МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ АВТОМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ КАФЕДРА
ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Расчетно-графическая работа

по дисциплине «Современные информационные технологии»
на тему «Одностраничное Web-приложение»

Студент Кузьмин Д.С.
Группа АВТ-318
Преподаватель Васюткина И.А.

Новосибирск 2015 г.

Содержание

1 Цель работы 2

2 Техническое задание 2

3 Проектирование 2

4 Описание пользовательского интерфейса 3

5 Структурное описание разработки 4

6 Вывод 25

7 Список литературы 26

8 Приложение А. Листинг программы 27

1. Цель работы

Разработать одностраничное веб-приложение с использованием веб-фрейворка Vaadin для отображения информации о погоде в различных городах, курсе валют и кол-ве посещений страницы приложения

2. Техническое задание

1. При создании приложения использоть веб-фрейворк Vaadin для создания интерфейса и взаимодействия между клиентом и сервером.
2. Реализовать получение и парсинг данных прогноза погоды на текущий и завтрашний день с сервиса ForecastIO
3. Реализовать получение и парсинг данных о курсах валют (доллар США, евро) с сайта ЦентроБанка России
4. Реализовать учет числа посещений страницы во время работы приложения (уникальные посетители, общее число посещений)
5. Добавить возможность обновлять данные из пунктов выше вручную по нажатию кнопки без перезагрузки страницы
6. Показывать IP-адрес пользователя, находящегося на странице
7. Показывать на странице время последнего обновления данных
8. Хранить информацию о посещениях из NoSQL БД Mongo

3. Проектирование

Была спрроектирована следующая структура классов:

Краткое описание реализованных абстракций:

- Классы типа «DataService» работают непосредственно с источниками данных (база данных, сторонние сервисы). Результатом их работы является класс с полезной информацией типа «Data»
- Классы типа «Data» содержат в себе информацию, необходимую и достаточную для ее отображения в графическом интерфейсе без дополнительных запросов к посторонним модулям.

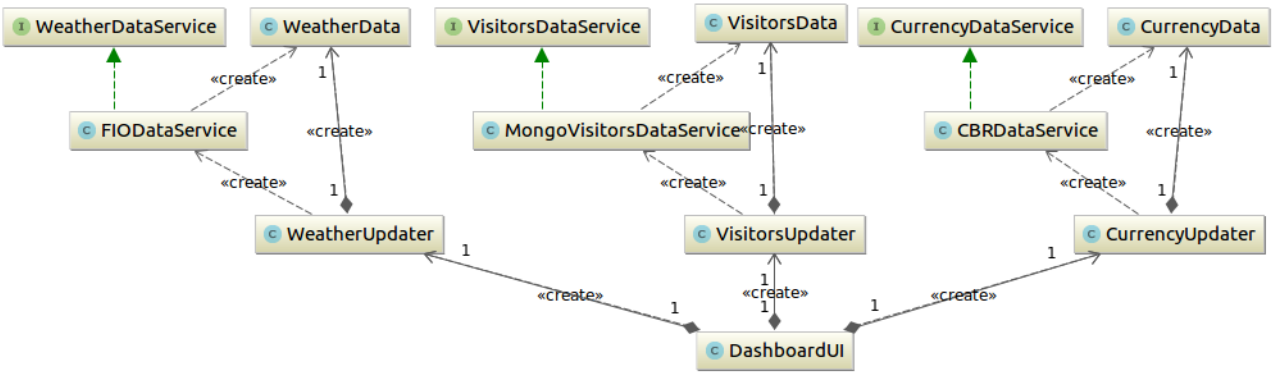

Рис. 1: Диаграмма классов проекта

- Классы типа «Updater» реализуют функционал обновления графических компонентов, предобрабатывая и передавая информацию из классов типа «Data» этим компонентам.
- Класс DashboardUI является корневым классом проекта. Он напрямую взаимодействует с фреймворком Vaadin - инициализирует интерфейс и является наследником его класса шаблона. Этот класс имеет внутренний класс DashboardUIServlet, который является наследником HttpServlet и привязывается к контейнеру сервлетов аннотацией фреймворка.

Привозникновении непредвиженных ситуаций, пользователю сообщается об ошибке в сплывающей панели.

4. Описание пользовательского интерфейса

Представление информации на стороне клиента показано на рисунке 2. На странице расположены 3 панели

- На левой панели отображается информация о погоде. В выпадающем списке можно выбрать населенный пункт, по которому будет загружаться информация. Доступно 3 населенных пункта - Москва, Санкт-Петербург и Новосибирск. Чуть ниже выводится температура в градусах по Цельсию на сегодняшний и завтрашний календарные дни. При смене пункта в выпадающем списке для применения изменений необходимо кликнуть по кнопке «Обновить».
- На средней панели отображается информация о текущей валюте - эквивалент Американского доллара и Евро в Российских рублях под колонкой «Курс» и изменение этого значения с момента последнего обновления в Центральном Банке России (обычно 1-3 дней) под колонкой «Изменение». Для обновления значений необходимо кликнуть по кнопке «Обновить» внизу панели.

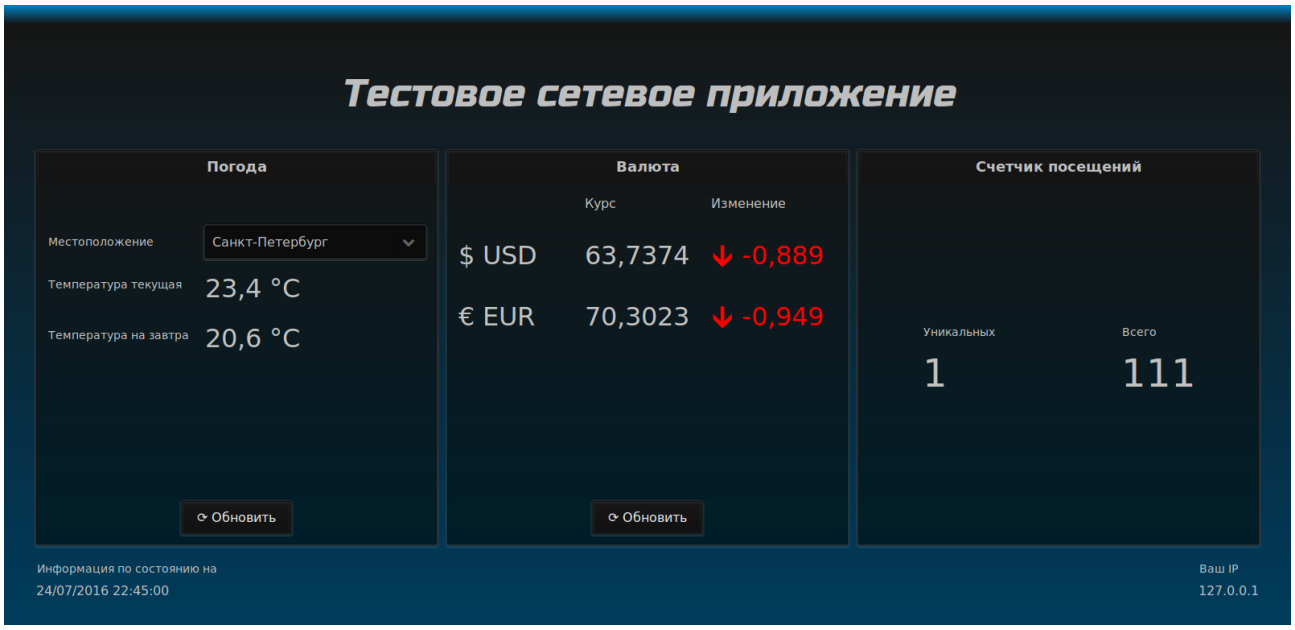

Рис. 2: Скриншот приложения

- На правой панели отображается счетчик посещений данной страницы. Под колонкой «Уникальных» отображено число уникальных посетителей, загрузивших данную страницу. Под колонкой «Всего» отображается общее количество посещений этой страницы.

В правом нижнем углу отображается текущий IP-адрес пользователя, загрузившего текущую страницу.

