Do the design and implementation follow design principles?

+ The code follows MVC properly which is an application of Separation of Concern.
+ Open-closed principle are used in classes which are open for extension and closed for modification using private variables and fields.
+ The Single Responsibility Principle is used in many of the smaller classes which contains only one method like ProductJsonFileParser.
+ The Interface Segregation Principle is used commonly. For example in the package favorites where they have public interface IProductListIO which is implemented by classes as ProductListFileIO. They also use interfaces for IProductParser.

- At some places like the controller they break against The Law of Demeter which has the rule "only talk to friends". Used to reduce coupling between classes or objects.
-

Does the project use a consistent coding style?

+ Yes the project follows a consistent coding convention.
+ Yes, code chunks are well-organised. The comments are found above the code.
+ CamelCases har commonly used throughout the project.
+ Clear dividing between packages with interfaces implemented by classes.

Is the code reusable?

+ Yes, because it is well structured in the way that it is well separated.
+ The use of interfaces makes the code easy to separate.

Is it easy to maintain?

+ Yes, as the code is easy to read and structured.
+ Because all components are separated from each other, they are easy to change.
+ Also the dependencies of the modules are well thought out.

Can we easily add/remove functionality?

+ Yes, as the code is easy to read and structured.
+ Also the dependencies of the modules are well thought out.
+ Classes and functions are generally small.
+ The extended usage and implementation of factory pattern makes it easier to add new functionality, as new factory methods can be added to create new objects, without having to change the code in too many places.

Are design patterns used?

+ The model module uses the facade pattern to mask and simplify the backend.
+ As it is listed in the provided SDD, the application implements a handful of design patterns such as facade pattern which is used to put a facade over the rest of the code.

Is the code documented?

+ Some code is well-documented using javadocs and some code has no documentation. If the rest of the code i as well-documented as the parts that has been commented,

Are proper names used?

+ Yes, the naming is mostly fine and the names files give an understanding of what the files contain, with a proper use of camelcase.

Is the design modular? Are there any unnecessary dependencies?

+ The code is highly separated into independent modules. Almost every package has a public interface to hide the internal details of the package. Modularity also means that the code is easy to maintain, refactor and test.
+ The program is modular because it follows an MVC architecture without circular or other unnecessary dependencies.

Does the code use proper abstractions?

+ The command package is well written, as it is easy to remove/add new commands and incorporate them into the design without having to rewrite a lot of code.
+ Good use of interfaces to make the code more abstract.

Is the code well tested?

+ Yes. There are plenty of tests (especially for model packages) that cover the most essential parts of the code.

Are there any security problems, are there any performance issues?

+ Favorites are working

- Images are loading slow, gets image from Systembolaget on every part of the navigation
- Tests weren't working properly
- The FileReader and FileWriter constructors instantiate, thats causing issues with garbage collection when the finalized methods are called.
- Instead of new FileWriter(fileName) use Files.newBufferedWriter(Paths.get(fileName))

Is the code easy to understand? Does it have an MVC structure, and is the model isolated from the other parts?

+ Yes, the code is distributed in different folders containing the different parts that make up the structure.
+ It seems that the code makes use of a MVC pattern.

Can the design or code be improved? Are there better solutions?

+ The code is well commented and follows a common style.
+ Naming of packages and classes is at times clear.

- The view is fixed in size. If the user had a different sized window or changed window size, the view would remain the same.
- It's possible to implement an Observer-pattern into the code to notify multiple objects about events that happen to the object that they are observing.