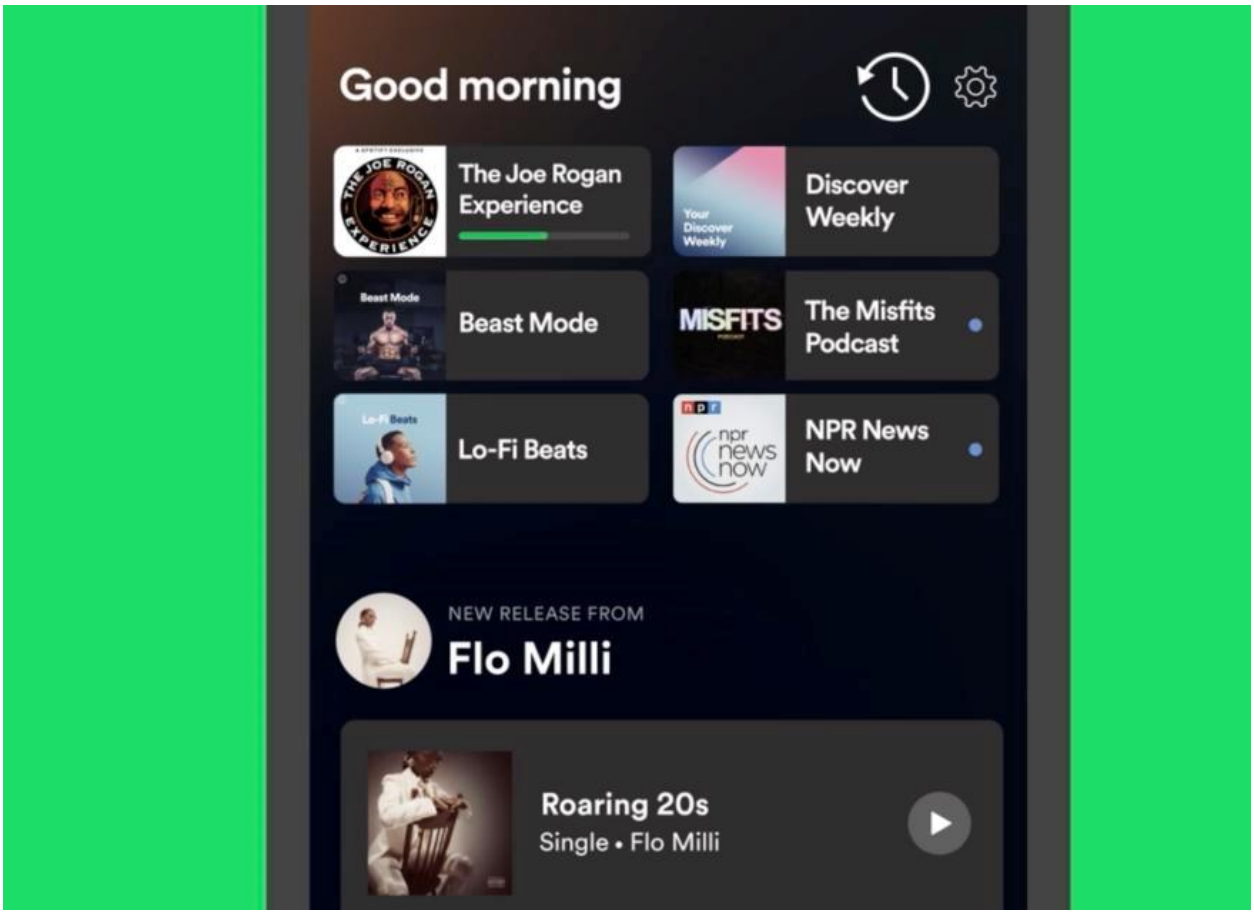
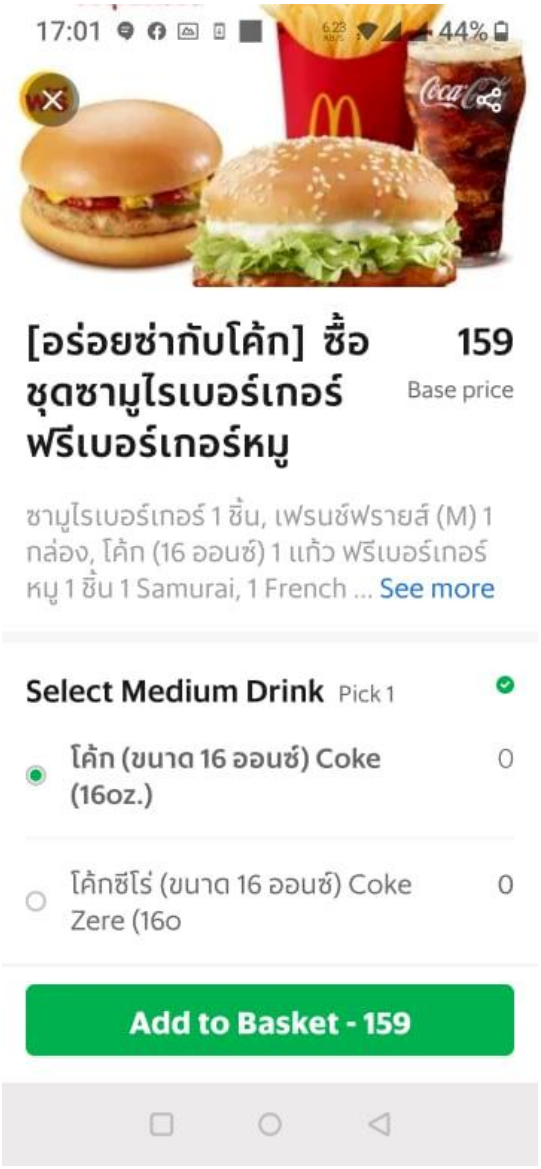
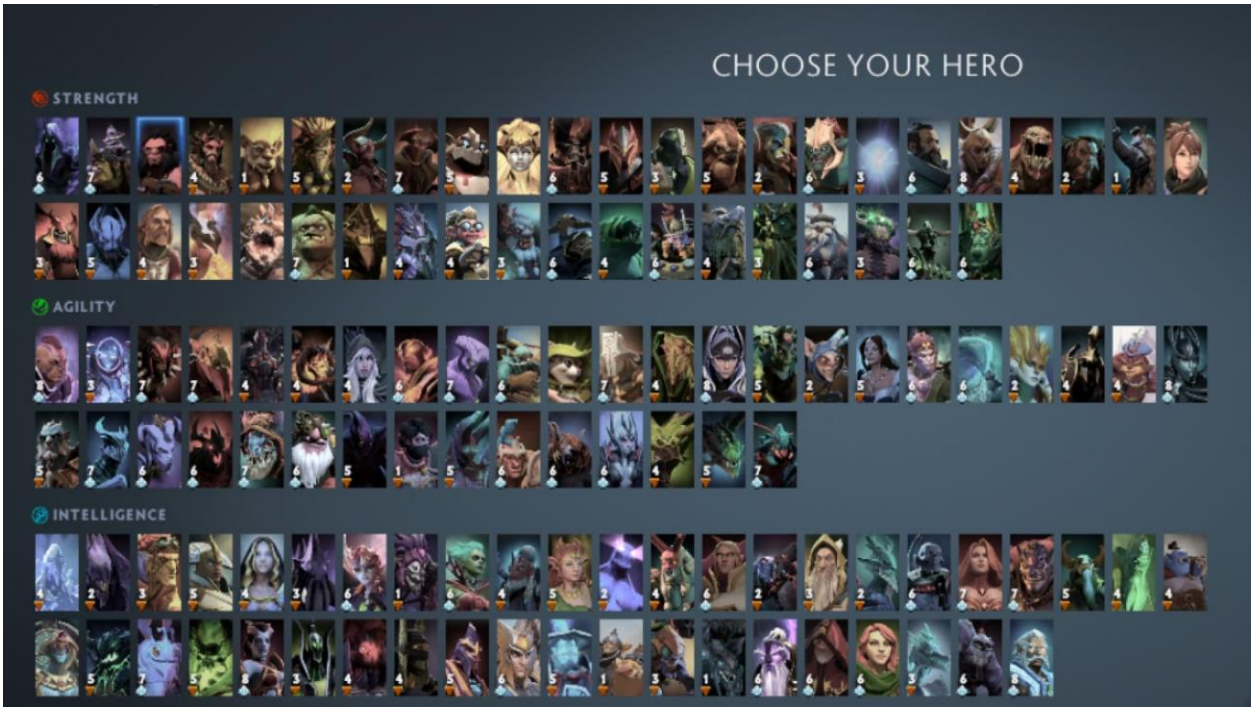
Adapter: Imagine that you are building a music streaming application that uses different APIs to retrieve song information from various sources. One of the APIs you are using returns song titles in uppercase, while the others return them in lowercase. To present a consistent user interface, you could use an adapter pattern to convert the song titles from uppercase to lowercase (or vice versa) before displaying them to the user.



