MAS Final Project – Gaming Electronic Store

1. User Requirements

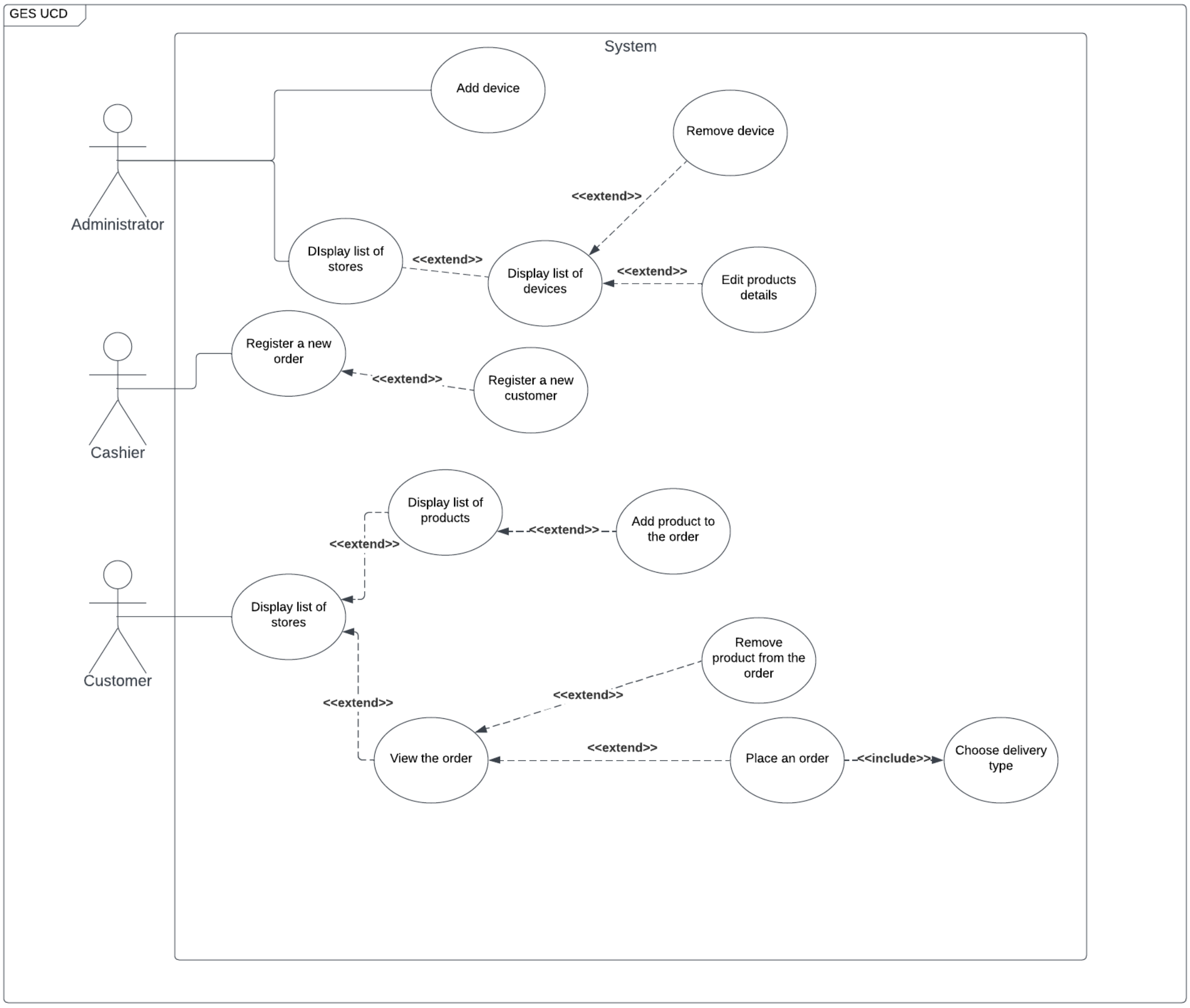
The Gaming Electronic Store (GEC) application must allow a modern way of handling the store information. The application should be able to display a list of devices, accessories, and games for purchase. Before entering the main page, administrator should be able to choose a desired store to work with. Devices, accessories and games should have separated searches. Devices of any type must contain information about their price, name and type as a general information, and details as specific information. Once device details are open, it should show device information.

Administrator can add or remove devices from the store. If the device is being removed, it is first being checked. If the device is the last one in the store, then the error appears. Otherwise, device can be deleted from the store

