TRANSPORTA UN SAKARU INSTITŪTS

КУРСОВОЙ ПРОЕКТ

ТЕМА: “БЮДЖЕТНЫЙ ПЛАНИРОВЩИК”

ДИСЦИПЛИНА: “ТЕХНОЛОГИЯ КОНСТРУИРОВАНИЯ ПО”

Выполнил: Maksims Voitovs
4 курс, 4001BD
St53825
Проверил(-а): (lekt., doc., asoc. prof., prof.) Vārds Uzvārds

RĪGA – 2013

СОДЕРЖАНИЕ

Цель работы

Целью данной курсовой работы является разработка программного обеспечения для планирования бюджета по средствам интернет интерфейса в соответствии с требованиями, предъявляемыми к унифицированному процессу разработки программного обеспечения. Визуальная разработка производилась в среде Rational Rose 2003, программная реализация (кодирование) в среде программирования Netbeans.

Предметная область

Система предоставляет хранение и изменение схем доходов и расходов. Система позволяет создавать учетную запись клиента и заполнять поля схем доходов и расходов распланированную на время от одного дня до нескольких лет, с и без изменения заполненных полей схемы. Схемы доходов и расходов разделены на типы в зависимости от области затрат и доходов.

Система поддерживает два вида ввода записей (данных) доходов и расходов. Имеется два вида: первый вид - заполнение полей на “лету”, что означает, что клиент системы заполняет поля по мере их появления (затрат и доходов) и выставляет периодичность этих полей в зависимости от реальной периодичности их появления, второй вид - заполнение полей в режиме “планирования” - что подразумевает под собой, что клиент планирует бюджет на какой-то определенный период времени и заполняет все поля доходов и расходов с периодичностью заданной в зависимости от времени на которую клиент рассчитывает распланировать свой бюджет. Система поддерживает функцию добавления своих полей в схемы расходов, которые будут учитываться при расчете бюджета. После расчета бюджета система позволяет выбрать желаемый метод получение готового бюджета: высылка на электронную почту или скачивание на накопитель компьютера клиента. Предполагаться общение с программой по средствам программного интерфейса мобильного телефона клиента.

Этап «Начало»

Определить набор требований, предъявляемых к разрабатываемому продукту, после первой встречи с заказчиком.

Программный продукт представляет собой бюджетный планировщик. Данная система должна выполнять следующие функции:

1. Добавление информации о доходах клиента
2. Добавление информации о расходах клиента
3. Подсчет схем доходов и расходов клиента по полученной информации о доходах и расходах от клиента

Идентификация актёров и вариантов использования:

В соответствии с поставленными требованиями, определим актера, фиксирующего роли внешних объектов, взаимодействующего с системой.

Описание актёров:

Актёр Client – описывает пользователя работающего с бюджетным планировщиком, производящий регистрацию и авторизацию в систему, а также выполняющего добавление и\или удаление и\или изменение полей схем доходов и расходов и получающий готовый бюджет на конкретное составленный на конкретное время.

Для актера Client определим соответствующие USE-CASE’ы, путем рассмотрения актера и его взаимодействия с системой. На рисунке 1 представлена Use Case диаграмма построенная на основании требований заказчика.

Актёр User инициирует только USE-CASE: Enter, остальные USE-CASE’ы инициируются по цепочке.

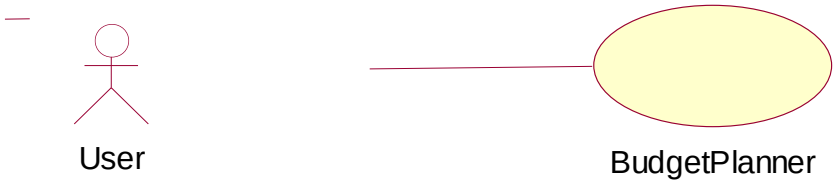


Рис.1 Начальная UseCase диаграмма (20% от полного представления)

Описание UseCase’ов:

Актёр User инициирует следующие USE-CASE’ы:

- **Enter** – USE-CASE элемент управления входом в систему, а именно: авторизацией и регистрацией клиента.
- **BudgetPlanner** – USE-CASE элемент управления расчетами, вводом, выводом и изменением информации.

