Sprawozdanie z Projektu z przedmiotu Event Driven Programming

Prowadzący: por. mgr inż. Michał Sobolewski

Przygotował: Mateusz Gajda

Grupa: WCY20IJ1S1

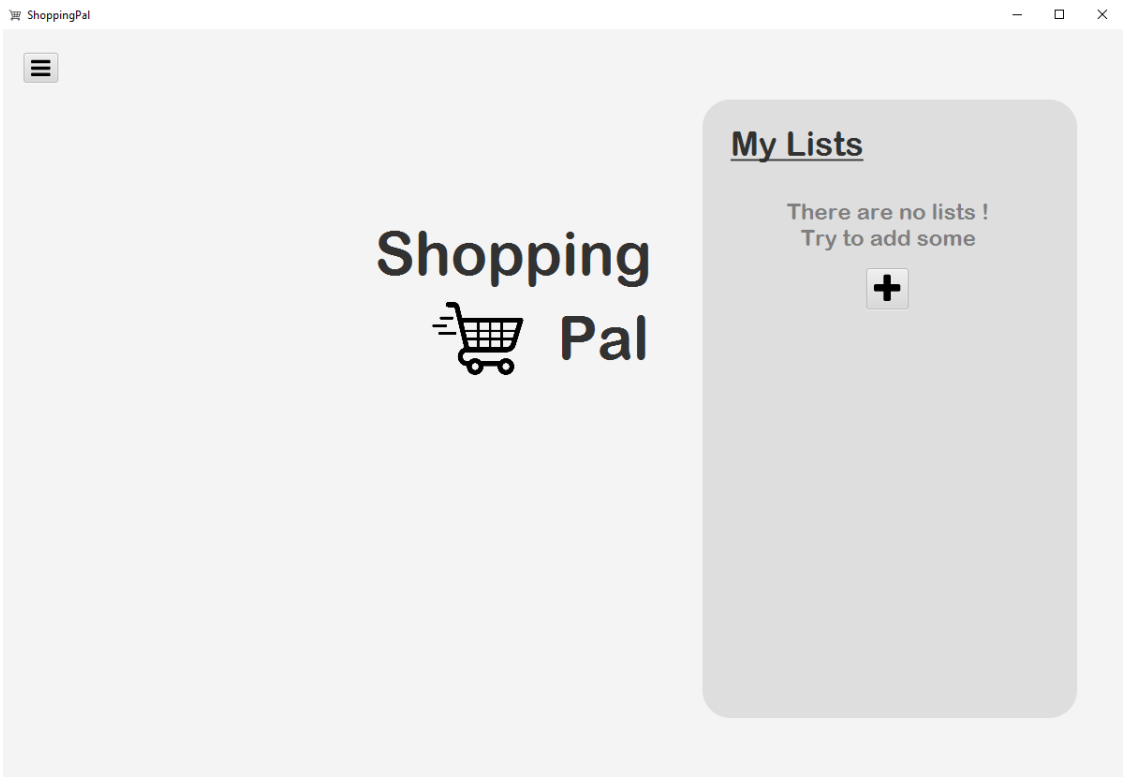
Krótki Opis Aplikacji:

ShoppingPal to aplikacja desktopowa, która umożliwia tworzenie własnych list na zakupy. Użytkownik może tworzyć i usuwać własne listy, dodawać i usuwać do nich własne produkty lub wyszukać produkty z Open Food Facts API. Aplikacja dodatkowo liczy ile mniej więcej będą kosztowały zakupy na podstawie takiej listy. Została ona napisana w języku Java a GUI zostało zaprojektowane w Scene Builderze.

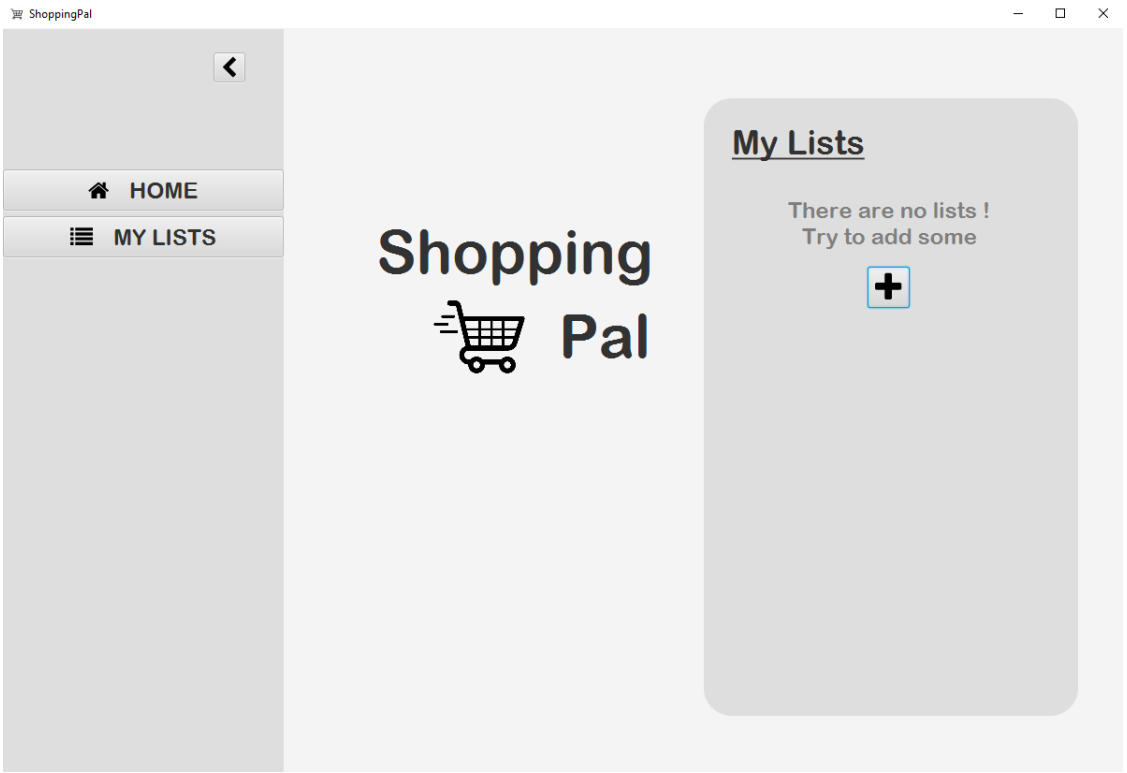
Opis wykorzystanych Technologii:

- JavaFX – technologia umożliwiająca tworzenie i wdrażanie aplikacji desktopowych. Umożliwia tworzenie dobrze wyglądającego GUI poprzez rozszerzenie technologii Java. Cały projekt został utworzony na bazie tej technologii
- MongoDB – baza danych typu NoSQL, umożliwiająca przechowywanie i pobieranie danych w formacie dokumentów JSON. W tym projekcie technologia ta została wykorzystana w przypadku tworzenia bazy danych list z zakupami utworzonymi przez użytkownika. Przetrzymuje ona takie wartości jak nazwa listy oraz produkty które użytkownik do niej dodał (wraz z ich atrybutami tj. ilość, cena oraz nazwa produktu)
- Scene Builder – narzędzie które umożliwia projektowanie GUI. Całe GUI projektu zostało zaprojektowane w tym narzędziu
- Maven – narzędzie automatyzujące budowę oprogramowania na platformę Java. Cały Build System tego projektu jest oparty na tym narzędziu.
- Apache Commons - biblioteka napisana w Javie, która zawiera wiele przydatnych klas i metod pomocniczych, które ułatwiają programowanie i zwiększają wydajność tworzonych aplikacji. W tym projekcie została użyta do edycji zmiennych typu String.
- Google Guava - to zestaw wspólnych bibliotek typu open source dla Javy. W tym projecie została ona wykorzystana do stowrzenia „Obserwatora” w postaci EventBus, który w zależności od zdarzenia zmienia interfejs graficzny

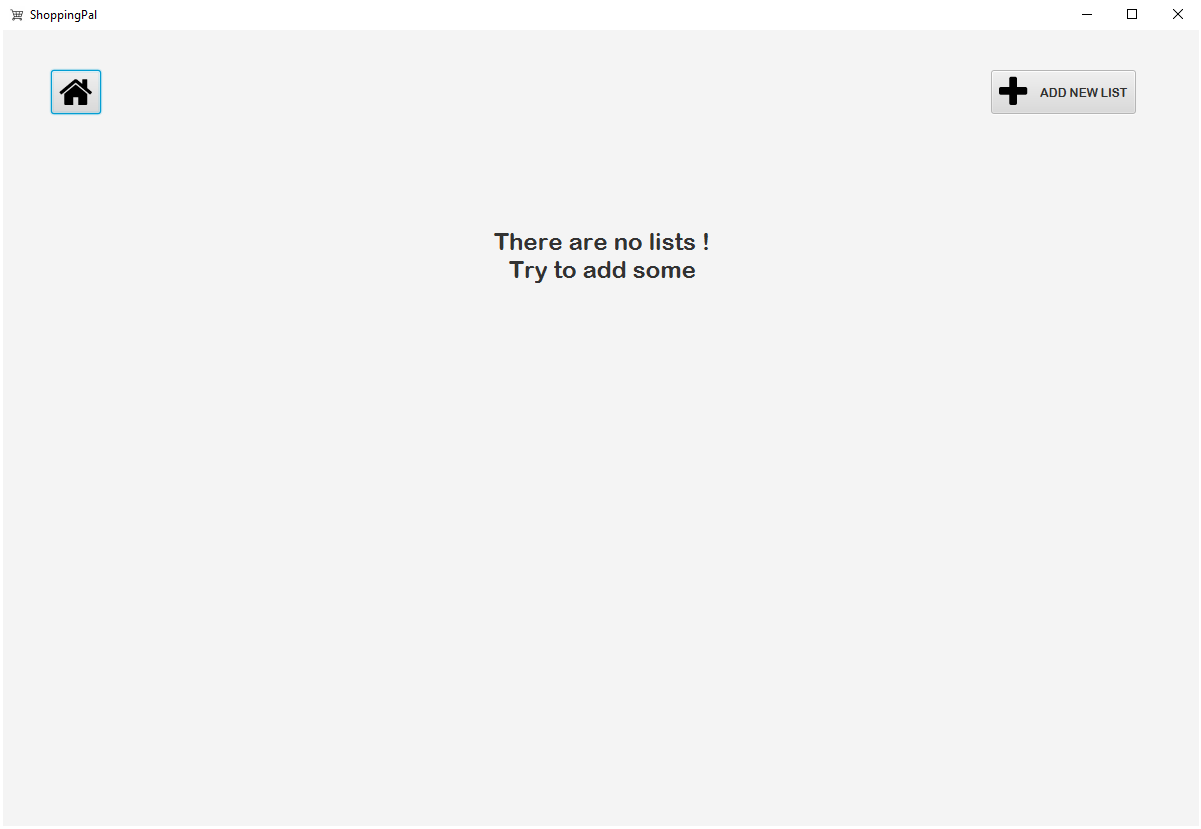
Screeny z Aplikacji:



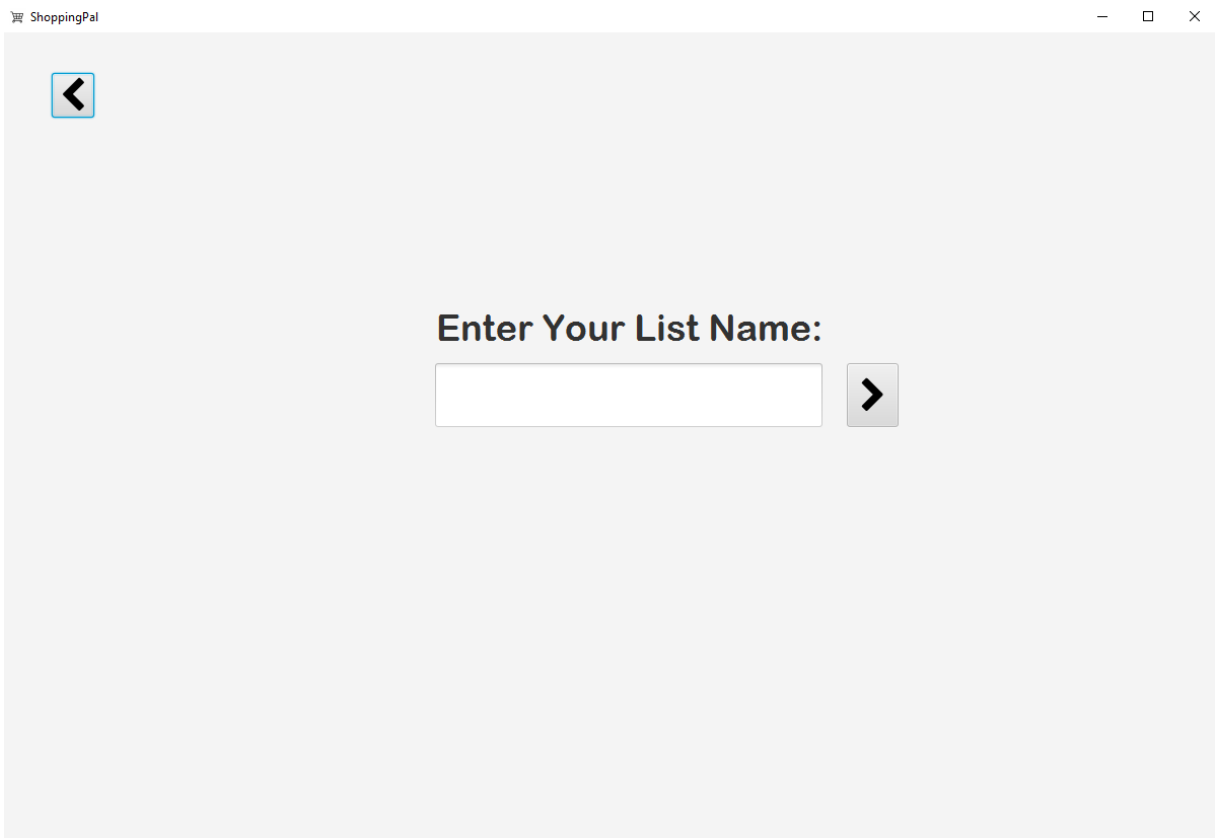
Rysunek 1 – HomeScreen bez list i bez menu