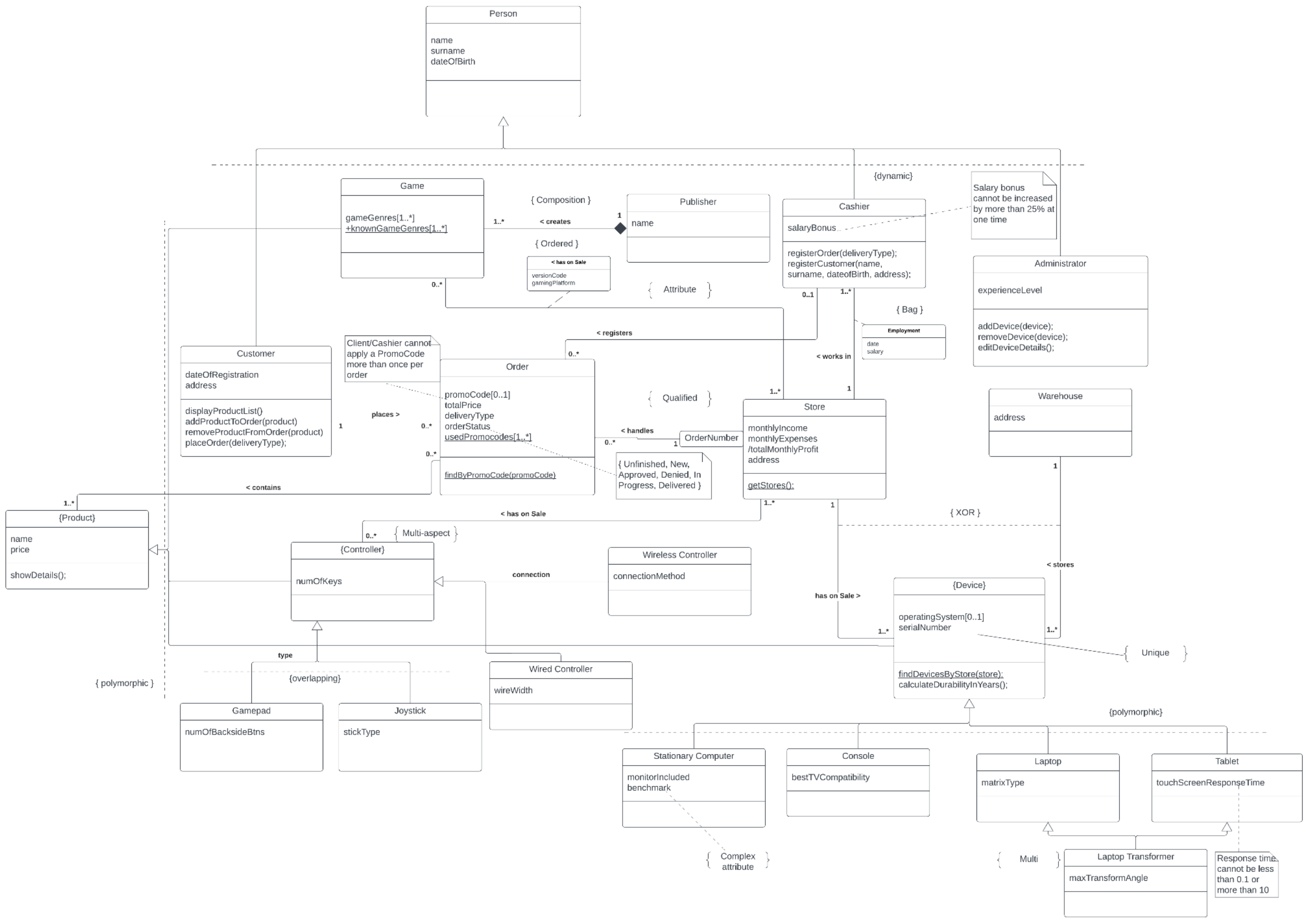
Additionally for the order, it should have a status showing the current state. States are (“Unfinished”, “New”, “Denied”, “Approved”, “In Progress”, “Delivered”).

The web application has three types of actors: Administrator, Customer and Cashier. A person can switch between roles, but they must have only one role at the time. Any person can open list of products, filter through them and view details. Person information should contain first name and last name, and date of birth. Customer must additionally have a registration date attribute. Customer should be able to add and remove products from the basket, also to place a new order. Cashier additionally must have a salary bonus attribute, which cannot be increased by more than 25% at one time. Cashier should be able to register a new order and register a new customer. For administrator, we want them to have experience level as an attribute, and be able to add and remove products, and edit product details.

