Specyfikacja programu Zarlock

Jacek Bzdak

February 1, 2011

Chapter 1

Ogólne zachowania

1.1 Baza danych

1.1.1 Robocza baza danych

Program ma roboczą bazę danych, zmiany są persystowane[1] po każdej operacji.

1.1.1.1 Położenie roboczej bazy danych

Robocza baza danych jest położona w: `$ZARLOCK_HOME/zarlock/prod/zarlock.db`, gdzie `$ZARLOCK_HOME` znaczy:

linux `$HOME`

windows Jest zmienna (albo to jest w rejestrze) która trafia w "Moje dokumenty"

Do decyzji:
Może lepiej przechowywać w katalogu programu?

1.1.1.2 Operacje na roboczej bazie danych

backup Zapisanie zawartości roboczej bazy danych do pliku

odczyt backupu Odczyt backupu i nadpisanie nim roboczej bazy danych

czyszczenie Kasowanie zawartości roboczej bazy danych.

1.1.1.3 Backup

Kopie zapasowe roboczej bazy danych przechowywane są w katalogu `$ZARLOCK_HOME/backup`. Każdy plik z kopią ma nazwę postaci: back{yyyy-mm-dd}_{nn}.db, gdzie:

[1] Czyli zapisane do pamięci trwałej — na dysk.

yyyy-mm-dd to data utworzenia backupu

nn numer backupu danego dnia

Kopie zapasowe są zapisywane w następujących sytuacjach:

- Przy zamykaniu programu.
- Przed nadpisaniem bazy danych

Chapter 2

Specyfikacja GUI

2.1 Produkt

Produkt to taki platoński ideał. Poszczególne partie należące do produktu są zasadniczo na siebie wymienne (nie ma znaczenia czy na śniadanei pójdzie jogurt trukawkowy czy waniliowy — w jadłospisie stoi jogurt) .

2.1.1 Pola:

nazwa unikalna nazwa

data ważności ilość dni jaką produkt jest ważny domyślnie

jednostka domyślna jednostka

2.1.2 Przykłady produktów:

- Chleb (data ważności 3 dni; jednostka sztuka)
- Dżem (data ważności nieistotne; jednostka słoik)

2.1.3 Jednostka

2.1.3.1 Widget

Combo box podpowiadający jednostki które już są w bazie danych (zarówbo produktów i partii). Patrz opis na 2.11.4.

2.1.3.2 Zasady parsowania

Co wpiszesz to jest

2.1.4 Data ważności

2.1.4.1 Widget

Text field

2.1.4.2 Zasady parsowania

Liczba ma być.

2.1.4.3 Semantyka

Domyślna ilość dni przez które partie danego produktu będą ważne. Jeśli jest puste (w db `NULL`) to partia ma puste pole na datę ważności.

2.2 Partia

Konkretna partia stuffu który trafił do magazynu.

2.2.1 Pola

specyfikator Coś co odróżnia partie od siebie. W Chleb baltonowski i chleb razowy — baltonowski i razowy to specyfikatory.

cena cena jednostkowa z VAT

ilość początkowa ile jednostek było na początku (pole typu float)

ilość bierzące ile jest w magazynie

booking date data przyjścia do magazynu

expiry date data przydatności do spożycia

opis opis — pole do wyszukiwania

numer faktury pole tekstowe — może być puste (czasem faktura nie dochodzi i spisy są z natury)

2.2.1.1 Specyfikator

Widget Combobox co się sam uzupełnia

Reguły parsowania Brak

2.2.1.2 Cena

Widget Pole tekstowe

Reguły parsowania

\d+([.,]?\d+) liczba

\d+([.,]?\d+) ([p\+](22)|(15)|(7)%?) Czylu 15 + 7 znaczy 15zł + 7%VAT

Jeśli jest ustawione że podajemy ceny za całą partię (a nie jednostkowe) cena wpisana jest dzielona przez ilość jednostek.

2.2.1.3 Ilość początkowa

Reguły parsowania Więcej niż zero!