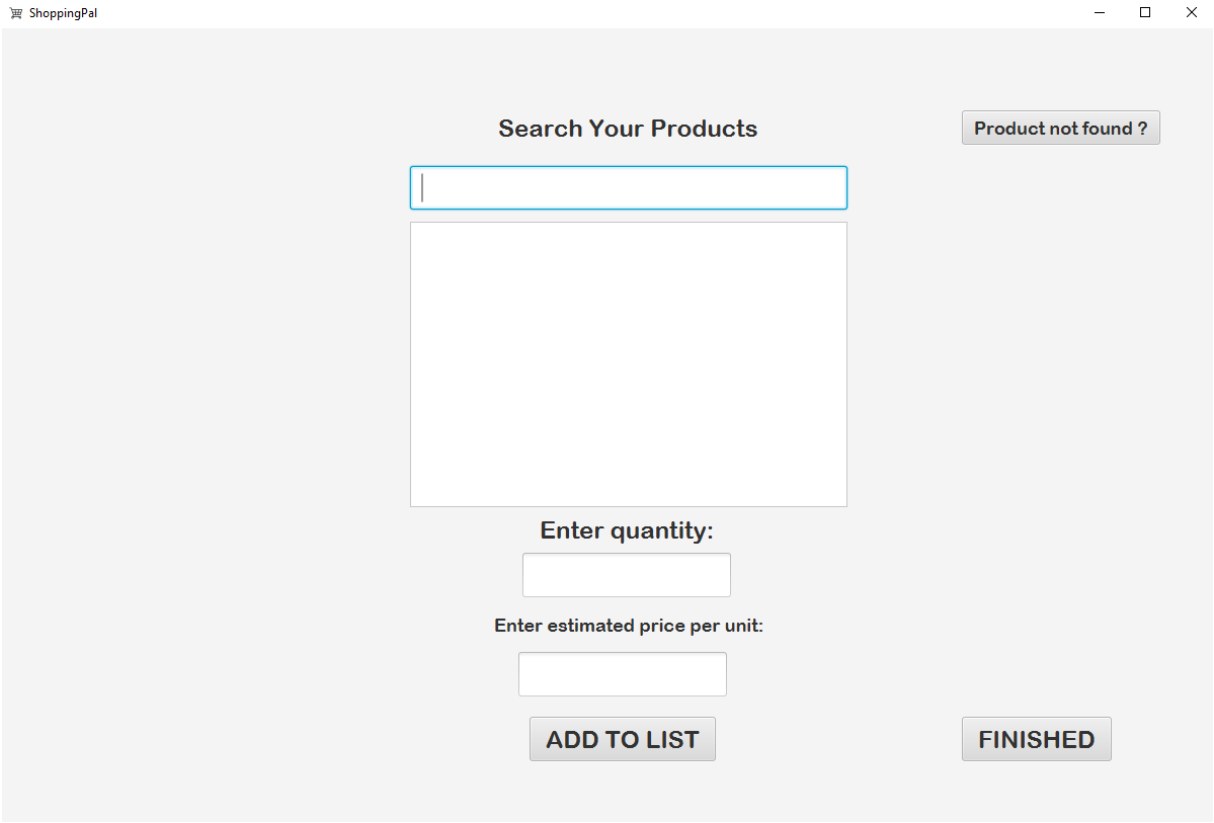
Rysunek 2 – HomeScreen bez list i z menu



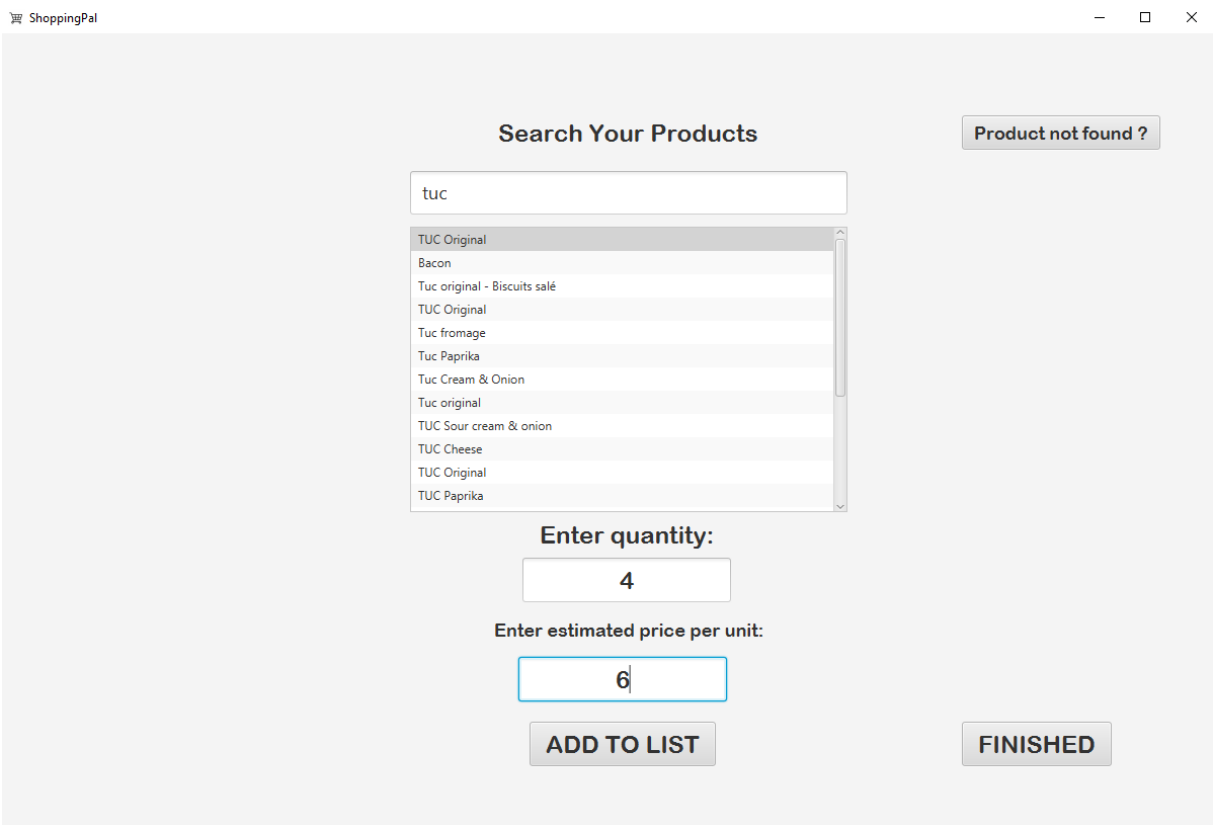
Rysunek 3 - Zakładka MyList bez zadnych list