В левом нижнем углу отображается время последнего обновления любой из 2х панелей - о погоде либо о курсе валют. Либо с момента последеней перезагрузки страницы.

5. Структурное описание разработки

Пакет com.xotonic.dashboard.weather

Структура *Стр.*

Интерфейсы

WeatherDataService 5
Интерфейс для взаимодействия с `DashboardUI` (стр. 14)

Классы

Cities 5
Список городов
FIODataService 6

Загрузчик прогноза погоды с forecast.io.

WeatherData 8

Класс для передачи информации от загрузчиков погодных информеров (`WeatherDataService` (стр. 5))

Интерфейс WeatherDataService

Интерфейс для взаимодействия с `DashboardUI` (стр. 14)

Объявление

```
public interface WeatherDataService
```

Реализуют

FIODataService (стр. 6)

Методы

- **getData**

```
WeatherData getData(Cities city) throws com.xotonic.dashboard.
    ExceptionForUser
```

  - **Параметры**
    - `city` – Запрашиваемый город
  - **Возвращемое значение** – Информация по этому городу
  - **Выбрасывает исключение**
    - `com.xotonic.dashboard.ExceptionForUser` –

Класс Cities

Список городов

Объявление

```
public final Класс Cities
 extends java.lang.Enum
```

Поля

- `public static final Cities MSK`
  - Москва
- `public static final Cities SPB`
  - Санкт-Петебург
- `public static final Cities NSK`
  - Новосибирск

Конструкторы

- **Cities**

```
private Cities()
```

Методы

- **byID**

```
public static Cities byID(int id)
```

  - **Описание**

    Вернуть город по порядковому номеру в списке
  - **Параметры**
    - `id` – номер в списке
  - **Возвращемое значение** –
- **valueOf**

```
public static Cities valueOf(java.lang.String name)
```

- **values**

```
public static Cities[] values()
```

Класс FIODataService

Загрузчик прогноза погоды с forecast.io.
Следует помнить, что сервис предоставляет ограниченноое количество запросов (всего 1000 единовременно)

Объявление

```
public class FIODataService
 extends java.lang.Object implements WeatherDataService
```

Поля

- `private static final java.lang.String` **`APPKEY`**
  – Бесплатный APIKEY (ключ разработчика)
- `private static final java.lang.String[]` **`lats`**
  – Широты городов
- `private static final java.lang.String[]` **`lons`**
  – Долготы городов

Конструкторы

- **FIODataService**

  ```
  public FIODataService()
  ```

Методы

- **getData**

  ```
  public WeatherData getData(Cities city) throws com.xotonic.dashboard.
     ExceptionForUser
  ```

  – **Описание**
  Получить информацию о погоде в конкретном городе
  – **Параметры**
  * `city` – Запрашиваемый город

– **Возвращемое значение** –

– **Выбрасывает исключение**

* `com.xotonic.dashboard.ExceptionForUser` –

Класс WeatherData

Класс для передачи информации от загрузчиков погодных информеров (`WeatherDataService` (стр. 5))

Объявление

```
public class WeatherData
 extends java.lang.Object
```

Поля

- `public float` **`celcium_today`**
  – Значение температуры за сегодня в градусах по Цельсию
- `public float` **`celcium_tomorrow`**
  – Значение температуры на завтра в градусах по Цельсию
- `public Cities` **`city`**
  – Город, для которого предназначен прогноз

Конструкторы

- **WeatherData**

```
public WeatherData()
```

Пакет com.xotonic.dashboard.currency

Структура *Стр.*

Интерфейсы

CurrencyDataService 9
Интерфейс для службы получения курса валют

Классы

CBRDataService 9
Парсинг курса валют с сайта Центрального Банка России
CurrencyData 11

Интерфейс CurrencyDataService

Интерфейс для службы получения курса валют

Объявление

```
public interface CurrencyDataService
```

Реализуют

CBRDataService (стр. 9)

Методы

- **getData**

```
CurrencyData getData() throws com.xotonic.dashboard.ExceptionForUser
```

Класс CBRDataService

Парсинг курса валют с сайта Центрального Банка России

Объявление

```
public class CBRDataService
 extends java.lang.Object implements CurrencyDataService
```

Поля

- `private final java.lang.String` **`ID_USD`**
  – Уникальный код Американского доллара в XML
- `private final java.lang.String` **`ID_EUR`**
  – Уникальный код Евро в XML
- `private final java.lang.String` **`URL`**
  – URL запроса к Центробанку

Конструкторы

- **CBRDataService**

```
public CBRDataService()
```

Методы

- **buildUrl**

```
private java.lang.String buildUrl(java.util.Date from,java.util.Date to,java
   .lang.String id)
```

  - **Описание**

    Вставить в URL параметры запроса

  - **Параметры**

    * `from` – Начальная дата
    * `to` – Конечная дата
    * `id` – Идентификатор валюты

  - **Возвращемое значение** – Готовый URL для отправки на сервер ЦБ

- **getData**

```
public CurrencyData getData() throws com.xotonic.dashboard.ExceptionForUser
```

  - **Описание**

    Выполнить запрос на сайт Центробанка по обеим валютам

  - **Возвращемое значение** – Информация о валюте

  - **Выбрасывает исключение**

    * `com.xotonic.dashboard.ExceptionForUser` –

- **setLastWorkingDay**

```
private void setLastWorkingDay(java.util.Calendar cal)
```

  - **Описание**

    Найти последний рабочий день в ЦБ

– **Параметры**

* `cal` –

Класс CurrencyData

Класс, содержащий информацию о курсе валют

Объявление

```
public class CurrencyData
 extends java.lang.Object
```

Поля

- `public float` **`EUR`**
  – Курс евро по отношению к рублю
- `public float` **`EURDelta`**
  – Изменение евро
- `public float` **`USD`**
  – Курс доллара по отношению к рублю
- `public float` **`USDDelta`**
  – Изменение доллара

Конструкторы

- **CurrencyData**

```
public CurrencyData()
```

Пакет com.xotonic.dashboard.ui

Структура *Стр.*

Классы

CurrencyUpdater 12
Обновление курса валют

DashboardUI 14

DashboardUI.DashboardUIServlet 15
Сервлет приложения
VisitorsUpdater 16
Обновление счетчика посещения

WeatherUpdater 17

Класс CurrencyUpdater

Обновление курса валют

Объявление

```
public class CurrencyUpdater
 extends java.lang.Object implements com.vaadin.ui.Button.ClickListener, java.
    lang.Runnable
```

Поля

- `private final com.vaadin.ui.Label` **`usdLabel`**
  – Надпись для значения текущего курса доллара
- `private final com.vaadin.ui.Label` **`usdDeltaLabel`**
  – Надпись для значения изменения текущего курса доллара
- `private final com.vaadin.ui.Label` **`eurLabel`**
  – Надпись для значения текущего курса евро
- `private final com.vaadin.ui.Label` **`eurDeltaLabel`**
  – Надпись для значения изменения текущего курса евро
- `private boolean` **`silent`**
  – Метка, отключающая вывод сообщения пользователю об успешном обновлении
- `private final com.vaadin.ui.Label` **`timeStatusValueLabel`**
  – Надпись с временем последнего обновления
- `private final java.text.DecimalFormat` **`currencyFormat`**
  – Форматирование для строк с числовыми значениями курса

- `private final java.text.DecimalFormat` **`currencyDeltaFormat`**
  - Форматирование для строк с числовыми значениями изменения курса
- `private com.xotonic.dashboard.currency.CurrencyData` **`currencyData`**
  - Информация о текущем курсе

Конструкторы

- **CurrencyUpdater**

```
public CurrencyUpdater(com.vaadin.ui.Label usdLabel,com.vaadin.ui.Label
    usdDeltaLabel,com.vaadin.ui.Label eurLabel,com.vaadin.ui.Label
    eurDeltaLabel,com.vaadin.ui.Label timeStatusValueLabel)
```

  - **Описание**

    Конструктор
  - **Параметры**
    - `usdLabel` – курс доллара
    - `usdDeltaLabel` – изменение курса доллара
    - `eurLabel` – курс евро
    - `eurDeltaLabel` – изменение курса евро
    - `timeStatusValueLabel` – дата и время, которое обновится после получения новых данных