Decorator: Suppose you are building an e-commerce website that allows customers to customize their orders with add-ons such as gift wrapping, personalized messages, or expedited shipping. Instead of creating separate classes for each possible combination of features, you could use a decorator pattern to dynamically add these features to the order object at runtime. For example, you could create a GiftWrappedOrderDecorator class that adds gift wrapping to an existing order, or a PersonalizedMessageOrderDecorator class that adds a personalized message to an existing order.



Factory method: Consider a scenario where you are developing a game that allows players to choose different characters to play as. Each character has a unique set of abilities and attributes, and you want to make it easy to add new characters in the future. You could use a factory method pattern to create instances of each character class. For example, you could create a CharacterFactory that has methods for creating instances of each character class, such as createStrength() or createAgility(). This would allow you to add new character classes to the game simply by adding a new method to the factory.



Singleton: Suppose you are building a museum exhibit that displays a rare and valuable artifact, such as an ancient manuscript. You want to ensure that the artifact is kept secure and that only one person can view it at a time to prevent damage or theft. You could use a singleton pattern to ensure that only one person at a time can view the artifact.

Here's how it might work:

1.The exhibit has a single room that houses the artifact, with a security guard stationed at the entrance to the room.

2.Visitors are only allowed to enter the room one at a time. When a visitor wants to view the artifact, they must go to the entrance of the room and request permission from the security guard.

3.The security guard checks to see if anyone is currently viewing the artifact. If no one is currently viewing it, the security guard grants permission to the visitor and opens the door to the room.

4.The visitor is then allowed to enter the room and view the artifact for a limited amount of time, such as 5 minutes. A timer is set to ensure that the visitor does not exceed the time limit.

When the visitor is finished viewing the artifact, they must leave the room and return to the entrance, at which point the security guard can grant permission to the next visitor.

Interning: Imagine that you are building a spell-checking application that processes large volumes of text. As part of the processing, the application needs to identify common words and phrases in the text. To improve performance, you could use an interning pattern to store frequently-used words and phrases in a cache, so that they can be quickly looked up and reused as needed. This would help to speed up the processing of the text and reduce memory usage.