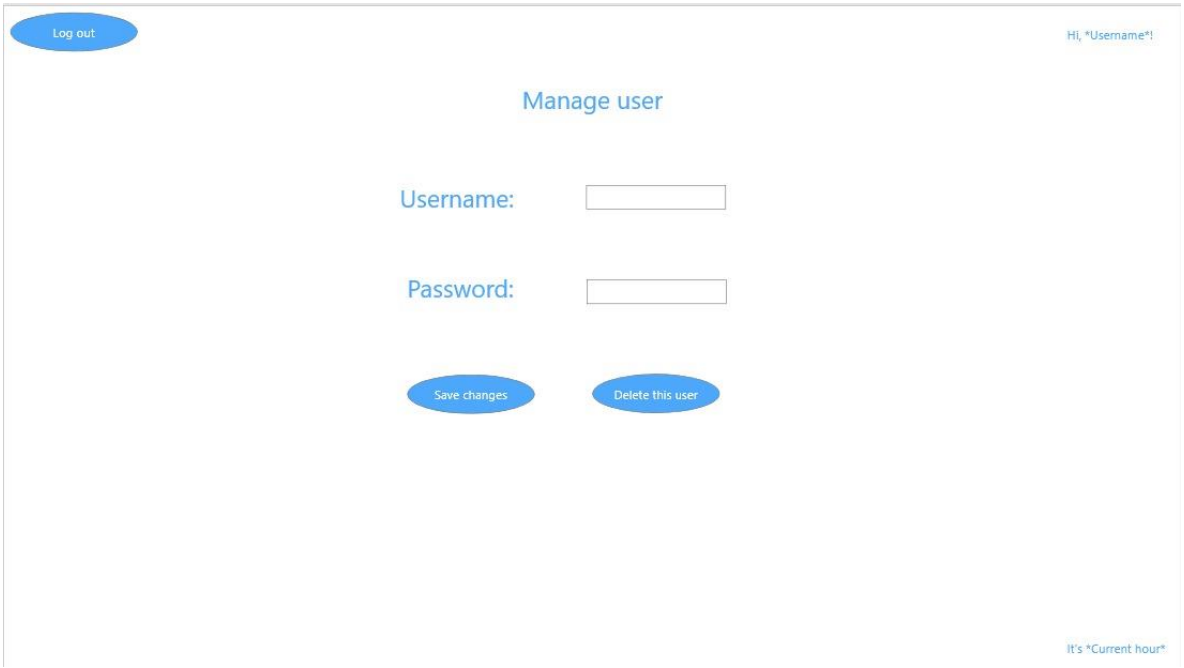
Justificación

Se quiere que el proyecto sea lo mas completo y funcional posible, así como un desarrollo rápido y eficiente, además, el trabajo en conjunto de dos desarrolladores permite la visualización de nuevas ideas y permite un mejor desarrollo de la aplicación.