Методы

- **buttonClick**

```
public void buttonClick(com.vaadin.ui.Button.ClickEvent event)
```

  - **Описание**

    Обработчик клика по кнопке ’Обновить’
  - **Параметры**
    - `event` –
- **isSilent**

```
public boolean isSilent()
```

– **Описание**

Вернуть флаг, включающий оповещение пользователя об успешном обновлении

- **run**

```
public void run()
```

– **Описание**

Эмуляция нажатия на кнопку обновления при запуске класса как Runnable

- **setSilent**

```
public void setSilent(boolean silent)
```

– **Описание**

Поставить флаг, включающий оповещение пользователя об успешном обновлении

- **updateCurrency**

```
private void updateCurrency()
```

– **Описание**

Обновить информацию о курсе валют

- **updateDateLabel**

```
private void updateDateLabel()
```

– **Описание**

Обновить текст с временем последнего обновления

Класс DashboardUI

Отрисовка UI и управление событиями

Объявление

```
public class DashboardUI
 extends com.vaadin.ui.UI
```

Поля

- `WeatherUpdater` **`weatherUpdater`**
  - Компонент для обновления погоды
- `CurrencyUpdater` **`currencyUpdater`**
  - Компонент для обновления курса валют
- `VisitorsUpdater` **`visitorsUpdater`**
  - Компонент для обновления количества посещений

Конструкторы

- **DashboardUI**

```
public DashboardUI()
```

Методы

- **init**

```
protected void init(com.vaadin.server.VaadinRequest vaadinRequest)
```

  - **Описание**

    Инициализация приложения
  - **Параметры**
    - `vaadinRequest` – обьект класса-обертки HttpRequest

Класс DashboardUI.DashboardUIServlet

Сервлет приложения

Объявление

```
public static Класс DashboardUI.DashboardUIServlet
 extends com.vaadin.server.VaadinServlet
```

Конструкторы

- **DashboardUIServlet**

  ```
  public DashboardUIServlet()
  ```

Класс VisitorsUpdater

Обновление счетчика посещения

Объявление

```
public class VisitorsUpdater
 extends java.lang.Object implements com.vaadin.ui.Button.ClickListener, java.
    lang.Runnable
```

Поля

- `private final com.vaadin.ui.Label` **`uniqueLabel`**
  - Метка для записи числа уникальных IP
- `private final com.vaadin.ui.Label` **`totalLabel`**
  - Метка для записи общего числа посещений
- `private com.xotonic.dashboard.visitors.VisitorsData` **`visitorsData`**
  - Информация о посещениях

Конструкторы

- **VisitorsUpdater**

  ```
  public VisitorsUpdater(com.vaadin.ui.Label uniqueLabel,com.vaadin.ui.Label
     totalLabel)
  ```

  - **Описание**

    Конструктор класса

Методы

- **buttonClick**

```
public void buttonClick(com.vaadin.ui.Button.ClickEvent event)
```

  - **Описание**

    Обработчик нажатия на кнопку обновить. На данный момент копка отсутсвует, но обработчик используется при перезагрузке страницы

- **run**

```
public void run()
```

  - **Описание**

    Обновление информации при запуске класса как Runnable

- **updateVisitors**

```
private void updateVisitors()
```

  - **Описание**

    Загрузить данные из базы данных

- **upsertAddress**

```
private void upsertAddress(java.lang.String ip)
```

  - **Описание**

    Сохранить IP-адрес в базе данных

  - **Параметры**

    * `ip` – IP-адрес

Класс WeatherUpdater

Обновление прогноза погоды

Объявление

```
public class WeatherUpdater
 extends java.lang.Object implements com.vaadin.ui.Button.ClickListener, java.
    lang.Runnable
```

Поля

- `private com.xotonic.dashboard.weather.WeatherData` **`weatherData`**
  – Информация о погоде в определенном населенном пункте
- `public final int` **`defaultCityId`**
  – НП, который будет загружаться по умолчанию
- `private final com.vaadin.ui.ComboBox` **`placeSelect`**
  – Выпадающий список с НП
- `private final java.util.ArrayList` **`places`**
  – Лист наименований НП
- `private final com.vaadin.ui.Label` **`currentTemperature`**
  – Надпись с текущей температурой
- `private final com.vaadin.ui.Label` **`tomorrowTemperature`**
  – Надпись с температурой на завтра
- `private final com.vaadin.ui.Label` **`timeStatusValueLabel`**
  – Надпись с временем последнего обновления
- `private boolean` **`silent`**
  – Метка, отключающая вывод сообщения пользователю об успешном обновлении
- `private final java.text.DecimalFormat` **`weatherFormat`**
  – Формат числа для отображения градусов по Цельсию

Конструкторы

- **WeatherUpdater**

```
public WeatherUpdater(com.vaadin.ui.ComboBox placeSelect,java.util.ArrayList
    places,com.vaadin.ui.Label currentTemperature,com.vaadin.ui.Label
   tomorrowTemperature,com.vaadin.ui.Label timeStatusValueLabel)
```

- **Параметры**
  - `placeSelect` – компонент со списком городов (Порядок городов должен соответсвовать порядку в `Cities` (стр. 5))
  - `places` – массив со списком городов в том же порядке, что и у placeSelect
  - `currentTemperature` – текущая температура
  - `tomorrowTemperature` – температура на завтра
  - `timeStatusValueLabel` – время и дата, которое будет обновляться после обновления погоды

Методы

- **buttonClick**

```
public void buttonClick(com.vaadin.ui.Button.ClickEvent event)
```

  - **Описание**

    Обработчик события по клику на кнопку Обновить
  - **Параметры**
    - `event` –

- **isSilent**

```
public boolean isSilent()
```

  - **Описание**

    Вернуть флаг, включающий оповещение пользователя об успешном обновлении

- **run**

```
public void run()
```

  - **Описание**

    Эмуляция нажатия на кнопку обновления при запуске класса как Runnable

- **setSilent**

```
public void setSilent(boolean silent)
```

  - **Описание**

    Поставить флаг, включающий оповещение пользователя об успешном обновлении

- **updateDateLabel**

```
private void updateDateLabel()
```

  - **Описание**

    Обновить текст с временем последнего обновления

- **updateWeather**

```
private void updateWeather(int id)
```

  - **Описание**

    Обновить информацию о погоде

  - **Параметры**

    * `id` – номер населеного пункта

Пакет com.xotonic.dashboard

Структура *Стр.*

Exception ExceptionForUser

Исключение, которое следует показать клиенту

Объявление

```
public class ExceptionForUser
 extends java.lang.Exception
```

Поля

- `private java.lang.String` **`what`**
  - Строковое описание ошибки

Конструкторы

- **ExceptionForUser**

```
public ExceptionForUser(java.lang.String what)
```

  - **Параметры**
    * `what` – строковое описание ошибки

Методы

- **what**

```
public java.lang.String what()
```

  - **Описание**

    Выдать строквое описание ошибки
  - **Возвращемое значение** –

Пакет com.xotonic.dashboard.visitors

Структура *Стр.*

Интерфейсы

VisitorsDataService 21
Интерфейс для службы по учету IP-адресов посетителей

Классы

MongoVisitorsDataService 22
Загрузчик/регистратор числа посещений из БД MongoDB
Порт, адрес, и БД сервера стандартные
Собсно, требуется рабочий mongod
Используется 3я версия драйвера
Формат документа
`{ ip : string, count : integer}`
VisitorsData 24

Интерфейс VisitorsDataService

Интерфейс для службы по учету IP-адресов посетителей

Объявление

```
public interface VisitorsDataService
```

Реализуют

MongoVisitorsDataService (стр. 22)

Методы

- **getData**

```
VisitorsData getData() throws com.xotonic.dashboard.ExceptionForUser
```

- **registerIP**

```
void registerIP(java.lang.String ip) throws com.xotonic.dashboard.
    ExceptionForUser
```