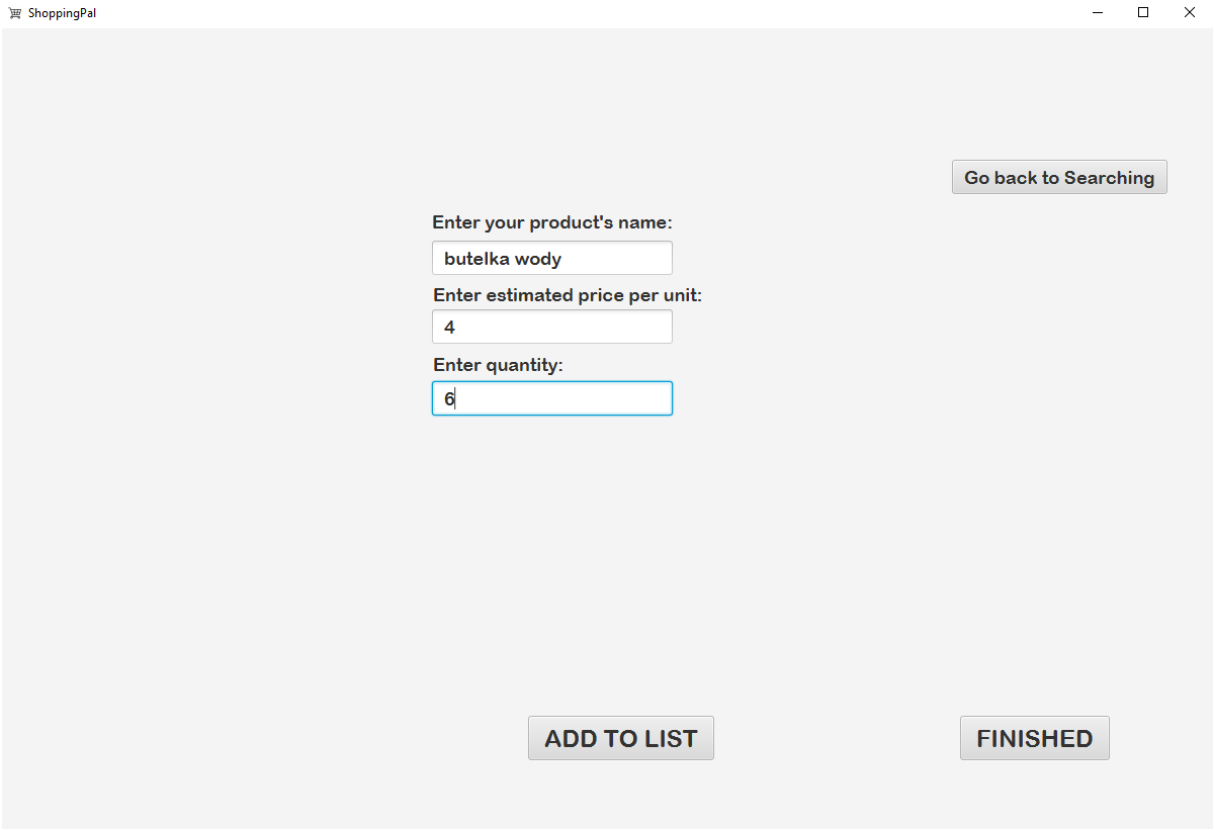
Rysunek 4 - Widok tworzenia listy



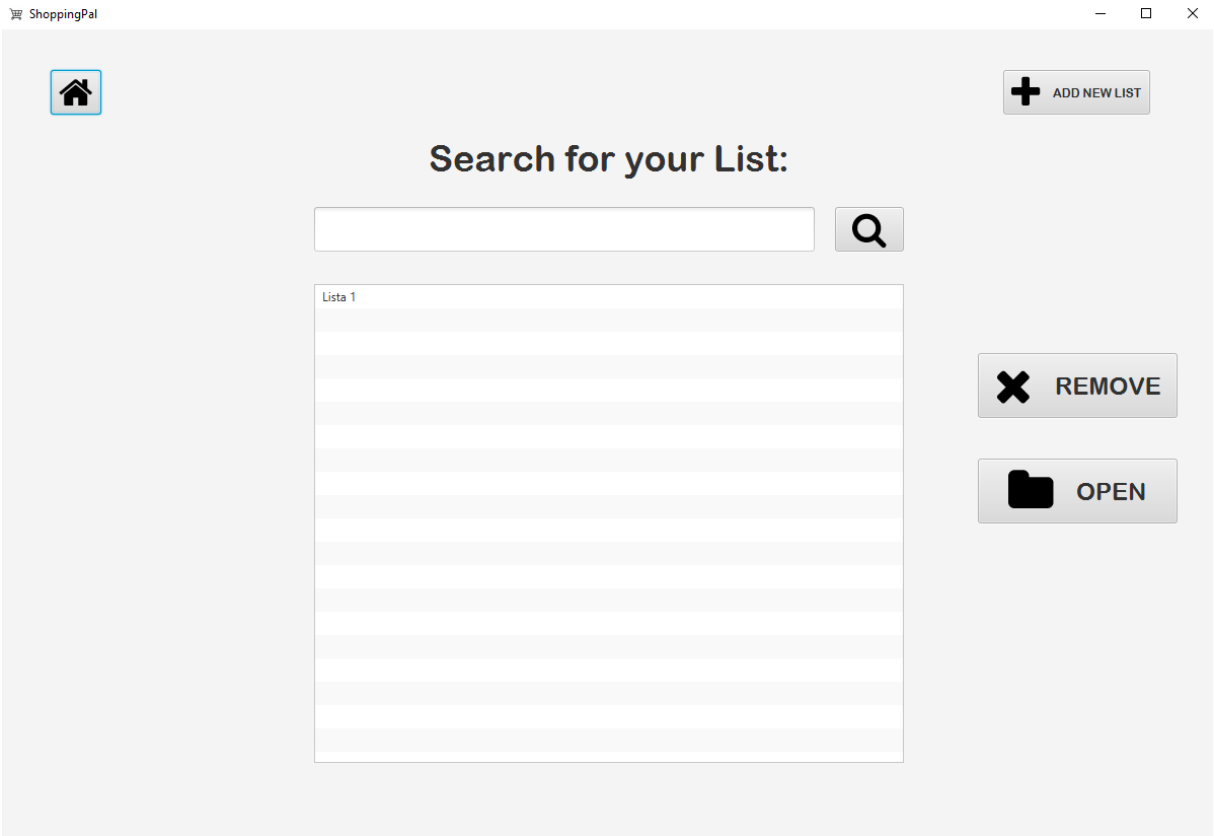
Rysunek 5 - Szukanie produktów



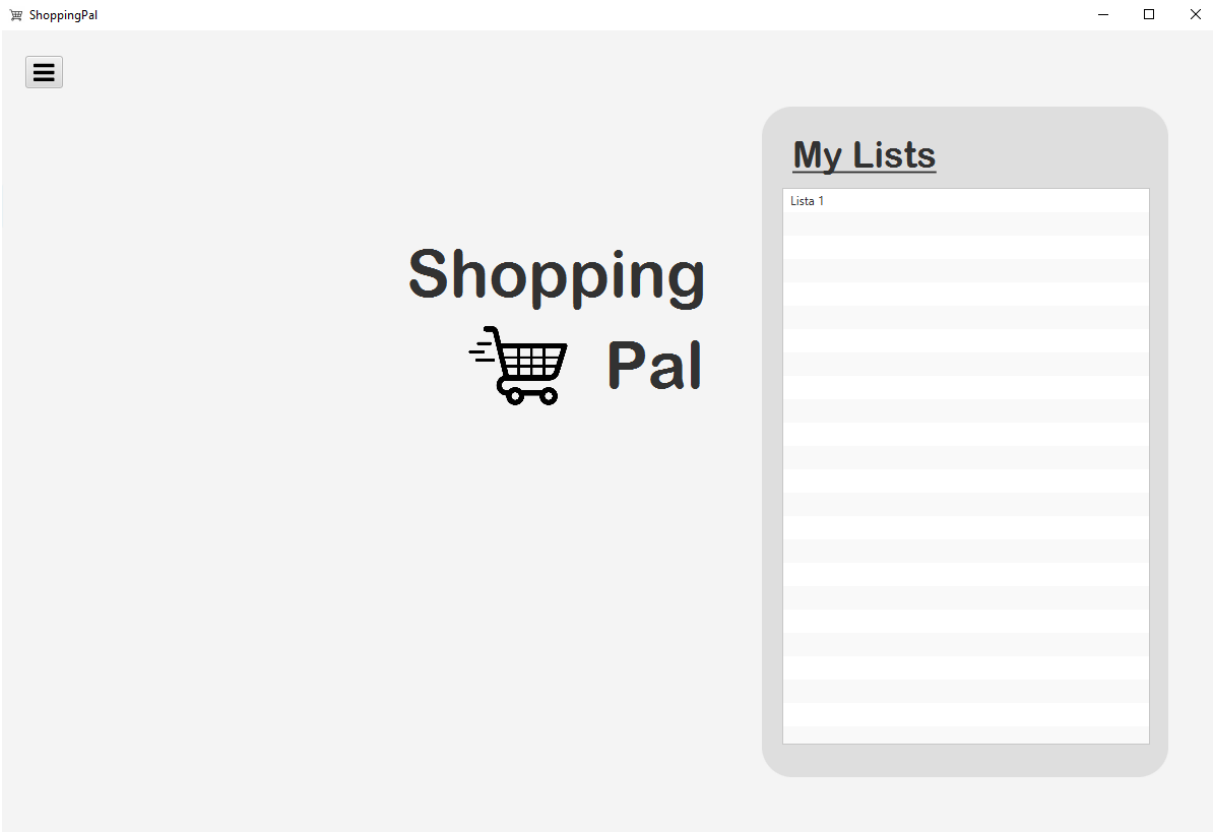
Rysunek 6 - Szukanie produktów



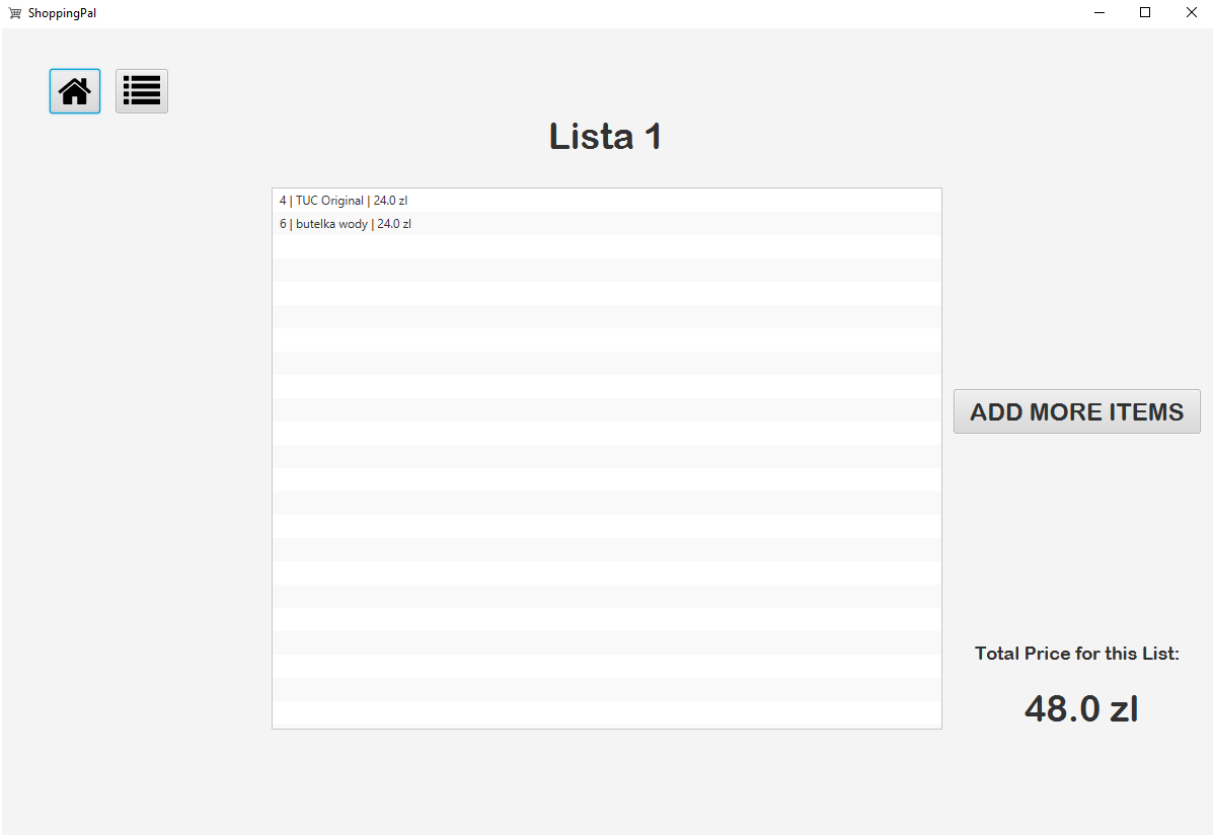
Rysunek 7 - Dodanie własnego produktu



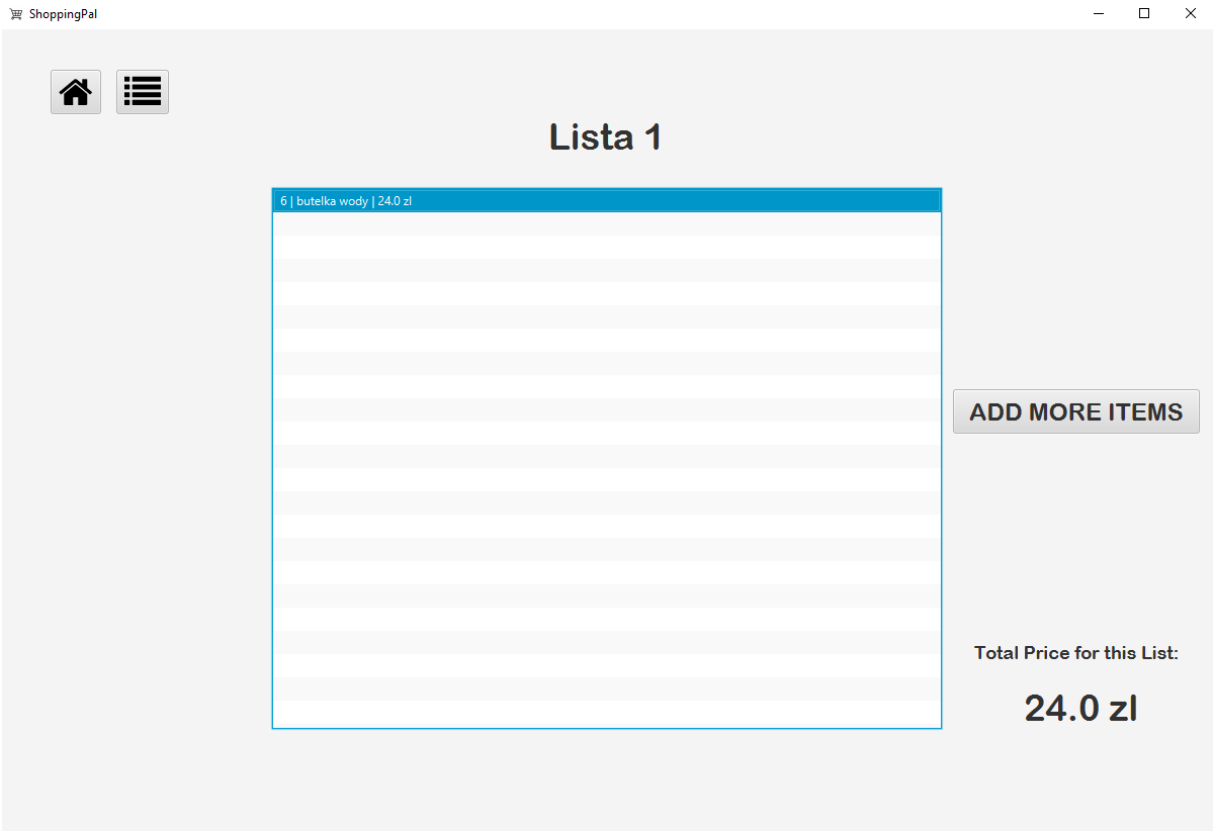