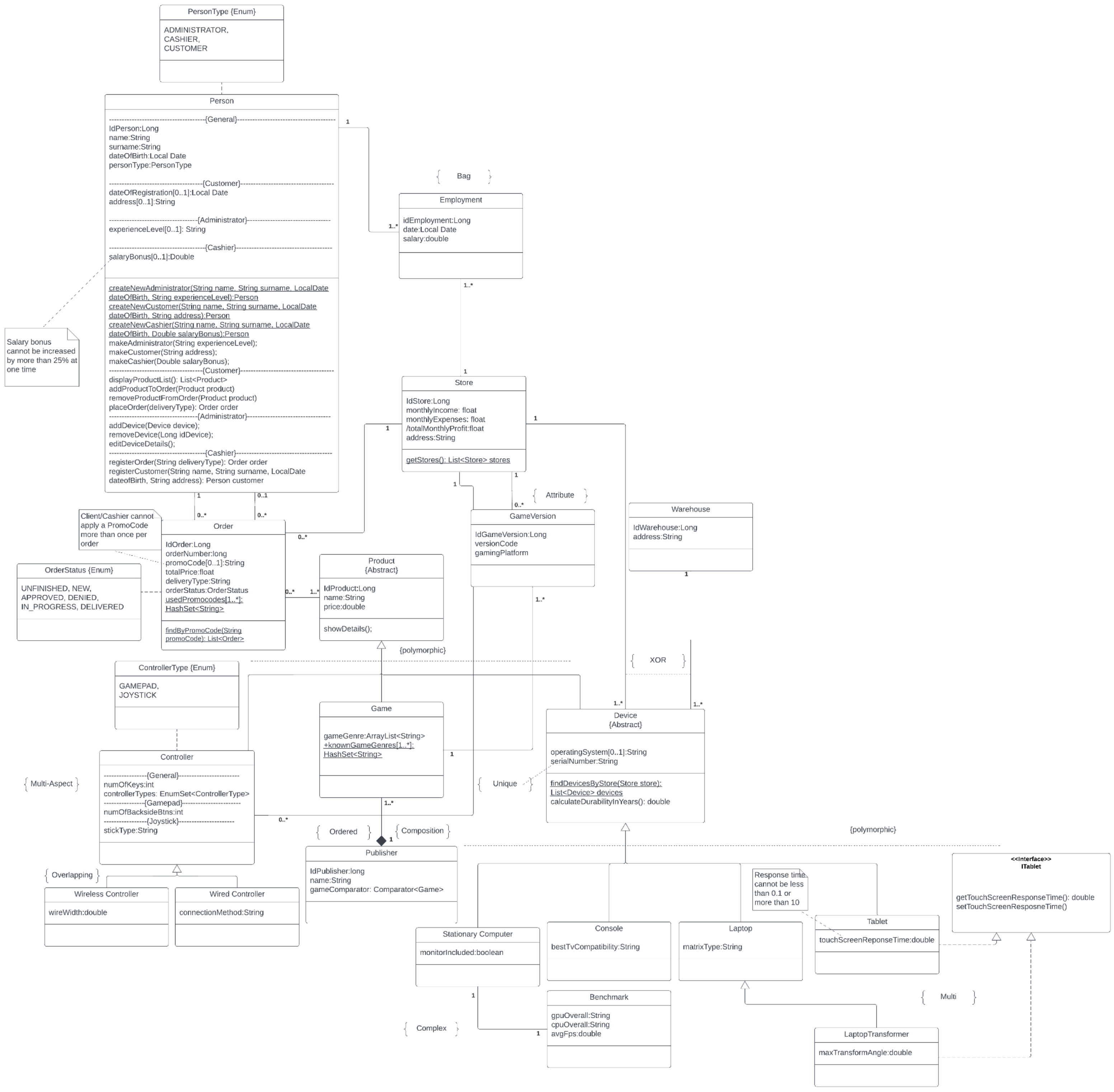
2. The use case diagram



3. The class diagram – analytical



4. The class diagram – design


PersonType {Enum}
ADMINISTRATOR, CASHIER, CUSTOMER
Person
{General}
IdPerson:Long
name:String
surname:String
dateOfBirth:Local Date
personType:PersonType
{Customer}
dateOfRegistration[0..1]:Local Date
address[0..1]:String
{Administrator}
experienceLevel[0..1]: String
{Cashier}
salaryBonus[0..1]:Double
createNewAdministrator(String name, String surname, LocalDate dateOfBirth, String experienceLevel):Person
createNewCustomer(String name, String surname, LocalDate dateOfBirth, String address):Person
createNewCashier(String name, String surname, LocalDate dateOfBirth, Double salaryBonus):Person
makeAdministrator(String experienceLevel);
makeCustomer(String address);
makeCashier(Double salaryBonus);
{Customer}
displayProductList(): List<Product>
addProductToOrder(Product product)
removeProductFromOrder(Product product)
placeOrder(deliveryType): Order order
{Administrator}
addDevice(Device device);
removeDevice(Long idDevice);
editDeviceDetails();
{Cashier}
registerOrder(String deliveryType): Order order
registerCustomer(String name, String surname, LocalDate dateofBirth, String address): Person customer
Salary bonus cannot be increased by more than 25% at one time
{ Bag }
Employment
idEmployment:Long
date:Local Date
salary:double
1
1..*
1..*
1
Store
IdStore:Long
monthlyIncome: float
monthlyExpenses: float
/totalMonthlyProfit:float
address:String
getStores(): List<Store> stores
{ Attribute }
GameVersion
IdGameVersion:Long
versionCode
gamingPlatform
Warehouse
IdWarehouse:Long
address:String
Client/Cashier cannot apply a PromoCode more than once per order
Order
IdOrder:Long
orderNumber:long
promoCode[0..1]:String
totalPrice:float
deliveryType:String
orderStatus:OrderStatus
usedPromocodes[1..*]: HashSet<String>
findByPromoCode(String promoCode): List<Order>
OrderStatus {Enum}
UNFINISHED, NEW, APPROVED, DENIED, IN_PROGRESS, DELIVERED
Product {Abstract}
IdProduct:Long
name:String
price:double
showDetails();
{polymorphic}
{ XOR }
ControllerType {Enum}
GAMEPAD, JOYSTICK
Game
gameGenre:ArrayList<String>
+knownGameGenres[1..*]: HashSet<String>
Device {Abstract}
operatingSystem[0..1]:String
serialNumber:String
findDevicesByStore(Store store): List<Device> devices
calculateDurabilityInYears(): double
{ Unique }
{ Multi-Aspect }
Controller
{General}
numOfKeys:int
controllerTypes: EnumSet<ControllerType>
{Gamepad}
numOfBacksideBtns:int
{Joystick}
stickType:String
{ Ordered }
{ Composition }
Publisher
IdPublisher:long
name:String
gameComparator: Comparator<Game>
{polymorphic}
{ Overlapping }
Wireless Controller
wireWidth:double
Wired Controller
connectionMethod:String
Stationary Computer
monitorIncluded:boolean
Console
bestTvCompatibility:String
Laptop
matrixType:String
Response time cannot be less than 0.1 or more than 10
Tablet
touchScreenReponseTime:double
<<interface>> ITablet
getTouchScreenResponseTime(): double
setTouchScreenResposneTime()
Benchmark
gpuOverall:String
cpuOverall:String
avgFps:double
{ Complex }
{ Multi }
LaptopTransformer
maxTransformAngle:double
0..*
0..1
1..*
1

5. The scenario for selected use-case - Remove device

5.1. Actors

Administrator

5.2. Purpose and Context

Administrator wants to remove device

5.3. Dependencies

5.3.1. Included use-cases

None

5.3.2. Extended use-cases

None

5.4. Assumptions and Pre-Conditions

1. Administrator pressed “Stores” button to open a list of stores
2. Administrator pressed “Devices” button to open a list of devices

Initiating business event

Administrator wants to remove a device from a specific store.