Класс MongoVisitorsDataService

Загрузчик/регистратор числа посещений из БД MongoDB
Порт, адрес, и БД сервера стандартные
Собсно, требуется рабочий mongod
Используется 3я версия драйвера
Формат документа

```
{ ip : string, count : integer}
```

Объявление

```
public class MongoVisitorsDataService
 extends java.lang.Object implements VisitorsDataService
```

Поля

- `java.lang.String` **`MongoDBServerAddress`**
  – Адрес сервера БД
- `java.lang.String` **`dbName`**
  – Имя БД

- `int` **`MongoDBServerPort`**
  - Порт сервера БД

Конструкторы

- **MongoVisitorsDataService**

```
public MongoVisitorsDataService()
```

Методы

- **connect**

```
private com.mongodb.client.MongoCollection connect() throws com.xotonic.
    dashboard.ExceptionForUser
```

  - **Описание**

    Устрановить соединение с сервером БД
  - **Возвращемое значение** –
  - **Выбрасывает исключение**
    - `com.xotonic.dashboard.ExceptionForUser` –

- **getData**

```
public VisitorsData getData() throws com.xotonic.dashboard.ExceptionForUser
```

  - **Описание**

    Получить информацию о посещениях из БД
  - **Возвращемое значение** –
  - **Выбрасывает исключение**
    - `com.xotonic.dashboard.ExceptionForUser` –

- **registerIP**

```
public void registerIP(java.lang.String ip) throws com.xotonic.dashboard.
    ExceptionForUser
```

– **Описание**

  Записать адрес в БД

– **Параметры**

  * `ip` – IP-адрес

– **Выбрасывает исключение**

  * `com.xotonic.dashboard.ExceptionForUser` –

Класс VisitorsData

Информация по статистике посещений

Объявление

```
public class VisitorsData
 extends java.lang.Object
```

Поля

- `public int` **`unique`**
  – Число уникальных посетителей
- `public int` **`total`**
  – Общее число посещений

Конструкторы

- **VisitorsData**

```
public VisitorsData()
```

6. Вывод

Было реализовано одностраничное приложение для отображения информации по погоде, курсе валют и количестве посещений. Как основа использовался веб-фреймворк Vaadin. Данная технология полностью реализует процесс разработки клиентской стороны приложения и процесс обмена информацией между браузером клиента и сервером Tomcat8, что существенно ускоряет процесс разработки.

7. Список литературы

1. Book of Vaadin: [Электронный ресурс]. URL: https://vaadin.com/book. (Дата обращения: 28.12.2016).
2. The MongoDB 3.2 Manual: [Электронный ресурс]. URL: https://docs.mongodb.com /v3.2/. (Дата обращения: 25.12.2016).
3. Java™ Platform Standard Ed. 8: [Электронный ресурс]. URL: https://docs.oracle.com /javase/8/docs/api/. (Дата обращения: 27.12.2016).
4. Java Platform Standard Edition 8 Documentation: [Электронный ресурс]. URL: https://docs.oracle.com/javase/8/docs/. (Дата обращения: 23.12.2016).

8. Приложение А. Листинг программы

ExceptionForUser.java

```
package com.xotonic.dashboard;

/**
 * Исключение, которое следует показать клиенту
 * @author xotonic
 */
public class ExceptionForUser extends Exception {
    /**
     * Строковое описание ошибки
     */
    private String what;

    /**
     * Выдать строквое описание ошибки
     * @return
     */
    public String what() {
        return what;
    }

    /**
     *
     * @param what строковое описание ошибки
     */
    public ExceptionForUser(String what)
    {
        this.what = what;
    }
}
```

CBRDataService.java

```
package com.xotonic.dashboard.currency;

import com.xotonic.dashboard.ExceptionForUser;
import java.io.IOException;
import java.net.URL;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

/**
 * Парсинг курса валют с сайта Центрального Банка России
 *
 * @author xotonic
```

```
 */
public class CBRDataService implements CurrencyDataService {

    /**
     * Уникальный код Американского доллара в XML
     */
    private final String ID_USD = "R01235";
    /**
     * Уникальный код Евро в XML
     */
    private final String ID_EUR = "R01239";

    /**
     * URL запроса к Центробанку
     */
    private final String URL = "http://www.cbr.ru/scripts/XML_dynamic.asp?
   date_req1=%s&date_req2=%s&VAL_NM_RQ=%s";

    /**
     * Вставить в URL параметры запроса
     * @param from Начальная дата
     * @param to Конечная дата
     * @param id Идентификатор валюты
     * @return Готовый URL для отправки на сервер ЦБ
     */
    private String buildUrl(Date from, Date to, String id) {
        SimpleDateFormat format = new SimpleDateFormat("dd/MM/yyyy");
        return String.format(URL, format.format(from), format.format(to), id);
    }

    /**
     * Найти последний рабочий день в ЦБ
     * @param cal
     */
    private void setLastWorkingDay(Calendar cal) {
        if (cal.get(Calendar.DAY_OF_WEEK) == Calendar.SUNDAY) {
            cal.add(Calendar.DAY_OF_YEAR, -1);
        } else if (cal.get(Calendar.DAY_OF_WEEK) == Calendar.MONDAY) {
            cal.add(Calendar.DAY_OF_YEAR, -2);
        }
    }

    /**
     * Выполнить запрос на сайт Центробанка по обеим валютам
     * @return Информация о валюте
     * @throws ExceptionForUser
     */
    @Override
    public CurrencyData getData() throws ExceptionForUser {
        CurrencyData cd = new CurrencyData();
        cd.USD = Float.NaN;
        cd.EUR = Float.NaN;
        cd.EURDelta = Float.NaN;
        cd.USDDelta = Float.NaN;


        /*
            У ЦБ в понедельник и воскресение выходные.
            Те.. в эти дни нет записей курса в xml
            Вместо них берем субботу
            Ахтунг! Не учитываются праздники!
```

```
         */
        Calendar cal = Calendar.getInstance();

        cal.set(Calendar.HOUR_OF_DAY, 0);
        cal.set(Calendar.MINUTE, 0);
        cal.set(Calendar.SECOND, 0);
        setLastWorkingDay(cal);
        Date to = cal.getTime();

        cal.add(Calendar.DAY_OF_YEAR, -1);
        setLastWorkingDay(cal);
        Date from = cal.getTime();

        /*
            USD
        */

        String urlUSD = buildUrl(from, to, ID_USD);
        System.out.println(urlUSD);

        try {
            DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
            dbf.setNamespaceAware(true);
            DocumentBuilder db = dbf.newDocumentBuilder();
            URL url = new URL(urlUSD);
            Document doc = db.parse(url.openStream());
            /*
            Вывод всего XML

            DOMSource domSource = new DOMSource(doc);
            StringWriter writer = new StringWriter();
            StreamResult result = new StreamResult(writer);
            TransformerFactory tf = TransformerFactory.newInstance();
            Transformer transformer = tf.newTransformer();
            transformer.transform(domSource, result);
            System.out.println("XML IN String format is: \n" + writer.toString()
);
             */
            NodeList records = doc.getElementsByTagName("Value");

            String value = records.item(0).getTextContent();
            String lastValue = records.item(1).getTextContent();
            float cur = Float.parseFloat(value.replace(',', '.'));
            float curLast = Float.parseFloat(lastValue.replace(',', '.'));

            cd.USD = cur;
            cd.USDDelta = cur - curLast;

        } catch (ParserConfigurationException | SAXException | IOException ex) {
            Logger.getLogger(CBRDataService.class.getName()).log(Level.SEVERE,
null, ex);
            /*

            } catch (TransformerConfigurationException ex) {
            Logger.getLogger(CBRDataService.class.getName()).log(Level.SEVERE,
null, ex);
        } catch (TransformerException ex) {
            Logger.getLogger(CBRDataService.class.getName()).log(Level.SEVERE,
null, ex);
        }
            */
```

```
            throw new ExceptionForUserОшибка(" запроса валюты " + ex.getMessage())
    ;

        }

        /*
            EUR
        */

        String urlEUR = buildUrl(from, to, ID_EUR);
        System.out.println(urlEUR);

        try {
            DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
            dbf.setNamespaceAware(true);
            DocumentBuilder db = dbf.newDocumentBuilder();
            URL url = new URL(urlEUR);
            Document doc = db.parse(url.openStream());

            NodeList records = doc.getElementsByTagName("Value");

            String value = records.item(0).getTextContent();
            String lastValue = records.item(1).getTextContent();
            float cur = Float.parseFloat(value.replace(',', '.'));
            float curLast = Float.parseFloat(lastValue.replace(',', '.'));

            cd.EUR = cur;
            cd.EURDelta = cur – curLast;

        } catch (ParserConfigurationException | SAXException | IOException ex) {
            Logger.getLogger(CBRDataService.class.getName()).log(Level.SEVERE,
    null, ex);
            throw new ExceptionForUserОшибка(" запроса валюты "+ex.getMessage());
        }



        return cd;
    }
}
```