Этап «Развитие»

На данном этапе разрабатываются сценарии работы программной системы для каждого из Use-case’ов, строятся диаграммы последовательности для каждого из сценариев, на основании объектов, входящих в диаграммы последовательности, определяются их абстракции – классы и осуществляется планирование итераций этапа Конструирование.

На этом этапе, после разговора с заказчиком возникла необходимость дополнительных функциональных возможностей системы. В итоге был определён дополнительный набор требований, предъявляемых к разрабатываемому продукту:

- Исправление информации о доходах и расходах клиента
- Просмотр схем доходов и расходов клиента
- Регистрация и вход клиента в систему
- Вывод рассчитанного бюджетного плана по схемам доходов и расходов

В соответствии с уточненными требованиями, опишем дополнительные функциональные возможности системы:

Описание UseCase’ов:

- **Registration** – USE-CASE элемент регистрации клиента в системе (теперь является частью BudgetPlanner).
- **Login** – USE-CASE элемент авторизации клиента в системе (теперь является частью BudgetPlanner).
- **BudgetPlanner** – USE-CASE элемент управления расчетами, вводом, выводом и изменением информации.
- **InformationInput** – USE-CASE элемент ввода информации в систему.
- **IncomeCalculation** – USE-CASE элемент подсчета суммы схем доходов.
- **OutcomeCalculation** – USE-CASE элемент подсчета суммы схем расходов.
- **InformationEditing** – USE-CASE элемент изменения и добавления полей схем расходов и доходов.
- **InformationOutput** – USE-CASE элемент вывода информации схем доходов и расходов.

В итоге на этапе развитие конечная UseCase диаграмма примет вид (рис.2):

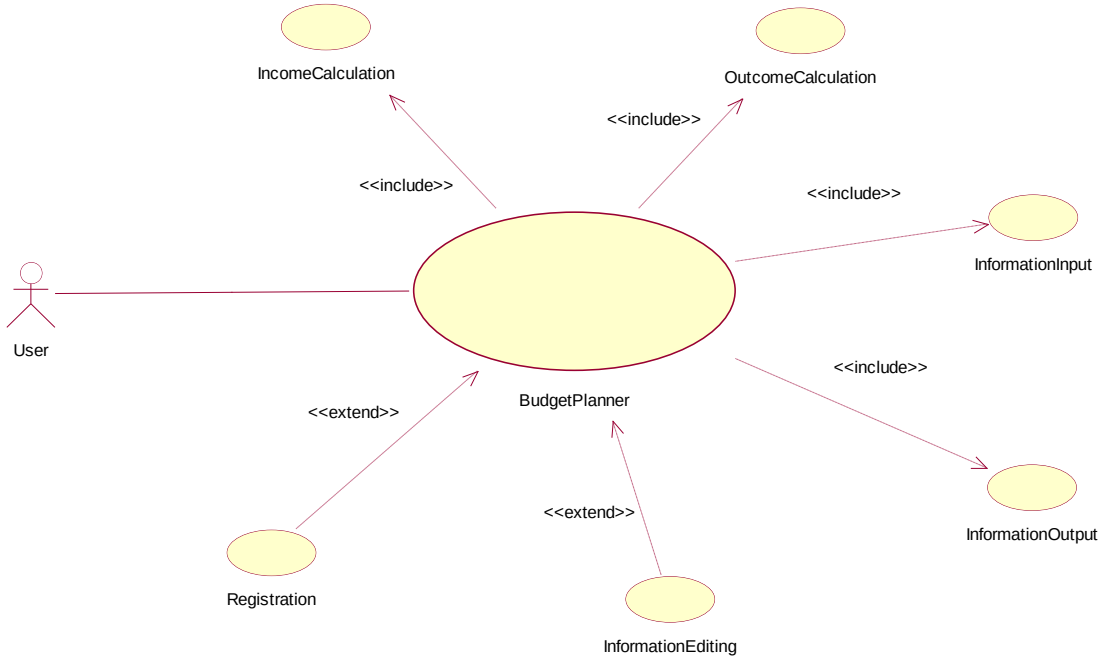


рис.2 КонечнаяUseCase диаграмма (80% от полного представления)