Sketches

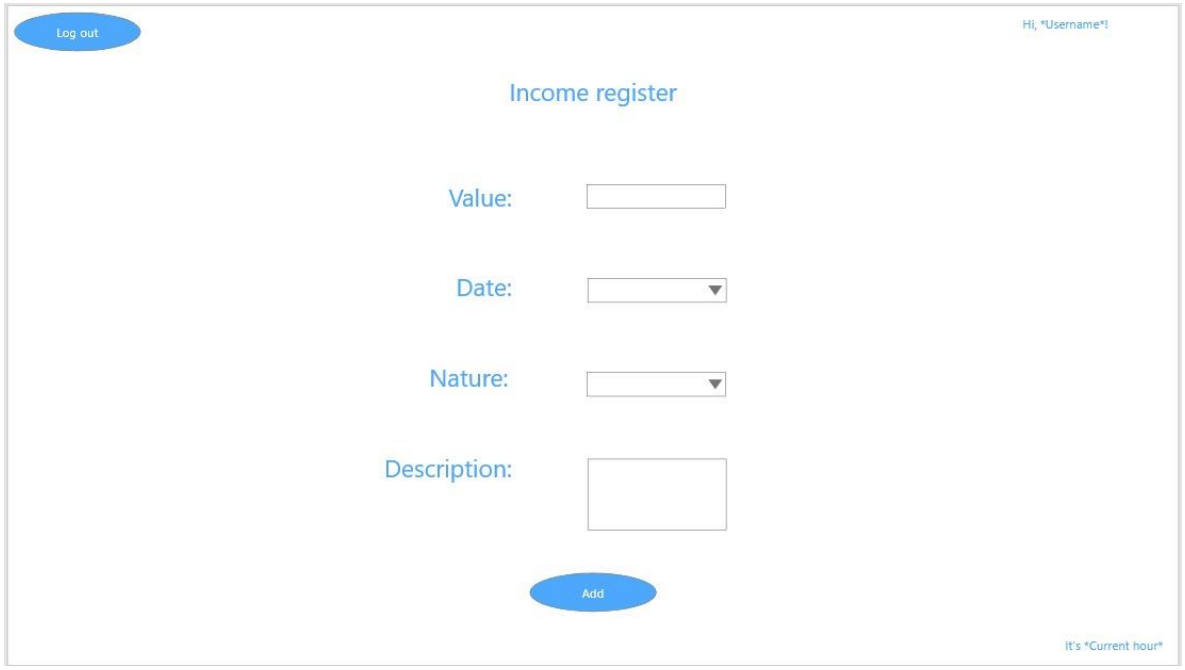
1. Pantalla principal

2. Editar perfil

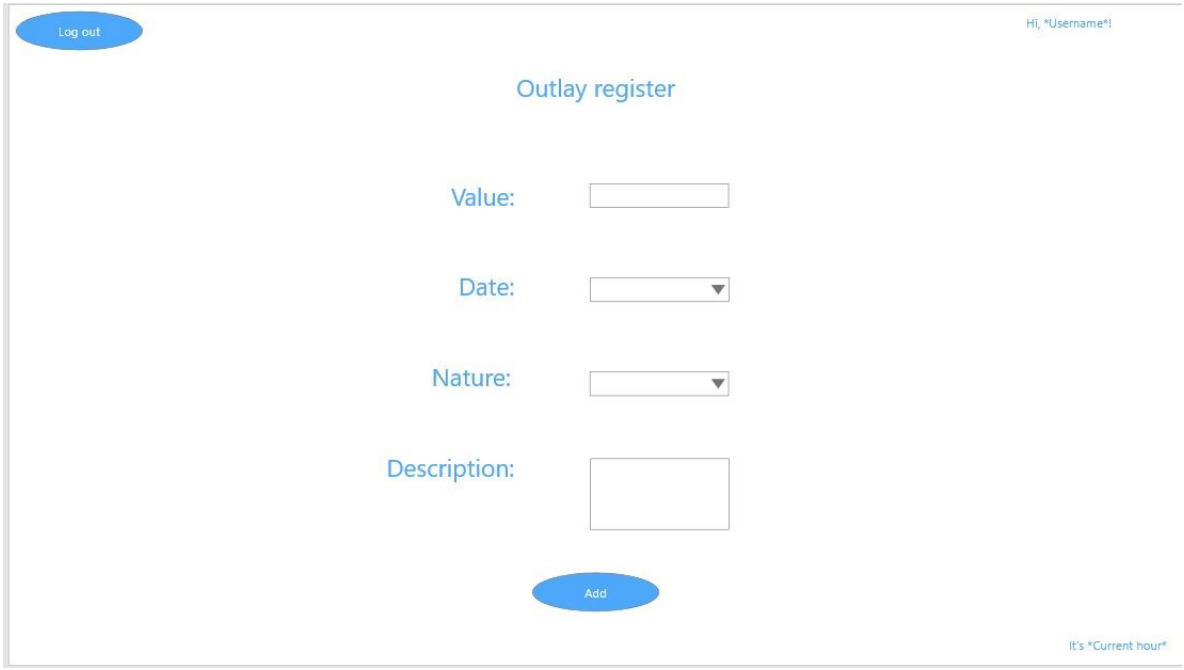


3. Menú principal

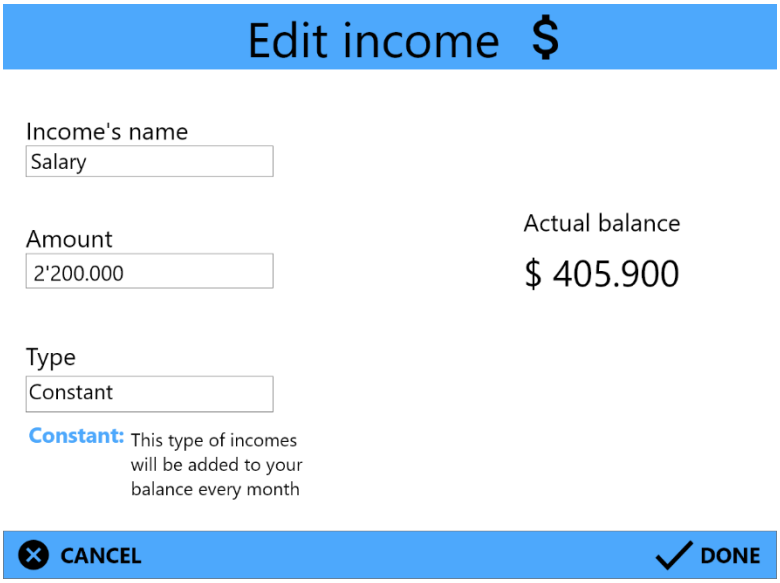
4. Agregar ingreso