CurrencyData.java

```
package com.xotonic.dashboard.currency;

/**
 * Класс, содержащий информацию о курсе валют
 * @author xotonic
 */
public class CurrencyData {
    /**
     * Курс евро по отношению к рублю
     */
    public float EUR;
    /**
     * Изменение евро
     */
    public float EURDelta;

    /**
```

```
     * Курс доллара по отношению к рублю
     */
    public float USD;
    /**
     * Изменение доллара
     */
    public float USDDelta;


}
```

CurrencyDataService.java

```
package com.xotonic.dashboard.currency;

import com.xotonic.dashboard.ExceptionForUser;

/**
 * Интерфейс для службы получения курса валют
 * @author xotonic
 */
public interface CurrencyDataService {
    public CurrencyData getData() throws ExceptionForUser ;
}
```

CurrencyUpdater.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package com.xotonic.dashboard.ui;

import com.vaadin.ui.*;
import com.xotonic.dashboard.ExceptionForUser;
import com.xotonic.dashboard.currency.*;
import java.math.RoundingMode;
import java.text.DecimalFormat;
import java.text.SimpleDateFormat;
import java.util.Calendar;

/**
 * Обновление курса валют<br>
 * @author xotonic
 */
public class CurrencyUpdater implements Button.ClickListener, Runnable {
    /**
     * Надпись для значения текущего курса доллара
     */
    private final Label usdLabel;
    /**
     * Надпись для значения изменения текущего курса доллара
     */
    private final Label usdDeltaLabel;
    /**
     * Надпись для значения текущего курса евро
     */
    private final Label eurLabel;
    /**
```

```
 * Надпись для значения изменения текущего курса евро
 */
private final Label eurDeltaLabel;
/**
 * Метка, отключающая вывод сообщения пользователю об успешном обновлении
 */
private boolean silent;
/**
 * Надпись с временем последнего обновления
 */
private final Label timeStatusValueLabel;

/** Форматирование для строк с числовыми значениями курса */
private final DecimalFormat currencyFormat;
/** Форматирование для строк с числовыми значениями изменения курса */
private final DecimalFormat currencyDeltaFormat;

/**
 * Информация о текущем курсе
 */
private CurrencyData currencyData = new CurrencyData();
/**  Конструктор
 * @param usdLabel курс доллара
 * @param usdDeltaLabel изменение курса доллара
 * @param eurLabel курс евро
 * @param eurDeltaLabel изменение курса евро
 * @param timeStatusValueLabel дата и время, которое обновится после получения
 * новых данных
 */
public CurrencyUpdater(
        Label usdLabel,
        Label usdDeltaLabel,
        Label eurLabel,
        Label eurDeltaLabel,
        Label timeStatusValueLabel)
        {
    this.usdLabel = usdLabel;
    this.usdDeltaLabel = usdDeltaLabel;
    this.eurLabel = eurLabel;
    this.eurDeltaLabel = eurDeltaLabel;
    this.timeStatusValueLabel = timeStatusValueLabel;

    this.silent = true;

    currencyFormat = new DecimalFormat("#.####");
    currencyFormat.setRoundingMode(RoundingMode.CEILING);
    currencyDeltaFormat = new DecimalFormat("+#.###;−#.###");
    currencyDeltaFormat.setRoundingMode(RoundingMode.CEILING);
}

/**
 * Обновить информацию о курсе валют
 */
private void updateCurrency() {
    CurrencyDataService loader = new CBRDataService();
    try {
        currencyData = loader.getData();
    } catch (ExceptionForUser e) {
        Notification.show(e.what(), Notification.Type.ERROR_MESSAGE);
    }
}
```

```
    /**
     * Обновить текст с временем последнего обновления
     */
    private void updateDateLabel() {
        if (timeStatusValueLabel != null) {
            timeStatusValueLabel.setValue(
                    new SimpleDateFormat("dd/MM/yyyy HH:mm:ss")
                    .format(Calendar.getInstance().getTime()));
        }
    }

    /**
     * Обработчик клика по кнопке Обновить''
     * @param event
     */
    @Override
    public void buttonClick(Button.ClickEvent event) {
        updateCurrency();
        usdLabel.setValue(currencyFormat.format(currencyData.USD));
        usdDeltaLabel.setValue(currencyDeltaFormat.format(currencyData.USDDelta)
    );
        usdDeltaLabel.setStyleName(currencyData.USDDelta < 0
                ? "currency-delta-negative" : "currency-delta-positive", true);

        eurLabel.setValue(currencyFormat.format(currencyData.EUR));
        eurDeltaLabel.setValue(currencyDeltaFormat.format(currencyData.EURDelta)
    );
        eurDeltaLabel.setStyleName(currencyData.EURDelta < 0
                ? "currency-delta-negative" : "currency-delta-positive", true);
        if (silent == false) {
            Notification.showВалюта(" обновлена", Notification.Type.
    HUMANIZED_MESSAGE);
        }
        updateDateLabel();
    }

    /**
     * Эмуляция нажатия на кнопку обновления при запуске класса как Runnable
     */
    @Override
    public void run() {
        buttonClick(null);
    }

    /**
     *  Вернуть флаг, включающий оповещение пользователя об успешном обновлении
     */
    public boolean isSilent() {
        return silent;
    }

    /**
     * Поставить флаг, включающий оповещение пользователя об успешном обновлении
     */
    public void setSilent(boolean silent) {
        this.silent = silent;
    }
}
```

DashboardUI.java

```
package com.xotonic.dashboard.ui;

import com.vaadin.annotations.Push;
import javax.servlet.annotation.WebServlet;

import com.vaadin.annotations.*;
import com.vaadin.server.*;
import com.vaadin.shared.ui.combobox.FilteringMode;
import com.vaadin.ui.*;
import java.util.ArrayList;

/**
 * Отрисовка UI и управление событиями
 */
@Theme("mytheme")
@Widgetset("com.xotonic.dashboard.DashboardAppWidgetset")
@Push // Аддон для управлением UI из другого потока
public class DashboardUI extends UI {

    /**
     * Компонент для обновления погоды
     */
    WeatherUpdater weatherUpdater;
    /**
     * Компонент для обновления курса валют
     */
    CurrencyUpdater currencyUpdater;
    /**
     * Компонент для обновления количества посещений
     */
    VisitorsUpdater visitorsUpdater;

    /**
     * Инициализация приложения
     * @param vaadinRequest обьект классаобертки– HttpRequest
     */
    @Override
    protected void init(VaadinRequest vaadinRequest) {
        VerticalLayout vlayout = new VerticalLayout();
        vlayout.addStyleName("outlined");
        vlayout.addStyleName("bg");
        vlayout.setSizeFull();
        vlayout.setMargin(true);
        HorizontalLayout hlayout = new HorizontalLayout();
        hlayout.addStyleName("outlined");
        hlayout.setSizeFull();
        setContent(vlayout);

        Label caption = new LabelТестовое(" сетевое приложение");
        caption.setStyleName("logo–label", true);
        caption.setWidth(null);
        vlayout.addComponent(caption);
        vlayout.setExpandRatio(caption, 0.2f);

        vlayout.setComponentAlignment(caption, Alignment.MIDDLE_CENTER);
        vlayout.addComponent(hlayout);
        vlayout.setExpandRatio(hlayout, 0.7f);