Rysunek 8 - Widok MyList po dodaniu listy



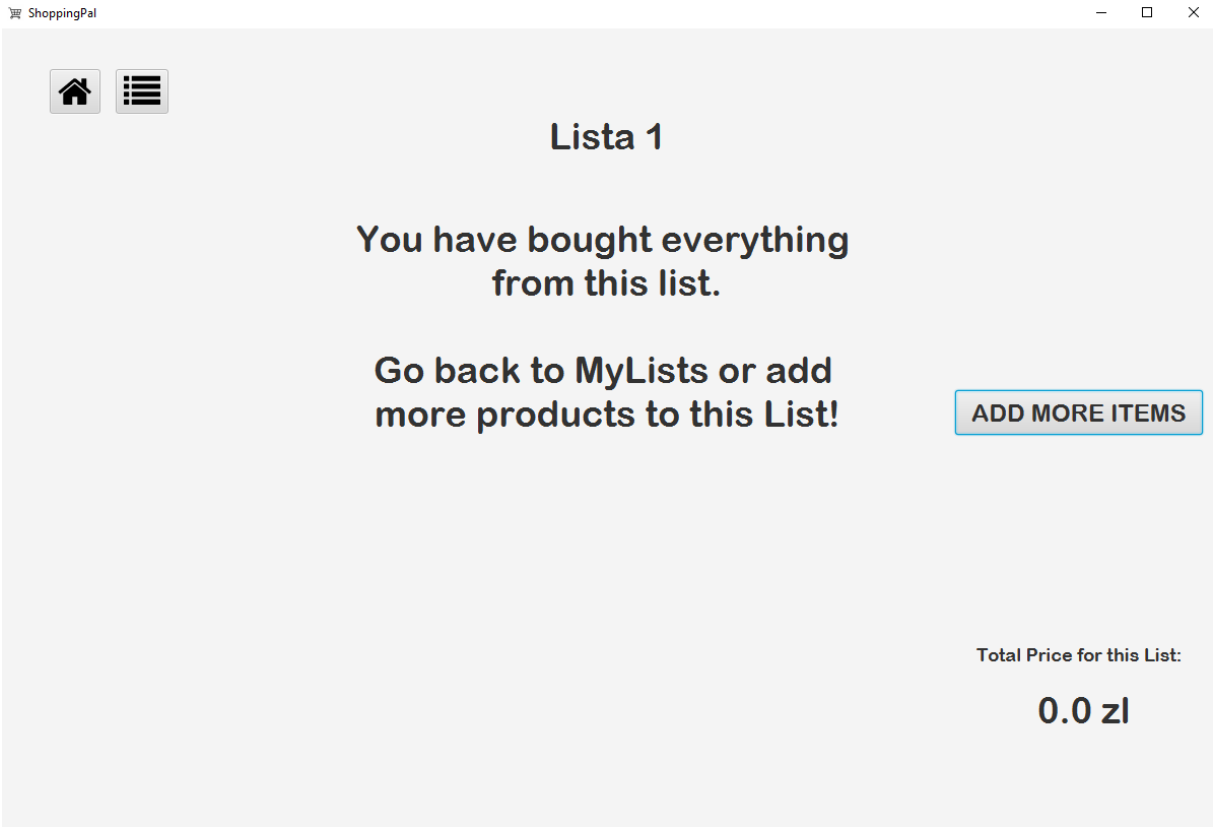
Rysunek 9 - HomeScreen po dodaniu listy



Rysunek 10 - Wejście w wybrana liste



Rysunek 11 - Usuwanie jednego produktu



Rysunek 12 - Usuwanie wszystkich produktów

Wykorzystane biblioteki i sposób ich zaimportowania do Projektu:

- Biblioteki MongoDB - com.mongodb, org.mongodb
- Podstawowe biblioteki Java – java.io, java.util, java.net
- Biblioteki JavaFX – javafx
- Biblioteki Google – com.google. guava
- Biblioteki Apache Commons - org.apache.commons
- Biblioteki JSON - org.json

Wszystkie biblioteki zostały dodane do pliku pom.xml jako dependencies i zaimportowane do Projektu przez zależność Mavenową.