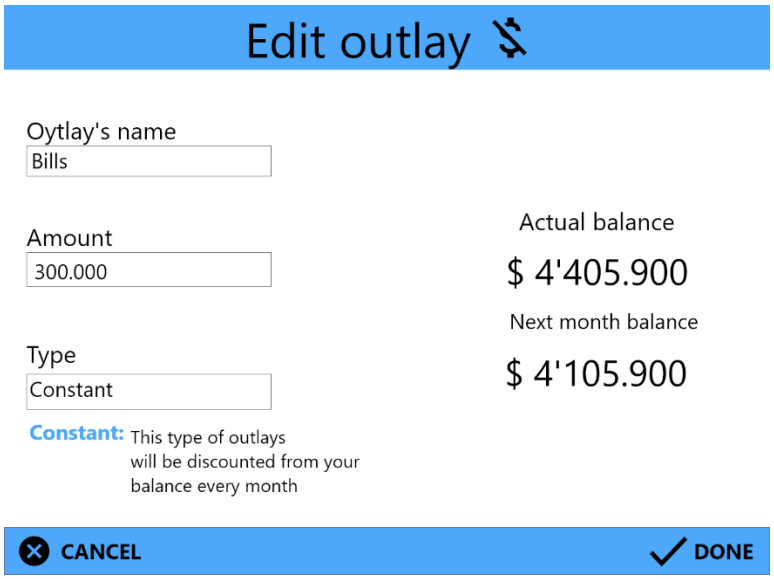
5. Agregar gasto



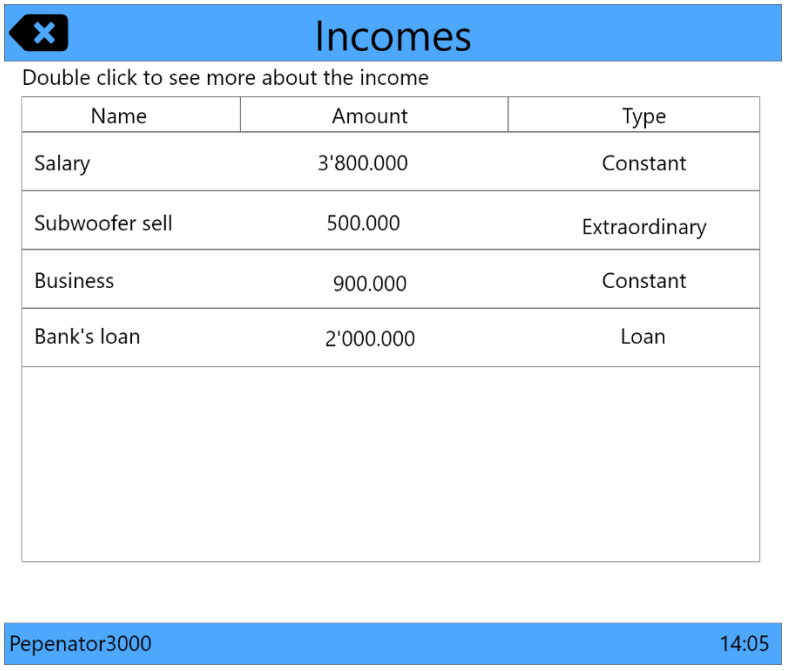
6. Editar ingreso



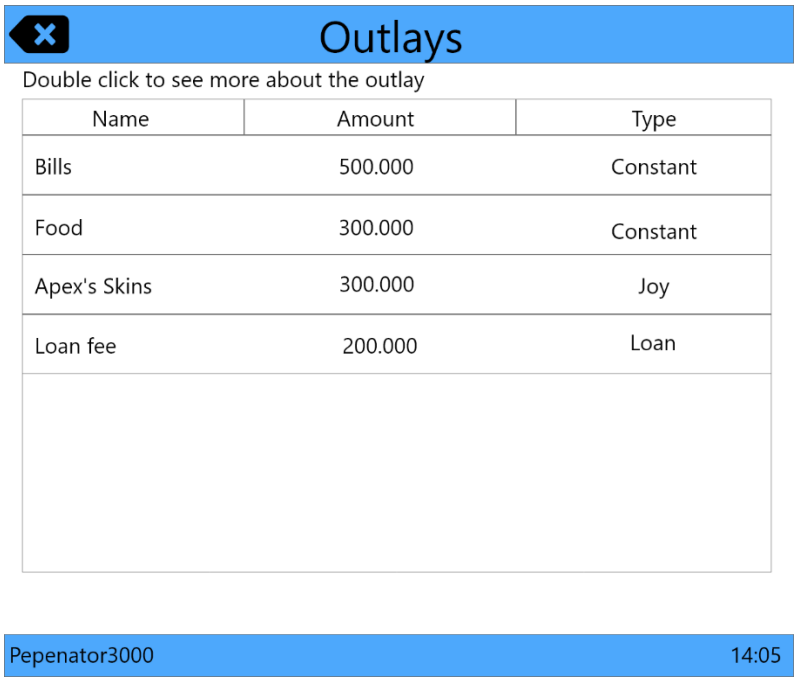
7. Editar gasto



8. Lista de ingresos