5.5. Basic flow of events

1. Administrator presses “Details” button on the device list page
2. Repository fetches information device details from the database
3. Application displays the page of device details
4. Administrator presses “Delete” button
5. Repository checks if the device is the last in the store
6. Repository deletes device from the database
7. Repository fetches device list information from the database
8. Application displays the page with updated list

5.6. Alternative flow of events

5.6.1. Administrator presses “back” button

4.a.1. Administrator is returned to the device list page

4.a.2. The use case ends

5.6.2. The device is the only product in the store

5.b.1. System shows notification error about the device being last in the store

5.b.2. The use case ends

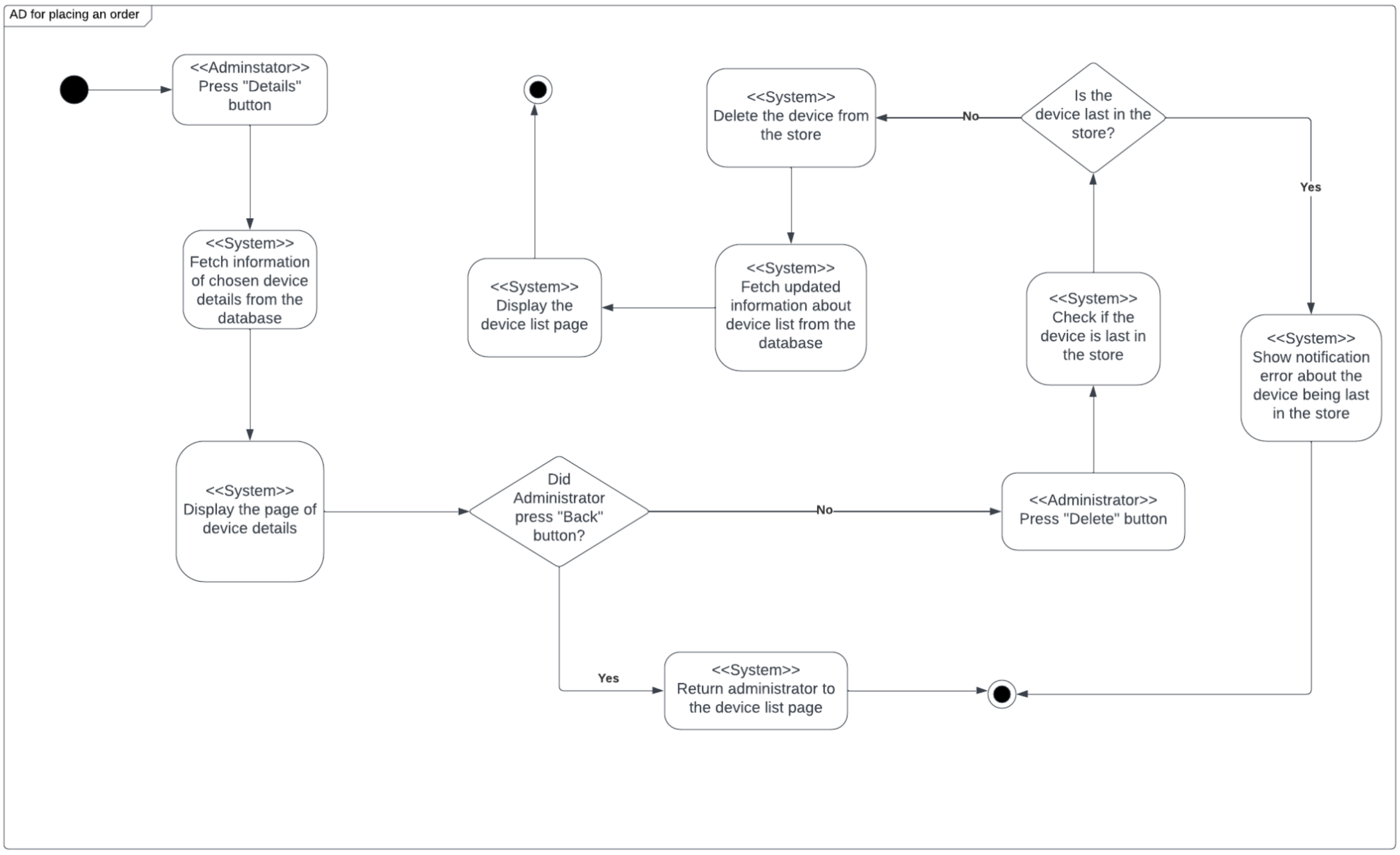
5.7. Extension points

Extension from “Display list of devices” use case

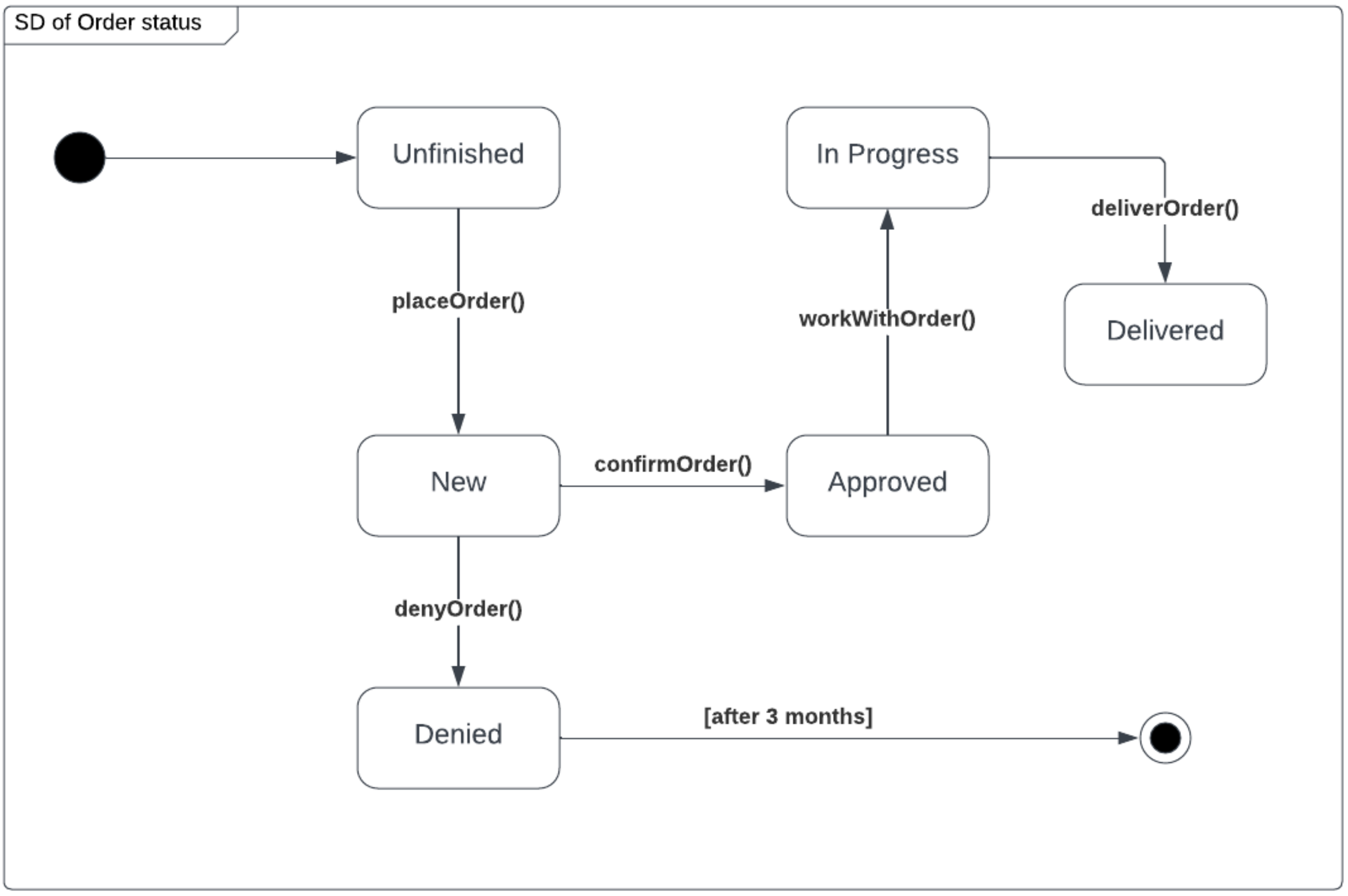
5.8. Post-Conditions

1. Administrator removed the device successfully
2. The device is deleted from the database

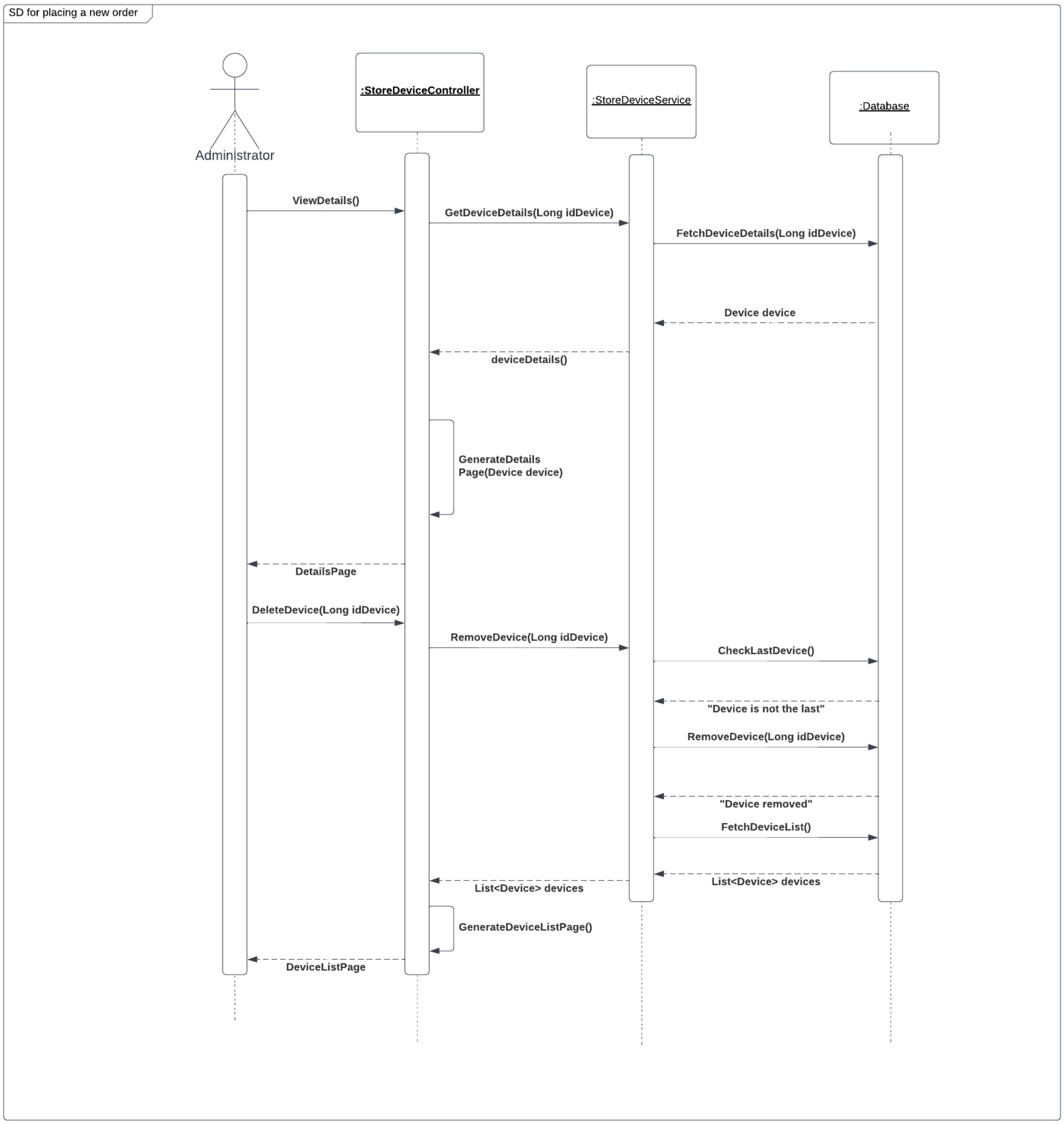
6. The activity diagram for selected use case