Dokładne wszystkie bilbioteki w pliku pom.xml:

```
<dependencies>
    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-controls</artifactId>
        <version>20-ea+4</version>
    </dependency>
    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-fxml</artifactId>
        <version>20-ea+4</version>
    </dependency>
    <dependency>
        <groupId>org.controlsfx</groupId>
        <artifactId>controlsfx</artifactId>
        <version>11.1.2</version>
    </dependency>
    <dependency>
        <groupId>org.kordamp.bootstrapfx</groupId>
        <artifactId>bootstrapfx-core</artifactId>
        <version>0.4.0</version>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>${junit.version}</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <version>${junit.version}</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.mongodb</groupId>
        <artifactId>mongo-java-driver</artifactId>
        <version>3.12.11</version>
    </dependency>
    <dependency>
        <groupId>org.mongodb</groupId>
        <artifactId>bson</artifactId>
        <version>3.12.11</version>
    </dependency>
    <dependency>
```

```
        <groupId>org.json</groupId>
        <artifactId>json</artifactId>
        <version>20201115</version>
    </dependency>
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-lang3</artifactId>
        <version>3.12.0</version>
    </dependency>
    <dependency>
        <groupId>com.google.guava</groupId>
        <artifactId>guava</artifactId>
        <version>30.1.1-jre</version>
    </dependency>
</dependencies>
```

Dodatkową biblioteką którą musiałem dodać oddzielnie to biblioteka **fontawesomefx-8.2**. Jest to biblioteka do specjalnych ikon dla Scene Buildera. Dodawałem ją w sposób następujący:

Zakładka File -> Project Structure -> Project Settings -> Libraries -> Dodanie biblioteki poprzez wciśnięcie przycisku z plusem -> Java -> podałem ścieżkę do bilbioteki(biblioteka znajduje się w folderze projektu w pod folderze src) -> Apply

Komponenty/Strony na których się wzorowano w implementacji:

- https://www.youtube.com/watch?v=9XJicRt_FaI&ab_channel=BroCode

Dzięki temu poradnikowi udało mi się obeznać ze Scene Builderem i JavaFX

- https://www.youtube.com/watch?v=VUVqamT8Npc&ab_channel=Randomcode

Poradnik ten pokazuje i naprowadza jak utworzyć Search Bar który posiadam w swoim projekcie

- https://www.youtube.com/watch?v=zXQJpxeEAGs&ab_channel=Ken
- https://genuinecoder.com/javafx-observable-list-tutorial/
- https://docs.oracle.com/javase/8/javafx/api/javafx/collections/ObservableList.html

Strony te pokazały jak obsługiwać ObsevableList która ułatwiła mi ustawianie wartości w ListView

- https://stackoverflow.com/questions/10949461/javafx-2-click-and-double-click

Z tej strony korzystałem w przypadku kiedy chciałem utworzyć event na podwójne wciśnięcie elementu z ListView

- https://openfoodfacts.github.io/api-documentation/
- https://www.youtube.com/watch?v=zZoboXqsCNw&ab_channel=Randomcode
- https://www.digitalocean.com/community/tutorials/java-httpurlconnection-example-java-http-request-get-post
- https://rapidapi.com/blog/how-to-use-an-api-with-java/
- https://docs.oracle.com/javaee/7/api/javax/json/JsonObject.html
- https://www.baeldung.com/java-org-json

Strony te pokazały mi jak połączyć się z API i jak pobrać dane JSON z takiego API i wybrać interesujące mnie dane

- https://www.baeldung.com/java-stream-filter-lambda
- https://www.geeksforgeeks.org/stream-filter-java-examples/
- https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html

Strony te pokazały mi jak łatwo można iterować, tworzyć nowe listy, filtrować i przetwarzać listy czy tablice co ułatwiło w pobieraniu niektórych danych.

- https://www.baeldung.com/java-asynchronous-programming
- https://www.baeldung.com/java-completablefuture
- https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html
- https://reflectoring.io/java-completablefuture/

Strony te pokazały mi jak korzystać z obiektów CompletableFuture i jak zwiększyć wydajność aplikacji poprzez programowanie asynchroniczne.

- https://guava.dev/releases/22.0/api/docs/com/google/common/eventbus/EventBus.html
- https://levelup.gitconnected.com/eventbus-in-java-publish-subscribe-to-the-rescue-ca0505cb53b1
- https://codentrick.com/observer-pattern-in-mobile-eventbus-and-notificationcenter/

Strony te pokazały mi jak skorzystać z EventBus jako obserwatora dzięki którem w zależności od zmiany na liście zmieniał się też interfejs graficzny

- https://www.javatpoint.com/multithreading-in-java
- https://jenkov.com/tutorials/java-concurrency/index.html
- https://www.baeldung.com/java-concurrency
- https://www.javatpoint.com/java-executorservice
- https://www.baeldung.com/java-executor-service-tutorial
- https://www.baeldung.com/java-single-thread-executor-service
- https://medium.com/javarevisited/java-multithreading-using-executor-service-ebffd18b00b6
- https://dzone.com/articles/java-concurrency-multi-threading-with-executorserv
- https://stackoverflow.com/questions/27033976/java-8-lambda-ambiguous-runnable-and-callback
- https://stackoverflow.com/questions/13784333/platform-runlater-and-task-in-javafx
- https://riptutorial.com/javafx/example/7291/updating-the-ui-using-platform-runlater
- https://jenkov.com/tutorials/javafx/concurrency.html

Strony te pokazały mi jak skorzystać z wielowątkowości i executorService w swoim projecie. Zostało to przeze mnie wykorzystane w momencie wyszukiwania produktów z API tak aby program w tle z każdym wpisanym przez użytkownika znakiem wyszukiwał odpowiednie produkty i w czasie rzeczywistym wyświetlał to na interfejsie graficznym.

- https://www.mongodb.com/developer/languages/java/java-setup-crud-operations/
- https://www.baeldung.com/java-mongodb
- https://stackoverflow.com/questions/15436542/mongodb-java-push-into-array
- https://www.mongodb.com/docs/drivers/java/sync/current/usage-examples/insertOne/
- https://www.tabnine.com/code/java/methods/com.mongodb.client.MongoCollection/findOneAndUpdate
- https://kb.objectrocket.com/mongo-db/how-to-iterate-through-mongodb-query-results-through-a-function-using-java-388
- https://stackoverflow.com/questions/22875150/mongodb-java-pull
- https://www.mongodb.com/docs/manual/reference/operator/update/pull/
- https://mongodb.github.io/mongo-java-driver/4.1/apidocs/bson/org/bson/Document.html
- https://stackoverflow.com/questions/49486141/mongodb-java-driver-aggregation-with-regex-filter
- https://www.tabnine.com/code/java/methods/com.mongodb.client.model.Filters/regex

Strony te pokazały mi jak poruszać się po bazie danych MongoDB i jak pobierać z niej interesujące mnie dane. W projekcie tym używam tej bazy danych do tworzenia własnych list z zakupami i w zależności od funkcji pobieram z niej odpowiednie dane.

- https://www.baeldung.com/java-properties
- https://crunchify.com/java-properties-file-how-to-read-config-properties-values-in-java/
- https://mkyong.com/java/java-properties-file-examples/
- https://medium.com/@sonaldwivedi/how-to-read-config-properties-file-in-java-6a501dc96b25
- https://www.youtube.com/watch?v=bljA8dpfWeQ&ab_channel=AutomationStepbyStep

Strony te pokazały mi jak utworzyć plik Config Properties i jak z niego korzystać.

