**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Aplikacja Intelligent Housewife**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

Opis:

Aplikacja ma za zadanie pomóc użytkownikowi w kontrolowaniu stanu produktów spożywczych w gospodarstwie domowym pilnując tego, aby produkty były uzupełniane, zapobiegać marnowaniu żywności oraz prowadząc statystyki sygnalizujące użytkownikowi m.in. że niektóre produkty nie są potrzebne użytkownikowi, gdyż ich nie używa i marnują się. Pozwala także na podstawie braków żywności automatycznie tworzyć listę zakupów i przekazywać ją użytkownikowi. Aplikacja jest połączona z ofertami partnerów handlowych i wskazuje, które produkty można kupić w najlepszej cenie w danym momencie.

1. **Wymagania funkcjonalne**
   1. Rejestracja lodówki

Opis: użytkownik może zarejestrować się za pomocą facebooka, imienia i nazwiska lub konta google. Wprowadza dane uzupełniające w postaci modelu lodówki, który posiada. System posiada bazę modelów lodówek i ich parametry (najważniejszym parametrem jest kubatura, która pozwoli nam na obliczenie dostępnego miejsca i możliwości zakupienia nowych produktów).

* 1. Ustalenie stanu bazowego produktów

Opis: użytkownik może ustalić jaki jest stan bazowy lodówki, co rozumie się jako listę produktów, które zawsze powinny znajdować się w lodówce. Jeśli system wykryje brak, któregoś z produktów w lodówce, przekazuje go do listy zakupów. Stan bazowy można modyfikować, dodawać do niego produkty i usuwać.

* 1. Dodawanie produktu do lodówki

Opis: funkcja pozwala dodać produkt do lodówki. Produkt będzie zawierał informację o kategorii, rodzaju, nazwie, marce, terminie ważności.

* 1. Usuwanie produktu z lodówki

Opis:

* 1. Lista zakupów

Opis:

* 1. Dodawanie produktu do listy zakupów

Opis:

* 1. Usuwanie produktu z listy zakupów

Opis:

* 1. Statystyki

Opis:

* 1. Alarm

Opis:

2.0. Aktualne promocje u partnerów

Opis:

2.1. Lokalizacja użytkownika

Opis:

1. **Wymagania niefunkcjonalne**

2.1. Wymagania jakościowe

- Aplikacja mobilna,

- Aplikacja webowa,

2.2. Używalność

- Aplikacja powinna wyglądać tak samo na tej samej klasie urządzeń mobilnych,

- Aplikacja i serwis powinny być dostępne w języku polskim i angielskim,

2.3. Wygląd

Standardy interfejsowe:

- Strona ma wyglądać podobnie w przeglądarkach: FireFox, Opera, Chrome, IE,

- Aplikacja mobilna – android,

2.4. Dokumentacja

- opis do czego służy aplikacja i w jaki sposób jej używać,

- samouczek (step by step) zawarty w aplikacji oraz w serwisie YouTube,

- pomoc programu,

2.5. Niezawodność i bezpieczeństwo

- W razie awarii serwera możliwe jest zaprzestanie pracy serwisu,

- Dane powinny być zabezpieczone poprzez stosowanie cyklicznych backupów baz danych,

- W przypadku wystąpienia wyjątku użytkownik jest informowany o jego wystąpieniu oraz podana jest mu możliwość zgłoszenia wyjątku serwisowi i skontaktowanie się z twórcami systemu,

- Metadane użytkowników muszą być zabezpieczone przed dostępem dla osób trzecich, a ich hasła muszą być hashowane,

- Hasła użytkowników nie powinny być trzymane jako „plain text”,

- Hasło powinno zawierać przynajmniej jedną cyfrę,

- Możliwe jest logowanie do serwisu za pomocą facebooka oraz konta google,

- Update aplikacji będzie wykonywany jedynie w godzinach pomiędzy 23 a 6 rano,

- W przypadku wystąpienia awarii w działaniu serwisu, projektant informuje, że w przypadku jednej doby od jej wystąpienia postara się usunąć usterkę,

* 1. Wydajność

- Rejestracja i weryfikacja konta użytkownika od chwili podania przez niego danych nie powinna przekraczać 30 minut,

- Serwer powinien mieć możliwość obsługi minimum tysiąca użytkowników jednocześnie,