AD for placing an order
<<Adminstator>> Press "Details" button
<<System>> Fetch information of chosen device details from the database
<<System>> Display the page of device details
Did Administrator press "Back" button?
Yes
<<System>> Return administrator to the device list page
No
<<Administrator>> Press "Delete" button
<<System>> Check if the device is last in the store
Is the device last in the store?
No
<<System>> Delete the device from the store
<<System>> Fetch updated information about device list from the database
<<System>> Display the device list page
Yes
<<System>> Show notification error about the device being last in the store

7. The state diagram for selected class

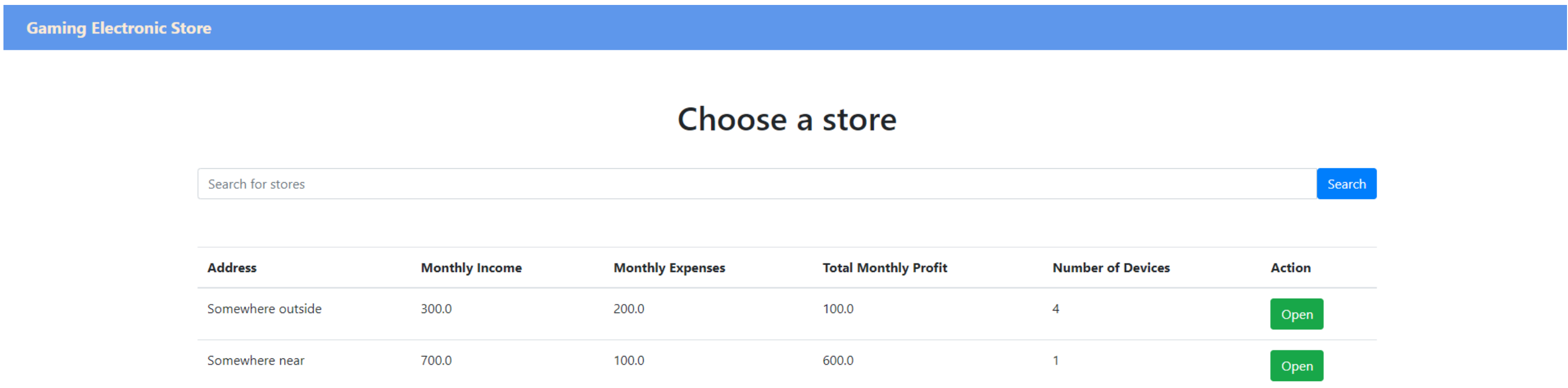


8. The sequence diagram for selected use case



9. The GUI Design

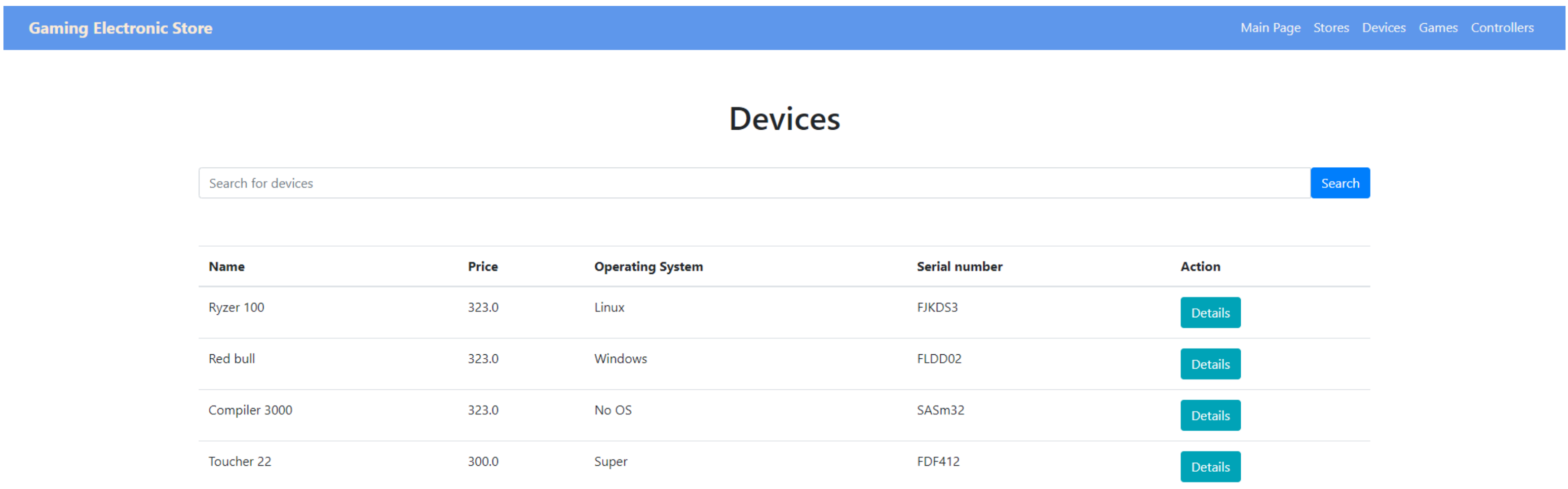
Page to choose store/main page



Choose a store

Address	Monthly Income	Monthly Expenses	Total Monthly Profit	Number of Devices	Action
Somewhere outside	300.0	200.0	100.0	4	Open
Somewhere near	700.0	100.0	600.0	1	Open