Определение сценариев:

- СценарийдляUse-Case'a Registration:
  Открывая страницу клиент тем самым инициализирует процесс регистрации, после того как клиент заполнит все поля регистрации и нажмет на кнопку «подтвердить» форма регистрации отправит данные из полей на валидацию, если валидация пройдет успешно, то затем следует процесс проверки имени пользователя на уникальность (имеется ли в уже в базе), в случае, если такого имени нету, то менеджер базы данных добавляет новые поля в базу данных, после чего инициализирует страницу оповещения о удачном завершении регистрации, которая в свою очередь ведет за собой инициализацию (открытие) страницы входа в систему, после того как клиент заполнит поля формы входа в систему, форма передаст данные на проверку имеются ли такой пользователь, дальше данные попадут на заявку на создание сессии, после чего, сессионный менеджер запросит у менеджера базы данных данные требуемые для отображения схем доходов и расходов.
- СценарийдляUse-Case'a IncomeCalculation:
  После того как первый раз, при создании новой схемы бюджета, клиент заполнил хотя бы одно поля схемы доходов, создаётся данные о этой схеме и каждый раз, когда эта схема корректируется клиентом, автоматически происходит пересчет схемы доходов и сохранение её в базу данных с помощью менеджера базы данных.
- СценарийдляUse-Case'a OutcomeCalculation:
  После того как первый раз, при создании новой схемы бюджета, клиент заполнил хотя бы одно поля схемы расходовсоздается, данные о этой схеме и каждый раз, когда эта схема корректируется клиентом, автоматически происходит пересчет схемы расходов по все страницам бюджетного планировщика и сохраняет схему расходов в базу данных с помощью менеджера базы данных.
- СценарийдляUse-Case'a InformationInput:
  Открытие страницы инициализирует процесс создания схемы доходов и расходов с пустыми полями схем, после заполнения хотя бы одного поля любой из схем, поля записываются в

структуру полей данных, после этого клиент нажимает кнопку подтверждения, и структура схем доходов и расходов сохраняется в базе данных после того как выполнится сценарий IncomeCalculation и OutcomeCalculation.

- СценарийдляUse-Case’a InformationOutput:
  Открытие страницы инициализирует обращение запроса данных о схемах бюджета главной страницы к менеджеру базы данных за этим следует выборка из базы данных, информации о полях схем бюджета, главная страница бюджетного планировщика также передает идентификатор названия бюджета, после чего менеджер базы данных передает главной странице данные полей схемы бюджета и выводит заполненные поля на страницах бюджета.
- СценарийдляUse-Case’a InformationEditing:
  При открытии страницы на которой необходимо изменить схему доходов или расходов, выполняется сценарий InformationOutput, затем после того как клиент изменит требуемое значение поля схемы, форма страницы проверяет введенные данные, как только клиент подтвердит изменение схемы нажав на кнопку подтверждения, форма передаст данные менеджеру базы данных, менеджер в свою очередь сверит идентификатор названия бюджетной схемы и идентификатор клиента, если все идентификаторы соответствуют реальным и они существует, менеджер базы данных сохранит схему бюджета под указанным названием.

Разработка диаграмм последовательности

Разработка диаграммы последовательности ведется отдельно для каждого Use-case’a.

Диаграмма последовательности для Use-Case’a Login выглядит следующим образом (рис.3):