        /*
```

```
     WEATHER
     */
    final Panel weatherPanel = new Panel("<centerПогода></center>");
    weatherPanel.addStyleName("frame-bg-weather");
    weatherPanel.setSizeFull();

    final ArrayList<String> places = new ArrayList<>();
    places.addМосква("");
    places.addСанктПетербург("-");
    places.addНовосибирск("");

    final ComboBox placeSelect = new ComboBoxМестоположение("", places);

    placeSelect.setWidth(100.0f, Unit.PERCENTAGE);

    placeSelect.setFilteringMode(FilteringMode.CONTAINS);
    placeSelect.setImmediate(true);

    // Отключаем пустой выбор
    placeSelect.setNullSelectionAllowed(false);
    final Label currentTemperature = new Label("-");
    currentTemperature.setStyleName("celcium", true);
    currentTemperature.setCaptionТемпература(" текущая");

    final Label tomorrowTemperature = new Label("-");
    tomorrowTemperature.setCaptionТемпература(" на завтра");
    tomorrowTemperature.setStyleName("celcium", true);

    FormLayout weatherFormLayout = new FormLayout(placeSelect,
currentTemperature, tomorrowTemperature);
    Button updateWeatherButton = new Button("\u27F3 Обновить");

    VerticalLayout weatherMainLayout = new VerticalLayout(weatherFormLayout)
;
    weatherMainLayout.setSizeFull();
    weatherMainLayout.setMargin(true);
    weatherMainLayout.addComponent(updateWeatherButton);
    weatherMainLayout.setComponentAlignment(updateWeatherButton, Alignment.
BOTTOM_CENTER);
    weatherPanel.setContent(weatherMainLayout);
    /*
     CURRENCY
     */

    Panel currencyPanel = new Panel("<centerВалюта></center>");
    currencyPanel.addStyleName("frame-bg-currency");

    currencyPanel.setSizeFull();
    GridLayout currencyGridLayout = new GridLayout();
    currencyGridLayout.setSizeFull();
    currencyGridLayout.setRows(3);
    currencyGridLayout.setColumns(3);
    Label usd = new Label("$ USD");
    Label eur = new Label("\u20AC EUR");
    usd.setStyleName("currency", true);
    eur.setStyleName("currency", true);

    currencyGridLayout.addComponent(usd, 0, 1);
    currencyGridLayout.addComponent(eur, 0, 2);
    currencyGridLayout.addComponent(new LabelКурс(""), 1, 0);
    currencyGridLayout.addComponent(new LabelИзменение(""), 2, 0);
```

```
        final Label usdLabel = new Label("0");
        final Label usdDeltaLabel = new Label("0");
        final Label eurLabel = new Label("0");
        final Label eurDeltaLabel = new Label("0");
        usdLabel.setStyleName("currency", true);
        usdDeltaLabel.setStyleName("currency", true);
        eurLabel.setStyleName("currency", true);
        eurDeltaLabel.setStyleName("currency", true);

        currencyGridLayout.addComponent(usdLabel, 1, 1);
        currencyGridLayout.addComponent(eurLabel, 1, 2);
        currencyGridLayout.addComponent(usdDeltaLabel, 2, 1);
        currencyGridLayout.addComponent(eurDeltaLabel, 2, 2);

        VerticalLayout mainCurrencyLayout = new VerticalLayout(
currencyGridLayout);
        mainCurrencyLayout.setSizeFull();
        mainCurrencyLayout.setMargin(true);
        mainCurrencyLayout.setSpacing(true);
        Button updateCurrencyButton = new Button("\u27F3 Обновить");
        mainCurrencyLayout.addComponent(updateCurrencyButton);
        mainCurrencyLayout.setComponentAlignment(updateCurrencyButton, Alignment
.BOTTOM_CENTER);
        currencyPanel.setContent(mainCurrencyLayout);


        /*
         VISITORS
         */
        Panel visitorsPanel = new Panel("<centerСчетчик> посещений</center>");
        visitorsPanel.addStyleName("frame-bg-visitors");
        Label ipUniqueLabel = new Label("0");
        ipUniqueLabel.setCaptionУникальных("");
        ipUniqueLabel.setSizeUndefined();
        ipUniqueLabel.setStyleName("visitors-counter-label", true);

        Label ipTotalLabel = new Label("0");
        ipTotalLabel.setCaptionВсего("");
        ipTotalLabel.setSizeUndefined();
        ipTotalLabel.setStyleName("visitors-counter-label", true);

        visitorsPanel.setSizeFull();
        HorizontalLayout visitorsMainLayout = new HorizontalLayout(ipUniqueLabel
, ipTotalLabel);
        visitorsMainLayout.setSizeFull();
        visitorsMainLayout.setComponentAlignment(ipUniqueLabel, Alignment.
MIDDLE_CENTER);
        visitorsMainLayout.setComponentAlignment(ipTotalLabel, Alignment.
MIDDLE_CENTER);
        visitorsPanel.setContent(visitorsMainLayout);

        hlayout.setSpacing(true);
        hlayout.addComponent(weatherPanel);
        hlayout.addComponent(currencyPanel);
        hlayout.addComponent(visitorsPanel);

        /*
         FOOTER
         */
```

```
        /*
          Дата состояния обновляется после запуска любого из листенеров
         (Visitors, Currency, Weather)
        */

        Label timeStatusValueLabel = new Label("---");
        timeStatusValueLabel.setCaptionИнформация(" по состоянию на");

        // Получаем IP и выводим
        final WebBrowser webBrowser = Page.getCurrent().getWebBrowser();


        Label ipValueLabel = new Label(webBrowser.getAddress());
        ipValueLabel.setCaptionВаш(" IP");

        HorizontalLayout infoHLayout = new HorizontalLayout();
        infoHLayout.addStyleName("outlined");
        infoHLayout.setSizeFull();

        infoHLayout.addComponent(timeStatusValueLabel);
        timeStatusValueLabel.setWidth(null);
        infoHLayout.setComponentAlignment(timeStatusValueLabel, Alignment.
BOTTOM_LEFT);

        infoHLayout.addComponent(ipValueLabel);
        ipValueLabel.setWidth(null);
        infoHLayout.setComponentAlignment(ipValueLabel, Alignment.BOTTOM_RIGHT);

        vlayout.addComponent(infoHLayout);
        vlayout.setExpandRatio(infoHLayout, 0.1f);

        /*
            INITIALIZING UPDATERS
        */

        weatherUpdater = new WeatherUpdater(placeSelect, places,
currentTemperature, tomorrowTemperature, timeStatusValueLabel);
        currencyUpdater = new CurrencyUpdater(usdLabel, usdDeltaLabel, eurLabel,
 eurDeltaLabel, timeStatusValueLabel);
        visitorsUpdater = new VisitorsUpdater(ipUniqueLabel, ipTotalLabel);

        updateWeatherButton.addClickListener(weatherUpdater);
        updateCurrencyButton.addClickListener(currencyUpdater);

        // Обновление данных происходит в отдельных потоках, чтобы у клиента не
        // задерживалась отрисовка интерфейса
        // Для обновления UI из отдельного потока понадобился thirdparty addon
        // vaadin-push
        this.access(weatherUpdater);
        this.access(currencyUpdater);
        this.access(visitorsUpdater);
    }

    /**
     * Сервлет приложения
     */
    @WebServlet(urlPatterns = "/*", name = "DashboardUIServlet", asyncSupported
= true)
    @VaadinServletConfiguration(ui = DashboardUI.class, productionMode = false)
    public static class DashboardUIServlet extends VaadinServlet {
    }
```

```
}
```

VisitorsUpdater.java

```
package com.xotonic.dashboard.ui;

import com.vaadin.server.Page;
import com.vaadin.server.WebBrowser;
import com.vaadin.ui.Button;
import com.vaadin.ui.Label;
import com.vaadin.ui.Notification;
import com.xotonic.dashboard.ExceptionForUser;
import com.xotonic.dashboard.visitors.MongoVisitorsDataService;
import com.xotonic.dashboard.visitors.VisitorsData;
import com.xotonic.dashboard.visitors.VisitorsDataService;