- Serwer powinien przesłać stronę w czasie maksymalnie 10 sekund,

* 1. Wspieralność

- Aplikacja została napisana w języku Java 8,

- Serwis i aplikacja będą utrzymywane przez projektantów,

- Komentarze w kodzie zostały napisane w języku angielskim,

- Nazwy zmiennych są użyteczne i ich nazewnictwo jasno określa co reprezentuje dana zmienna,

* 1. Wymagania instalacyjne

- Serwis jest instalowany raz z ewentualną możliwością przeniesienia,

- Użytkownik może ściągnąć aplikację ze sklepu play lub oficjalnej strony internetowej aplikacji,

- Aplikacja powinna instalować się szybko i bez problemów jakie mogłaby sprawić bardziej skomplikowana instalacja przeciętnemu użytkownikowi,

* 1. Wymagania prawne

- Zapytanie użytkownika o lokalizację lub o zgodę na jego geolokalizację na podstawie pozycji GPS urządzenia mobilnego celem przesyłania ofert handlowych partnerów, którzy prowadzą działalność handlową w okolicy użytkownika,

- Określenie licencji,

**PLAN DZIAŁAŃ**

1. **TWORZENIE KLAS PRODUKTÓW**

* Klasą nadrzędną wszystkich produktów zawartych w aplikacji jest klasa *Product.* Posiada pola, które będzie miał każdy produkt, który będzie dziedziczył po tej klasie, takie jak:

- String name – reprezentujące nazwę produktu,

- String producer – reprezentujące producenta produktu,

- String dateOfPurchase – reprezentujące datę zakupu produktu i umieszczenia go w lodówce,

- String dateOfExpiration – reprezentujące datę ważności produktu, na jej podstawie będą wywoływane alarmy o zbliżającym się terminie upływu ważności produktu,

(Daty powinny być wywoływane z metod generujących te daty zawartych w klasie *Date*)

***Trello*** – **Napisz klasę Product**

* Klasa *Dairy* reprezentująca nabiał. Oprócz pól dziedziczonych po klasie *Product* zawiera też w konstruktorze wywołanie z enuma typu nabiału. Enum zawarty jest w klasie *DairyType* , a klasa *Dairy* posiada pole:

- private DairyType type - reprezentujące typ nabiału, np. ser żółty, mleko, twaróg etc.

***Trello*** – **Napisz klasy Dairy i enum DairyType**

* Klasa *Bread* reprezentująca pieczywo. Oprócz pól dziedziczonych po klasie *Product* zawiera też w konstruktorze wywołanie z enuma typu pieczywa. Enum zawarty jest w klasie *BreadType* , a klasa *Bread* posiada pole:

- private BreadType type - reprezentujące typ pieczywa, np. chleb, bułka, bagietka etc.

Pieczywo jest specyficznym typem produktu, którego termin ważności musi być generowany. Dlatego klasa Date posiada metodę która do aktualnej daty dodaje 3 dni jako upływu terminu ważności.

***Trello*** – **Napisz klasy Bread i enum BreadType**

* Klasa *Meat* reprezentująca mięso. Oprócz pól dziedziczonych po klasie *Product* zawiera też w konstruktorze wywołanie z enuma typu mięsa. Enum zawarty jest w klasie *MeatType* , a klasa *Meat* posiada pola:

- private MeatType type - reprezentujące typ mięsa, np. wieprzowina, wołowina, drób, dziczyzna, etc.

- private boolean isFrozen – reprezentujące informację czy mięso jest zamrożone, jeśli jest zamrożone, należy do daty spożycia dodać jeden rok i umieścić produkt w zamrażarce,

- private boolean isSliced - reprezentujące informację czy mięso jest pofiletowane czy w całości,

***Trello*** – **Napisz klasy Meat i enum MeatType**

* Klasa *Fish* reprezentująca ryby. Oprócz pól dziedziczonych po klasie *Product* zawiera też w konstruktorze wywołanie z enuma typu ryby. Enum zawarty jest w klasie *FishType* , a klasa *Fish* posiada pola:

- private FishType type - reprezentujące typ ryby, w tym przypadku sposobu jej przyrządzenia lub zapakowania np. wędzona, filet, tusza, puszka etc.

- private boolean isFrozen – reprezentujące informację czy ryba jest zamrożona, jeśli jest zamrożona, jeśli zamrożona należy do daty spożycia dodać jeden rok i umieścić produkt w zamrażarce,

- private boolean isSliced - reprezentujące informację czy mięso jest pofiletowane czy w całości,

Pieczywo jest specyficznym typem produktu, którego termin ważności musi być generowany. Dlatego klasa Date posiada metodę która do aktualnej daty dodaje 3 dni jako upływu terminu ważności.