- https://refactoring.guru/pl/design-patterns/behavioral-patterns
- https://refactoring.guru/pl/design-patterns/structural-patterns

Strony te przedstawiają zarys wzorców behawioralnych i strukturalnych.

Score Card:

- ✓ Technologia Java SE
- ✓ Aplikacja GUI

Cała aplikacja posiada GUI i została napisana z wykorzystaniem JavaFX i Scene Buildera

- ✓ Wielowątkowa

Wielowątkowość została przeze mnie użyta w momencie wyszukiwania produktów z API. Użytkownik wpisuje słowo które go interesuje po czym aplikacja w tle w czasie rzeczywistym szuka, zwraca listę odpowiadających produktów i aktualizuje interfejs graficzny. Dokładny moment użycia wielowątkowości:

```
public ObservableList<String> searchProductsMulti(String GivenInput) {
    ObservableList<String> productNames =
FXCollections.observableArrayList();
    if (GivenInput == null || GivenInput.isEmpty()) {
        return productNames;
    }

    executorService.submit(() -> {
        List<Product> searchResults =
OpenFoodFactsAPISearch.SearchForProduct(GivenInput);
        List<String> names =
searchResults.stream().map(Product::getName).toList();
        Platform.runLater(() -> {
            productNames.addAll(names);
        });
    });

    return productNames;
}
```

- ✓ Programowanie asynchroniczne

Programowanie asynchroniczne zostało przeze mnie wykorzystane głównie w funkcjach związanych z listami zakupów tj. Dodawanie produktów do lity, pobieranie wszystkich list, pobieranie lsity po jej nazwie i usuwanie listy. Do tego korzystam z obiektów CompletableFuture, które w tle pobierają interesujące mnie dane i zwracają „w przyszłości”. W tym samym czasie aplikacja może kontynuować działanie i nic się nie blokuje. Dodatkowo można stwierdzić, że wykorzystywana jest wielowątkowość przez użycie metody CompletableFuture.supplyAsync która uruchamia inny wątek niż wątek główny aplikacji. Przykład użycia programowania asynchronicznego:

```
public CompletableFuture<List<ShoppingList>> getAllShoppingLists() {
    return CompletableFuture.supplyAsync(() -> {
        List<ShoppingList> shoppingLists = new ArrayList<>();
        try (MongoCursor<Document> cursor = collection.find().iterator()) {
            while (cursor.hasNext()) {
                Document document = cursor.next();
                String listName = document.getString(nazwaListy_key);
                List<Document> shoppingList =
document.getList(ShoppingList_key, Document.class);
                List<Product> productList = new ArrayList<>();
```

```
                for(Document item : shoppingList)
                {
                    String productName = item.getString(productName_key);
                    int quantity = item.getInteger(quantity_key);
                    double price = item.getDouble(price_key);
                    Product product = new Product(productName, quantity,
price);
                    productList.add(product);
                }
                ShoppingList shoppingListObj = new ShoppingList(listName,
productList);
                shoppingLists.add(shoppingListObj);
            }
        }
        return shoppingLists;
    });
}
```

- ✓ Zdarzenia (własne zdarzenia)

Aplikacja ta posiada wiele własnych zdarzeń od różnego rodzaju funkcji po wciśnięciu przycisków po wyświetlanie błędów na interfejsie graficznym poprzez złe wpisanie wartości.

- ✓ Wykorzystanie danych pobranych z dwóch zewnętrznych usług webowych

Aplikacja ta korzysta tylko z jednego API pobranego z zewnętrznej usługi webowej i jest to Open Food Facts API

- ✓ Bazy danych (JDBC, JPA)

W swojej aplikacji wykorzystałem bazę danych MongoDB i w niej przetrzymuje wszystkie listy utworzone przez użytkownika.

- ✓ Config Properties

W aplikacji tej został użyty plik Properties który przetrzymuje wszelkie stałe zmienne które są wykorzystywane w programie. Plik jest ten ładowany w zależności zmiany sceny.

- ✓ Własne 2 komponenty graficzne

Program posiada wiele komponentów graficznych – guziki, zdjęcia, ikony, ListView itd.

- ✓ Dodatkowo Punktowane wykorzystanie reflection API

W momencie przełączania scen w kodzie jest używana metoda getClass() do pobierania informacji o klasie do której należy metoda. Następnie wywoływana jest metoda getResource() która zwraca URL pliku FXML który jest sceną i ma być załadowany przez FXMLLoader. Dodatkowo klasa FXMLLoader wewnętrznie korzysta z Reflection API do wczytywania plików FXML i tworzenia obiektów kontrolerów które zostały zdefiniowane w pliku.

- ✓ Wykorzystanie bibliotek Apache Commons oraz magistrali zdarzeniowych

Biblioteka Apache Commons została przeze mnie wykorzystana w celu edycji zmiennej typu String:

```
userGivenProductName = StringUtils.replace(userGivenProductName, " ", "+");
```

Magistrala zdarzeniowa z kolei została przeze mnie wykorzystana poprzez EventBus który przekazuje zdarzenia miedzy różnymi komponentami. W tym projekcie EventBus zmienia interfejs graficzny w sytuacji usunięcia produktów z listy czy też usunięcia całej listy. Przykład użycia:

```
public CompletableFuture<Void> deleteProduct(String listName, String
productName, double price, int quantity) {
    Document filter = new Document()
            .append(nazwaListy_key, listName)
            .append("shopping_list.product_name", productName);
    Document update = new Document()
            .append("$pull", new Document(ShoppingList_key, new
Document(productName_key, productName)));
    return CompletableFuture.runAsync(() -> {
        collection.updateOne(filter, update);
        eventBus.post(new DeletedProductEvent(listName, productName, price,
quantity));
    });
}
```

- ✓ Zastosowanie wzorców programowania obiektowego i zdarzeniowego

Cała aplikacja została napisana w sposób obiektowy i zdarzeniowy.

- ✓ Zastosowanie wzorców projektowych z serii strukturalnych i behawioralnych

Zastosowany przeze mnie wzorzec projektowy z serii strukturalnych to Adapter. Wykorzystywany jest on podczas szukania produktów w API, a dokładniej ma on postać strumienia który konwertuje listę produktów na listę nazw produktów, która potem jest dodawana do ObservableList. Dodatkowo można stwierdzić że adapterem jest cała klasa do szukania produktów z API – klasa zwracająca produkty z API zwraca je w postaci List<Products>, a klasa która wywołuje metodę szukania tych produktów zmienia te liste na ObservableList<String>, która potem może być łatwo wykorzystana i odczytana przez inne klasy np. klasę interfejsu graficznego.

Zastosowany przeze mnie wzorzec projektowy z serii behawioralnych to Obserwator. Wykorzystywany jest do tego EventBus który pełni właśnie rolę takiego obserwatora i jeśli jakiś obiekt zmieni swój stan to inny obiekt będzie o tym wiedział i odpowiednio zareaguje.

- ✓ Wymaga się całkowitego autorstwa kodów źródłowych