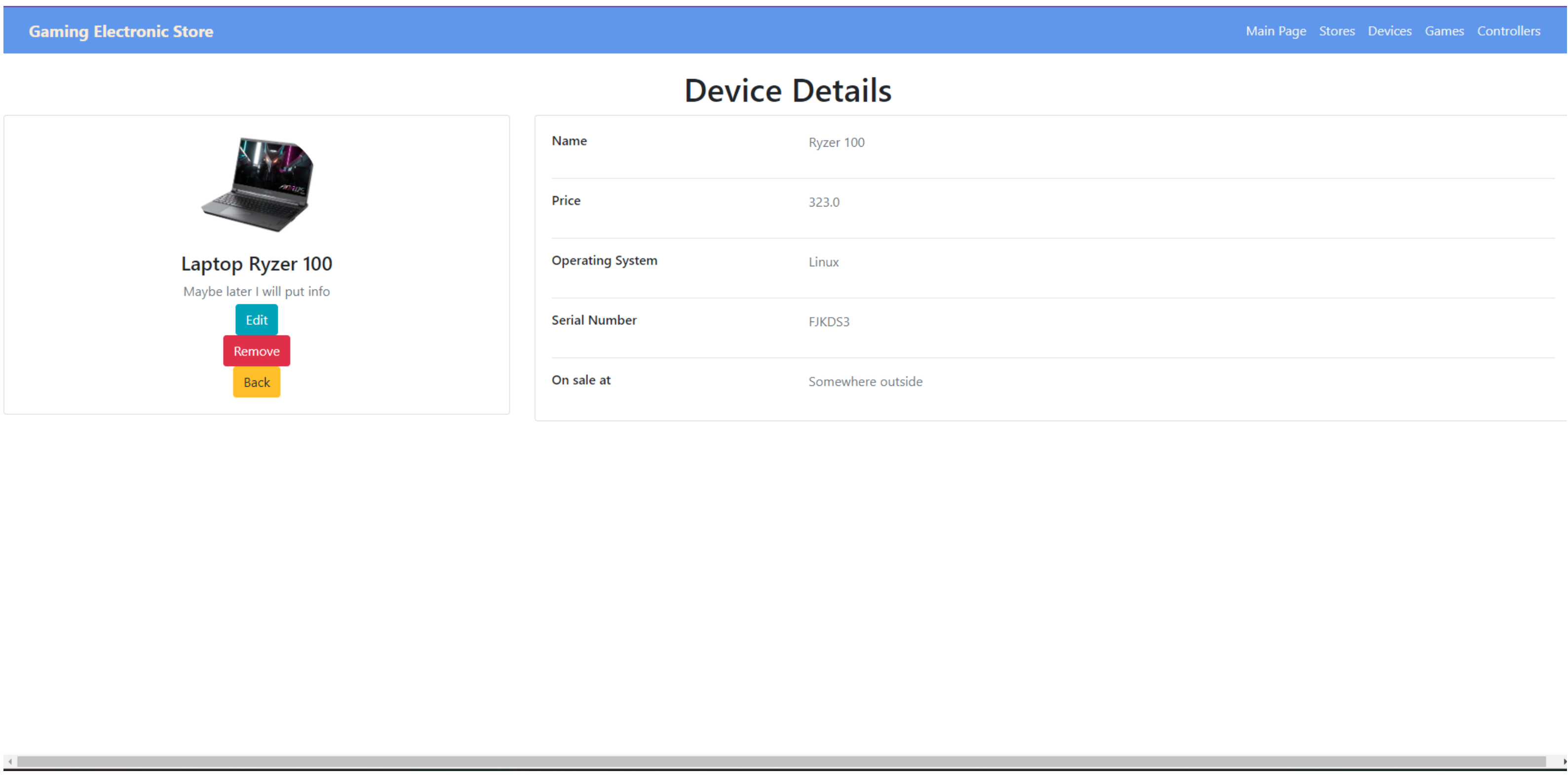
Device list of the chosen store



Devices

Name	Price	Operating System	Serial number	Action
Ryzer 100	323.0	Linux	FJKDS3	Details
Red bull	323.0	Windows	FLDD02	Details
Compiler 3000	323.0	No OS	SASm32	Details
Toucher 22	300.0	Super	FDF412	Details