/**
 * Обновление счетчика посещения<br>

 * @author xotonic
 */
public class VisitorsUpdater implements Button.ClickListener, Runnable {

    /**
     * Метка для записи числа уникальных IP
     */
    private final Label uniqueLabel;
    /**
     * Метка для записи общего числа посещений
     */
    private final Label totalLabel;
    /**
     * Информация о посещениях
     */
    private VisitorsData visitorsData = new VisitorsData();

    /**
     * Конструктор класса
     */
    public VisitorsUpdater(Label uniqueLabel, Label totalLabel) {
        this.uniqueLabel = uniqueLabel;
        this.totalLabel = totalLabel;
    }

    /**
     * Обработчик нажатия на кнопку обновить. На данный момент копка отсутсвует, но
    обработчик используется
     * при перезагрузке страницы
     */
    @Override
    public void buttonClick(Button.ClickEvent event) {
        final WebBrowser webBrowser = Page.getCurrent().getWebBrowser();
        upsertAddress(webBrowser.getAddress());
        updateVisitors();
        uniqueLabel.setValue(Integer.toString(visitorsData.unique));
        totalLabel.setValue(Integer.toString(visitorsData.total));
    }

    /** Обновление информации при запуске класса как Runnable */
    @Override
```

```
    public void run() {
        buttonClick(null);
    }

    /**
     * Загрузить данные из базы данных
     */
    private void updateVisitors() {
        VisitorsDataService loader = new MongoVisitorsDataService();
        try {
            visitorsData = loader.getData();
        } catch (ExceptionForUser e) {
            Notification.show(e.what(), Notification.Type.ERROR_MESSAGE);
        }
    }

    /**
     * Сохранить IPадрес— в базе данных
     * @param ip IPадрес—
     */
    private void upsertAddress(String ip) {
        try {
            new MongoVisitorsDataService().registerIP(ip);
        } catch (ExceptionForUser e) {
            Notification.show(e.what(), Notification.Type.ERROR_MESSAGE);
        }
    }
}
```

WeatherUpdater.java

```
package com.xotonic.dashboard.ui;

import com.vaadin.ui.*;
import com.xotonic.dashboard.ExceptionForUser;
import com.xotonic.dashboard.weather.*;
import java.math.RoundingMode;
import java.text.DecimalFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;

/**
 * Обновление прогноза погоды<br>
 * @author xotonic
 */
public class WeatherUpdater implements Button.ClickListener, Runnable {

    /**
     * Информация о погоде в определенном населенном пункте
     */
    private WeatherData weatherData = new WeatherData();
    /**
     * НП, который будет загружаться по умолчанию
     */
    public final int defaultCityId = Cities.NSK.ordinal() — 1;
    /**
     * Выпадающий список с НП
     */
    private final ComboBox placeSelect;
```

```
/**
 * Лист наименований НП
 */
private final ArrayList<String> places;
/**
 * Надпись с текущей температурой
 */
private final Label currentTemperature;
/**
 * Надпись с температурой на завтра
 */
private final Label tomorrowTemperature;
/**
 * Надпись с временем последнего обновления
 */
private final Label timeStatusValueLabel;
/**
 * Метка, отключающая вывод сообщения пользователю об успешном обновлении
 */
private boolean silent;
/**
 * Формат числа для отображения градусов по Цельсию
 */
private final DecimalFormat weatherFormat;
/**
 * @param placeSelect компонент со списком городов ( Порядок городов должен
 * соответсвовать порядку в {@link Cities})
 * @param places массив со списком городов в том же порядке, что и у
placeSelect
 * @param currentTemperature текущая температура
 * @param tomorrowTemperature температура на завтра
 * @param timeStatusValueLabel время и дата, которое будет обновляться после
 * обновления погоды
*/
public WeatherUpdater(ComboBox placeSelect, ArrayList<String> places,
        Label currentTemperature,
        Label tomorrowTemperature,
        Label timeStatusValueLabel) {
    this.placeSelect = placeSelect;
    this.places = places;
    this.currentTemperature = currentTemperature;
    this.tomorrowTemperature = tomorrowTemperature;
    this.timeStatusValueLabel = timeStatusValueLabel;
    this.silent = true;
    weatherFormat = new DecimalFormat("#.#");
    weatherFormat.setRoundingMode(RoundingMode.CEILING);
    placeSelect.select(defaultCityId);
    placeSelect.setValue(places.get(defaultCityId));

}

/**
 * Обработчик события по клику на кнопку Обновить
 * @param event
 */
@Override
public void buttonClick(Button.ClickEvent event) {
    String selectedCity = (String) placeSelect.getValue();
    int id = places.indexOf(selectedCity);
    updateWeather(id);
    currentTemperature.setValue(weatherFormat.format(weatherData.
```

```
    celcium_today));
         tomorrowTemperature.setValue(weatherFormat.format(weatherData.
    celcium_tomorrow));
         if (silent == false) {
             Notification.showПогода(" обновлена", selectedCity, Notification.Type
    .HUMANIZED_MESSAGE);
         }
         updateDateLabel();

     }

     /**
      * Обновить текст с временем последнего обновления
      */
     private void updateDateLabel() {
         if (timeStatusValueLabel != null) {
             timeStatusValueLabel.setValue(
                     new SimpleDateFormat("dd/MM/yyyy HH:mm:ss")
                     .format(Calendar.getInstance().getTime()));
         }
     }

     /**
      * Обновить информацию о погоде
      * @param id номер населеного пункта
      */
     private void updateWeather(int id) {
         WeatherDataService loader = new FIODataService();
         try {
             weatherData = loader.getData(Cities.values()[id]);
         } catch (ExceptionForUser e) {
             Notification.show(e.what(), Notification.Type.ERROR_MESSAGE);
         }
     }
     /**
      * Эмуляция нажатия на кнопку обновления при запуске класса как Runnable
      */
     @Override
     public void run() {
         buttonClick(null);
     }

     /**
      *  Вернуть флаг, включающий оповещение пользователя об успешном обновлении
      */
     public boolean isSilent() {
         return silent;
     }

     /**
      * Поставить флаг, включающий оповещение пользователя об успешном обновлении
      */
     public void setSilent(boolean silent) {
         this.silent = silent;
     }
}
```

MongoVisitorsDataService.java

```
package com.xotonic.dashboard.visitors;
```

```
import com.mongodb.MongoClient;
import com.mongodb.MongoClientOptions;
import com.mongodb.MongoTimeoutException;
import com.mongodb.ServerAddress;
import com.mongodb.client.AggregateIterable;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.model.UpdateOptions;
import com.xotonic.dashboard.ExceptionForUser;
import org.bson.Document;

import static com.mongodb.client.model.Filters.eq;
import static com.mongodb.client.model.Updates.inc;
import static java.util.Arrays.asList;

/**
 * Загрузчикрегистратор/ числа посещений из БД MongoDB <br>
 * Порт, адрес, и БД сервера стандартные <br>
 * Собсно, требуется рабочий mongod <br>
 * Используется я3 версия драйвера
 * <p>
 * Формат документа<br>
 * <code>{ ip : string, count : integer}</code>
 * </p>
 *
 * @author xotonic
 */
public class MongoVisitorsDataService implements VisitorsDataService {

    /**
     * Адрес сервера БД
     */
    String MongoDBServerAddress = "localhost";
    /**
     * Имя БД
     */
    String dbName = "test";
    /**
     * Порт сервера БД
     */
    int MongoDBServerPort = 27017;

    /**
     * Устрановить соединение с сервером БД
     * @return
     * @throws ExceptionForUser
     */
    private MongoCollection connect() throws ExceptionForUser {
        System.out.println("CONNECTING TO MONGO SERVER");
        try {
            MongoClientOptions opts = MongoClientOptions
                    .builder()
                    .socketTimeout(5000)
                    .connectTimeout(3000)
                    .maxWaitTime(3000)
                    .build();
            ServerAddress addr = new ServerAddress(MongoDBServerAddress,
   MongoDBServerPort);
            MongoCollection collection
                    = new MongoClient(addr, opts)
                    .getDatabase(dbName)
                    .getCollection("visitors");
```