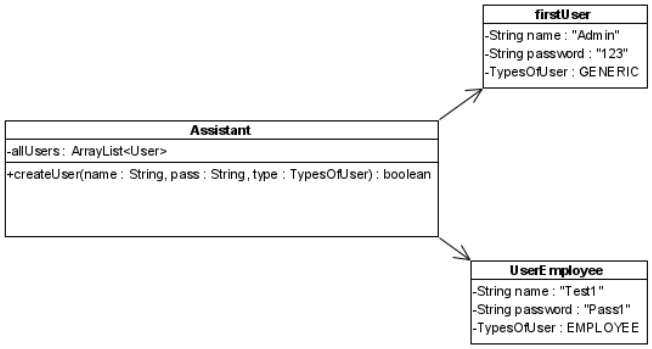


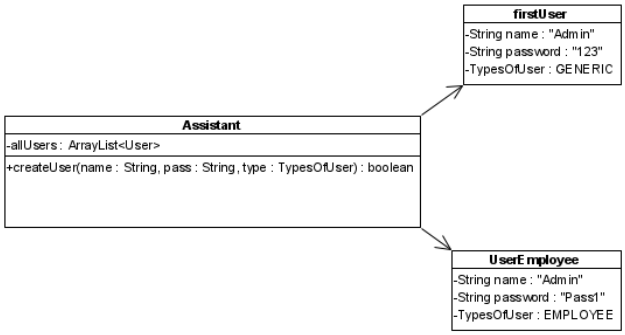
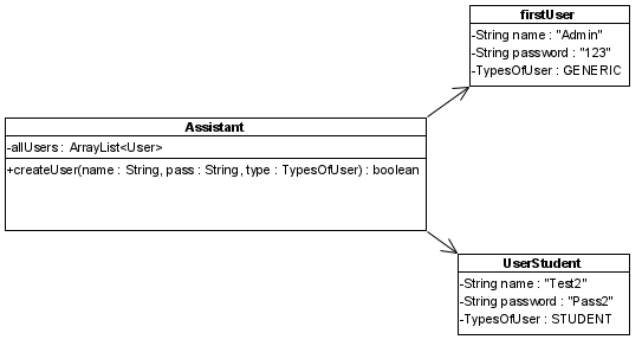
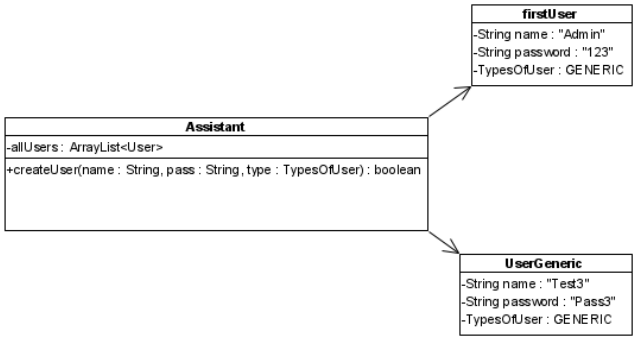
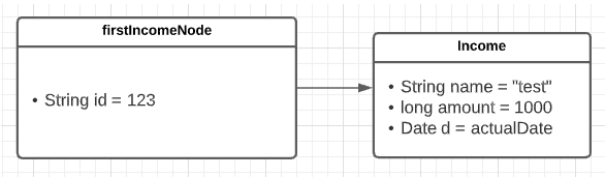
9. Lista de gastos