Device details of the chosen device


Gaming Electronic Store

Main Page Stores Devices Games Controllers

Device Details

Laptop Ryzer 100

Maybe later I will put info

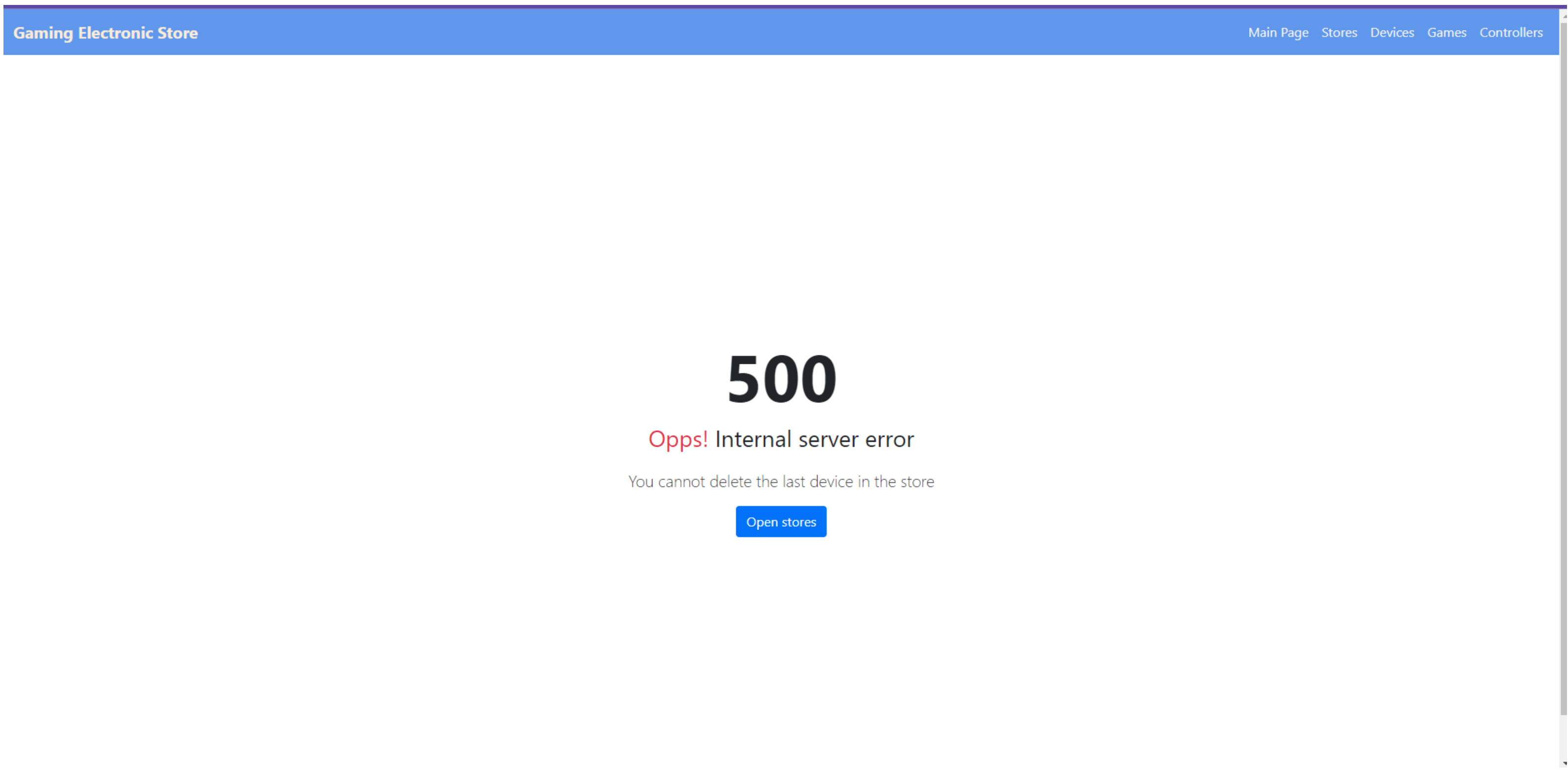
Edit

Remove

Back

Name	Ryzer 100
Price	323.0
Operating System	Linux
Serial Number	FJKDS3
On sale at	Somewhere outside

Error after attempt of device removal


Gaming Electronic Store

Main Page Stores Devices Games Controllers

500

Opps! Internal server error

You cannot delete the last device in the store

Open stores

10. The discussion of design decisions and the effect of dynamic analysis

Design decisions

Association with an attribute was made using the class GameVersion as a middle class in many to many relationship between Store and Game.

Qualified association is represented as a relation between Order and Store, where order list in Store class is sorted by the attribute orderNumber.

Composition association is represented as a relation between Game and Publisher, making Game a part of whole class Publisher. Game cannot exist without Publisher, and Publisher must have at least one game. It also has an **Ordered constraint**, where the list of games of the publisher is ordered using class attribute gameComparator: Comparator<Game>

For **Multi-Aspect inheritance**, two ways of class implementation are used: class Controller is inherited by 2 separate classes Wireless Controller and Wired Controller; flattening method – class Controller has an attribute controllerTypes of type Hashet<ControllerType>. Enum class ControllerType allows to set different types of Controller (Gamepad, Joystick), therefore setting inheriting class without creating extra classes. ControllerType distinction also represents **Overlapping inheritance**, where Controller can be of both types at one time. However, editing controllerType attribute of existing class is not permitted, since types cannot change.

Dynamic inheritance is represented as a Person class with an attribute personType of enum type PersonType, using flattening method. A person can be only of one type (CASHIER, CUSTOMER or ADMINISTRATOR) at one time. However, personType is editable, allowing to switch between inheriting classes by setting the attribute to a different enum value; therefore, removing previous specific class attributes and setting new ones.

Since it is not possible to directly implement **Multi-inheritance**, it was decided to use a different approach. Classes Laptop and Tablet are inherited by LaptopTransformer. It directly extends Laptop; however, in order to extend

Tablet, it implements interface ITablet instead, consisting of getTouchScreenResponseTime() and setTouchScreenResponseTime() methods, therefore indirectly extending class Tablet. Accordingly, class Tablet implements ITablet too.

Bag constraint was used for association "works in" between Person of type Cashier and Store. It is represented as a middle-class Employment between two classes with date and salary attributes. It allows to store duplicate associations between the same Cashier and Store by considering their employment.

XOR constraint was applied to two associations between Device to Store and Device to Warehouse. Device can be either stored in Warehouse or be on sale in Store. If one of associations is used when forming an association, then the other one should be empty (null). Device cannot have two empty associations; it always should have either one of them.

Custom business constraint is applied to promocode attribute in Order class. Promocode can be used only once per order and should be a valid promocode in a class attribute allPromocodes of type HashSet<String>.

Effect of dynamic analysis

Dynamic analysis resulted in forming two alternative flows in selected use-case "**Remove device**":

1) Administrator may go back to the device list page by pressing the "Back" button. After that, it displays a list of devices without changes
2) Administrator attempts to remove a last device from the chosen store. Then, it should display a notification error about the last device in the store.

The GUI templates were made using Bootstrap, and then implemented in the final version of the project.

Done by Maraimbekov Azamat, group 15c, s22850