```
            System.out.println("CONNECTED SUCCESSFULLY");

            return collection;
        } catch (MongoTimeoutException e) {
            throw new ExceptionForUserHe(" удалось подключиться к серверу БД");
        }
    }

    /**
     * Получить информацию о посещениях из БД
     * @return
     * @throws ExceptionForUser
     */
    @Override
    public VisitorsData getData() throws ExceptionForUser {
        VisitorsData data = new VisitorsData();
        data.total = -1;
        data.unique = -1;
        MongoCollection<Document> collection
                = connect();
        System.out.println("AGRREGATING");

        AggregateIterable<Document> sum = collection.aggregate(asList(
                new Document("$group",
                        new Document("_id", null)
                        .append("total",
                                new Document("$sum", "$count"))
                        .append("unique",
                                new Document("$sum", 1)))));
        if (sum.first() != null) {
            data.total = sum.first().getInteger("total");
            data.unique = sum.first().getInteger("unique");
        }

        return data;
    }

    /**
     * Записать адрес в БД
     * @param ip IPадрес-
     * @throws ExceptionForUser
     */
    @Override
    public void registerIP(String ip) throws ExceptionForUser {
        try {
            MongoCollection col = connect();
            System.out.println("UPDATING IP COLLECTION");

            col.updateOne(
                    eq("ip", ip),
                    inc("count", 1),
                    new UpdateOptions().upsert(true));
        } catch (MongoTimeoutException e) {
            throw new ExceptionForUserHe(" удалось подключиться к серверу БД");
        }
    }

}
```

VisitorsData.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package com.xotonic.dashboard.visitors;

/**
 * Информация по статистике посещений
 * @author xotonic
 */
public class VisitorsData {
    /**
     * Число уникальных посетителей
     */
    public int unique;
    /**
     * Общее число посещений
     */
    public int total;
}
```

VisitorsDataService.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package com.xotonic.dashboard.visitors;

import com.xotonic.dashboard.ExceptionForUser;

/**
 * Интерфейс для службы по учету IPадресов— посетителей
 * @author xotonic
 */
public interface VisitorsDataService {
    VisitorsData getData() throws ExceptionForUser;
    void registerIP(String ip) throws ExceptionForUser ;
}
```

Cities.java

```
package com.xotonic.dashboard.weather;

/**
 * Список городов
 * @author xotonic
 */

public  enum Cities {
    /**
     * Москва
     */
    MSK,
    /**
     * СанктПетебург—
     */
```

```
    SPB,
    /**
     * Новосибирск
     */
    NSK;

    /**
     * Вернуть город по порядковому номеру в списке
     * @param id номер в списке
     * @return
     */
    public static Cities byID(int id) {
        return Cities.values()[id];
    }
}
```

FIODataService.java

```
package com.xotonic.dashboard.weather;

import com.github.dvdme.ForecastIOLib.*;
import com.xotonic.dashboard.ExceptionForUser;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.logging.*;
/**
 * Загрузчик прогноза погоды с forecast.io.<br>
 * Следует помнить, что сервис предоставляет ограниченноое количество запросов всего(
 * 1000 единовременно)
 * @author xotonic
 */
public class FIODataService implements WeatherDataService {

    /**
     * Бесплатный APIKEY ключ( разработчика)
     */
    private static final String APPKEY = "7a3f541cf9e9d01fd1be5eff81a80ee8";

    /** Широты городов */
    private static final String lats[] =
    {
        "55.7522200", // MSK
        "59.9386300", // SPB
        "55.0415000"  // NSK
    };
    /** Долготы городов */
    private static final String lons[] =
    {
        "37.6155600",// MSK
        "30.3141300",// SPB
        "82.9346000" // NSK
    };

    /**
     * Получить информацию о погоде в конкретном городе
     * @param city Запрашиваемый город
     * @return
     * @throws ExceptionForUser
     */
```

```
@Override
public WeatherData getData(Cities city) throws ExceptionForUser {
    WeatherData data = new WeatherData();
    data.city = city;
    data.celcium_tomorrow = -1;

    String lat = lats[city.ordinal()];
    String lon = lons[city.ordinal()];

    ForecastIO fio = new ForecastIO(APPKEY);
    fio.setUnits(ForecastIO.UNITS_SI);
    fio.setLang(ForecastIO.LANG_ENGLISH);
    fio.getForecast(lat, lon);
    /*
        При неудачной попыткой соединения с сервером
        библиотека кидает NullPointerException
    */
    try {
    System.out.println("Latitude: " + fio.getLatitude());
    System.out.println("Longitude: " + fio.getLongitude());
    System.out.println("Timezone: " + fio.getTimezone());
    } catch (NullPointerException e)
    {
        throw new ExceptionForUserОшибка(" подключения к серверу погоды");
    }
    //System.out.println("Offset: " + fio.getOffset());

    FIOCurrently currently = new FIOCurrently(fio);
    //Print currently data
    System.out.println("\nCurrently\n");
    /*String[] f = currently.get().getFieldsArray();
    for (int i = 0; i < f.length; i++) {
        System.out.println(f[i] + ": " + currently.get().getByKey(f[i]));
    }*/
    double currentT = currently.get().temperature();
    SimpleDateFormat parserSDF = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss")
;
    Date currentTDate = null;

    data.celcium_today = (float)currentT;

    /*
             Минимальная дата, которая может считаться следующий днем
     */
    Calendar cal = Calendar.getInstance();
    cal.add(Calendar.DAY_OF_YEAR, 1);
    cal.set(Calendar.HOUR_OF_DAY, 0);
    cal.set(Calendar.MINUTE, 0);
    cal.set(Calendar.SECOND, 0);
    Date tom = cal.getTime();

    try {
        currentTDate = (Date) parserSDF.parseObject(currently.get().time());
        System.out.format("%s %f\n", currentTDate.toString(), currentT);
    } catch (ParseException ex) {
        Logger.getLogger(FIODataService.class.getName()).log(Level.SEVERE,
null, ex);
    }
    FIODaily daily = new FIODaily(fio);
    //In case there is no daily data available
    if (daily.days() < 0) {
```

```
            System.out.println("No daily data.");
        } else {
            System.out.println("\nDaily:\n");
        }
        //Print daily data
        for (int i = 0; i < daily.days(); i++) {
            try {
                double t = (daily.getDay(i).apparentTemperatureMax() + daily.
   getDay(i).apparentTemperatureMin()) / 2;
                Date tDate = (Date) parserSDF.parseObject(daily.getDay(i).time()
   );
                //System.out.format("%s %f\n", tDate.toString(), t);
                if (tDate.after(tom))
                {
                    data.celcium_tomorrow = (float)t;
                    break;
                }
            } catch (ParseException ex) {
                Logger.getLogger(FIODataService.class.getName()).log(Level.
   SEVERE, null, ex);
            }
        }

        return data;
    }

}
```

WeatherData.java

```
package com.xotonic.dashboard.weather;

/**
 * Класс для передачи информации от загрузчиков погодных информеров
 * ({@link WeatherDataService})
 * @author xotonic
 */
public class WeatherData {
    /**
     * Значение температуры за сегодня в градусах по Цельсию
     */
    public float celcium_today;
    /**
     * Значение температуры на завтра в градусах по Цельсию
     */
    public float celcium_tomorrow;
    /**
     * Город, для которого предназначен прогноз
     */
    public Cities city;
}
```

WeatherDataService.java

```
package com.xotonic.dashboard.weather;

import com.xotonic.dashboard.ExceptionForUser;
import com.xotonic.dashboard.ui.DashboardUI;
```

```
/**
 * Интерфейс для взаимодействия с {@link DashboardUI}
 * @author xotonic
 */
public interface WeatherDataService {

    /**
     * @param city Запрашиваемый город
     * @return Информация по этому городу
     * @throws com.xotonic.dashboard.ExceptionForUser
     */
    WeatherData getData(Cities city) throws ExceptionForUser;
}
```