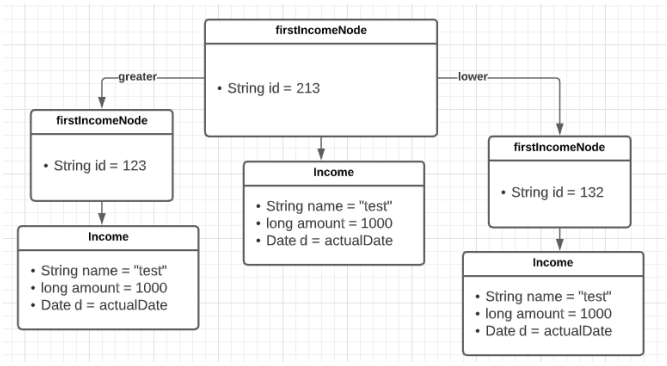


ENTREGA 2

Diseño de escenarios.

Nombre	Clase	Escenario
setupScenery1	Assistant	

setupScenery2	Assistant	
SetUpScenery3	Assistant	
SetUpScenery4	Assistant	
Nombre	Clase	Escenario
setupScenary1		

setupScenary2		

Objetivo de la Prueba: verificar que el método de adición funciona correctamente**.**				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
Assistant	AddUser	setUpScenery1	name: “Test1” password: “Pass1” type: EMPLOYEE	True. El usuario fue agregado pues no había otro con el mismo nombre

Objetivo de la Prueba: verificar que el método de adición detecta si hay otro usuario con el mismo nombre.				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
Assistant	AddUser	setUpScenery2	name: “Admin” password: “Pass1” type: EMPLOYEE	False. El usuario no fue agregado porque ya había otro con el mismo nombre.

Objetivo de la Prueba: verificar que los métodos getters funcionan correctamente.				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
UserEmployee	getName	setUpScenery2		Test1. El método funciona correctamente
UserEmployee	getPassword	setUpScenery2		Pass1. El método funciona correctamente
UserEmployee	getType	setUpScenery2		EMPLOYEE. El método funciona correctamente

Objetivo de la Prueba: verificar que el método de adición funciona correctamente**.**				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
Assistant	AddUser	setUpScenery3	name: “Test2” password: “Pass2” type: STUDENT	True. El usuario fue agregado correctamente porque no había otro con el mismo nombre

Objetivo de la Prueba: verificar que los métodos getters funcionan correctamente.				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
UserStudent	getName	setUpScenery3		Test2. El método funciona correctamente
UserStudent	getPassword	setUpScenery3		Pass2. El método funciona correctamente
UserStudent	getType	setUpScenery3		STUDENT. El método funciona correctamente

Objetivo de la Prueba: verificar que el método de adición funciona correctamente**.**

Clase	Método	Escenario	Valores de Entrada	Resultado
Assistant	AddUser	setUpScenery4	name: “Test3” password: “Pass3” type: GENERIC	True. El usuario fue agregado porque no había otro con el mismo nombre.

Objetivo de la Prueba: verificar que los métodos getters funcionan correctamente.				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
UserGeneric	getName	setUpScenery4		Test3. El método funciona correctamente
UserGeneric	getPassword	setUpScenery4		Pass3. El método funciona correctamente
UserGeneric	getType	setUpScenery4		GENERIC. El método funciona correctamente

Objetivo de la Prueba: Verificar la correcta adición de un nuevo IncomeNode al árbol binario de Incomes				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
IncomeNode	addNode	setupScenary1	Income newIncome Name = “Burguer” Amount = 100000 Date = actualDate Purpose = “Hunger”	True Se agrega correctamente el nuevo nodo al árbol binario de búsqueda. No es necesario verificar que se repitan ya que nunca se podrán repetir Nodos con la misma informacion

Objetivo de la Prueba: Verificar la correcta creación de una ArrayList en base al árbol binario de búsqueda de incomes				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**

IncomeNode	realIncomes	setupScenary2		True Se genera un ArrayList con todos los incomes dentro del árbol binario de los incomes

Objetivo de la Prueba: Verificar el método de búsqueda de un Income en el árbol binario de busqueda				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
IncomeNode	searchNode	setupScenary1	Income in	True Se encontró el income deseado en base al nombre del Income

NOTA: Los requerimientos funcionales implementados en esta versión fueron: Visualizar el balance de dinero, Administrar la información del usuario actual y Administrar ingresos de distinta naturaleza