***Trello*** – **Napisz klasy Fish i enum FishType**

* Klasa *Meats* reprezentująca wędliny. Oprócz pól dziedziczonych po klasie *Product* zawiera też w konstruktorze wywołanie z enuma typu wędliny. Enum zawarty jest w klasie *MeatsType* , a klasa *Meats* posiada pola:

- private MeatsType type - reprezentujące typ wędliny, np. drobiowa, wieprzowa, kiełbasa, szynka, ogonówka etc.

- private boolean isFrozen – reprezentujące informację czy mięso jest zamrożone, jeśli jest zamrożone, należy do daty spożycia dodać jeden rok i umieścić produkt w zamrażarce,

- private boolean isSliced - reprezentujące informację czy wędlina jest pokrojona na plasterki czy w całości, jeśli jest pokrojona na plasterki, nie powinna zostać zamrożona,

***Trello*** – **Napisz klasy Meats i enum MeatsType**

* Klasa *FruitsVegetables* reprezentująca owoce i warzywa. Oprócz pól dziedziczonych po klasie *Product* posiada pola:

- private boolean isFruit – reprezentujące informację czy produkt jest warzywem czy owocem, jeśli true jest owocem, jeśli false jest warzywem,

- private boolean isInTheFridge - reprezentujące informację czy warzywo/owoc ma zostać umieszczone w lodówce czy w szafce, jeśli w lodówce to termin przydatności spożycia zmieni się o 1 miesiąc, a jeśli na zewnątrz, to produkty będą zdatne do spożycia jedynie 7 dni,

***Trello*** – **Napisz klasę FruitsVegetables**

Sweets(Slodycze):

Spices(Przyprawy)

Others(Pozostałe) – produkty, które nie muszą być umieszczone w lodówce i ich termin ważności jest zawsze taki jak na paczce, nie można na niego w żaden sposób wpłynąć. Np. makaron, ryż etc.

1. **TWORZENIE KLAS DODATKOWYCH**

* Klasą określającą daty jest klasa *Date*. Powinna zawierać w sobie metody dostosowane do różnych użyć:

*generateCurrentDate();*  - generuje aktualną datę na min. do wykorzystania przy wprowadzaniu produktów oraz sprawdzaniu aktualnej daty dla codziennego porównywania z terminami upływu ważności produktów w naszym magazynie,

*generateBreadDateOfExpiration();* - generuje datę w tym przypadku odpowiedzialną za datę ważności produktu, który nie ma takiej daty podanej przez producenta, czyli na przykład pieczywo = *generateCurrentDate() + 3 dni do przodu,*

*generateFrozenDateOfExpiration();* - generuje datę w tym przypadku odpowiedzialną za datę ważności produktu, który jest lub będzie zmrożony czyli na przykład ryby = *generateCurrentDate() + 1 rok do przodu,*

*generateFridgeVegetablesFruitsDateOfExpiration();* - generuje datę przydatności spożycia owoców i warzyw, które znajdują się w lodówce *generateCurrentDate() + 1 miesiąc do przodu,*

*generateOutsideVegetablesFruitsDateOfExpiration();* - generuje datę przydatności spożycia owoców i warzyw, które znajdują się na zewnątrz *generateCurrentDate() + 7 dni do przodu,*

Format generowanej daty: (DD/mm/YYYY , HH:mm:SS)

***Trello*** – **Napisz klasę Date**

* Klasa *UnitTests – wszystkie metody muszą zostać przetestowane w tej klasie*

***Trello*** – **Napisz klasę UnitTests**

1. **TWORZENIE KLAS MAGAZYNOWANIA PRODUKTÓW I STANU PODSTAWOWEGO**

KONTENERY: Lodówka, zamrażarka, szafka, chlebak

STAN PODSTAWOWY

1. **TWORZENIE KLAS LIST – ZAKUPÓW ORAZ PRODUKTÓW NAJCZĘŚCIEJ MARNOWANYCH**

LISTA ZAKUPÓW – stanowi produkty, które trafiają tu, gdyż skończyły się w stanie podstawowym

PRODUKTY WYRZUCANE:

STATYSTYKI WYRZUCANIA:

1. **PROJEKTOWANIE GUI**

wymagania,

model biznesowy,

wykres przypadków użycia