Podczas edycji nie wolno zmienić ilości początkiowej tak żeby się okazało że ilość bierząca jest ujemna[1].

2.2.1.4 Ilość bierząca

Read only

2.2.1.5 Boking date

Patrz parsowanie dat 2.11.3.1

Parsowanie Może być wpisane wstecz i w przód. Musi być mniejsze niz expiry_date

2.2.1.6 Expiry date

Patrz parsowanie dat 2.11.3.1

Parsowanie Może być wpisane wstecz i w przód. Musi być większe niż booking_date

2.2.1.7 Jedostka

Samouzupełniający combobox

Zasady parsowania Wsio ugodno:

Sztuka, kilogram, bohenek

2.3 Wyprowadzenie

2.3.1 Pola

partia partia do której wyprowadzamy

[1] Ja bym to ograniczenie wrzucił w bazke i potem łapał w GUI

ilosc ilość – big_decimal

data data wyprowadzenia

create_date data utworzenia wyprowadzenia

tytułem Powód wyprowadzenia

danie danie do którego przypisano wyprowadzenie — opcjonalne(!)

powod_wyprowadzenia enum opisujący powód wyprowadzenia

2.3.2 Wydawanie

Można wydać produkt bez przypisania do dania. Trzeba gdzieś dać taką opcję :)

2.4 Powod wyprowadzenia

Encja opisująca powód wyprowadzenia. Nowe powody wyprowadzenia dodaje programista.
Pola:

powód wyprowadzenia opis

wliczanie do stawek boolean. Czy wliczamy dane wyprowadzenie do stawki żywnościowej.

Powody wyprowadzenia:

Minął termin mija termin ważności

Nieświeży zestarzał się

Na cele programowe

Na posilek

Powód 'na posiłek' dodawany jest automatycznie i nie da się ustawić z poziomu Gui.

2.5 Dzien

Dzień obozu. Musi być metoda dodania en masse całego obozu dni.
Pola:

id id dnia

date data

ilosc osob ilość osób na terenie

Dzień po utworzeniu ma standardowo cztery nieusuwalne posiłki: śniadanie, obiad, podwieczorek, kolację.

2.6 Ilość osób na terenie

Jak to rozwiążesz w bazie danych to Twoja zabawka. Może być zagnieżdżone w dniu, może być oddzielna encja.

Pola:

uczestnicy ilość uczestników

kadra ilość kadry

inni ilość innych

2.7 Posilek

Posilek.

nazwa Label posiłku

dodatkowy czy jest to posiłek dodatkowy

dzien łącze do dnia

Posiłki standardowe nie są dodatkowe, posiłki które dodaje użytkownik są dodatkowe.

2.8 Danie

Danie.

nazwa nazwa

2.9 Dane statyczne

Każdy obóz ma następujące dane statyczne (wrzuć w głupią tabelkę konfigową).

komendant Imie i nazwisko komendanta

kwatermistrz Imie i nazwisko kwatermistrza

Nazwa obozu Nazwa oboz

Stawka żywnościowa

budżet żywnościowy

2.10 Zachowania — posiłki

Dla mnie rozwiązanie z wersji Java jest niezadowalające. Więc liczę na inwencję.

Na pewno musisz spełnić następujące wymagania:

- Przy każdym dniu masz podaną stawkę dzienną
- Przy każdym posiłku masz podaną stawkę za posiłek
- Przy daniu też!
- W głównym oknie programu masz napis w stylu: Wydano XX z YYY zł (ZZZ% budżetu)

2.10.1 Wydawanie na dania

Są dwie metody wydawania na danie: uproszczona i pełna.

uproszczona Masz combo boza podajesz w nim dane w formacie 'nazwa_produktu — specyfikator [jednostka]' program to parsuje przeszukuje nazę danych i wyświetla pasujące wejścia w formacie: 'nazwa_produktu — specyfikator [jednostka] : ilość_dostępna'. Po wybraniu jakiegoś wejścia w CB aktwyuje się text field w którym wpisujesz ilość wydaną.

pełna Ze stanu magazynu wybierasz partię którą wydajesz i wpisujesz ilość

2.10.1.1 Opis metody uproszczonej

Algorytm powinien:

- Wybierać w bazie danych tylko partie które: są na stanie (jest ich więcej niż zero; dzień posiłku jest >= dzień dodania partii), są danego dnia ważne.
- Łączyć partie które mają takie same parametry. To znaczy: jeśli są w magazynie trzy partie 'chleb — baltonowski [bochenek]' któreh jest odpowiednio 2 3 i 5 bochenków to w ComboBoxie pojawia jedno wyprowadzenie które ma 10 sztuk.
- Jedno kliknięcie usera może powodować wydanie wielu partii. Przy czym najpierw wydawane są te z któtszą datą przydatności.

2.11 Ogólne

2.11.1 Arytmetyka

Zmiennoprzecinkowa obliczenia robimy z dokładnością do 2 miejsc po przecinku.

- W javie BigDecimal
- W pythonie decimal
- W C++/QT na pewno są do tego klasy
- Ważne żeby zaokrąglać wartości koło zera. Tj żeby nie było tak że po filtrowaniu na ilość wieszą od zera pojawiały się partie z ilością 0.0 bo w bazie danych jest tego 0.0001 kg z reszt w dzieleniu.

2.11.2 Ogólne zasady

Ogólna zasada działania widgetów jest taka. Po wyjściu z okienka widzisz wartość sparsowaną przez program. Jak do niego wrócisz pracujesz na tym co wprowdziłeś.

- Wpisujesz jakąś wartość np. +5
- po wyjściu program zamienia to na datęi wyświetla 21.08.2010
- po powrocie do okienka masz znów +5

2.11.3 Daty

2.11.3.1 Parsowanie dat

Daty parsowane są według takiej metody:

[+-]\d +- ileś dni od określonej daty[2]

dzi[śs] dziś

\d\d? data w bierzącym miesiącu[3]

\d\d[.:,-]\d\d([.:,-]\d\d)? Data w formacie DD-MM(-YY)? Różne separatory są ważne nie ma co się ograniczać.

słownie dziś jutro po jutrze...

[2]Normalnie jest to dziś, są wyjątki

[3]Może lepiej zmieńmy na najbliższa data z takim dniem miesiąca. 01 wpisane 27.07 znaczy 01.08 a nie 01.07

2.11.3.2 Zachowanie widgeta

Szczególnie ważne jest tutaj zachowanie z 2.11.2! Bo jak będzie ryfa z wprowadzaniem dat to będzie kompletna ryfa!

2.11.4 Samouzupełniające combo boxy

Działają troszkę jak smart seach w googlach.

Wpisujesz w pole tekstowe "C" a w popupie pojawiają się:

- Chleb
- Cukierki
- Cukinia
- Celnika nerki

Generalnie robimy to kwerendą `LIKE`.

2.11.4.1 Uwaga implementacyjna

Jest szansa że koleś będzie wpusywać szybciej niż będzie się odświerzać gui.

W Javie implementacja była z gruba taka:

- Wyszukania były robione w wątkach background <- to zostaje!
- Jeśli jest bierzące wyszukiwanie w toku nie odpalamy następnego a zapisujemy do jednoelementowej kolejki (potencjalnie coś tam nadpisując)
- Przy wyjściu z wyszukania sprawdzamy czy w kolejce coś jest — wtedy kasujemy kolejkę i odpalamy to wyszkiwanie od zera.

2.11.5 Ogólne przemyślenia

Jak najwięcej logiki wrzucaj w bazę danych (triggery//constrainty) — tylko je potem cholera łap!

2.11.6 Raporty

- Stan magazynu na dzień
- To co wyszło danego dnia
- Kartoteki magazynowe czyli: Bierzemy wszystkie partie produktu o tej samej cenie i wypluwamy po kolei ile czego na stan weszło/zeszło.

Chapter 3

Raporty

3.1 Zasadnicze rozwiązania

Wygenerowanie dokumentacji na dowolny dzień i na cały obóz to jedno kliknięcie, oddzielnie jednym kliknięciem generujesz też kartoteki.