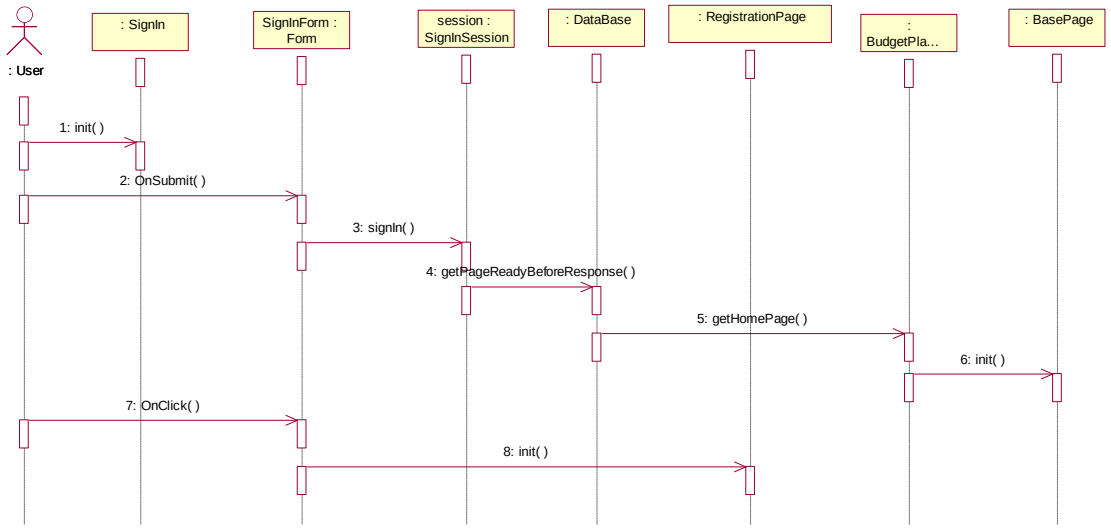


рис.3

Диаграмма последовательности для Use-Case’a Registration выглядит следующим образом (рис.4):

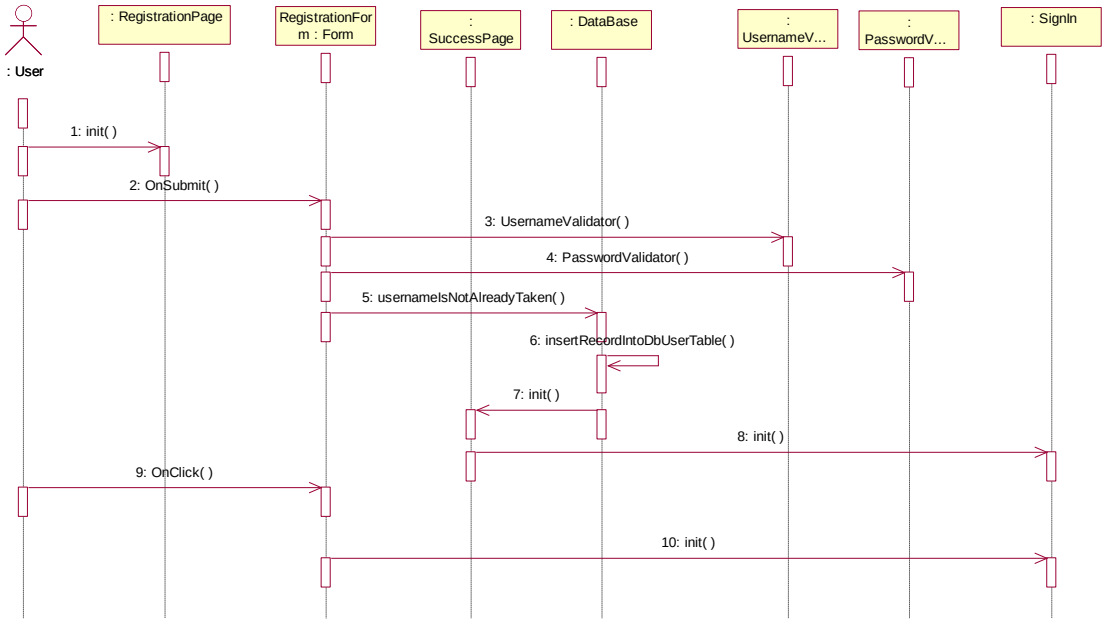

: User
: RegistrationPage
RegistrationForm : Form
: SuccessPage
: DataBase
: UsernameV...
: PasswordV...
: SignIn
1: init( )
2: OnSubmit( )
3: UsernameValidator( )
4: PasswordValidator( )
5: usernameIsNotAlreadyTaken( )
6: insertRecordIntoDbUserTable( )
7: init( )
8: init( )
9: OnClick( )
10: init( )

рис.4

Диаграмма последовательности для Use-Case'а IncomeCalculation выглядит следующим образом (рис.5):

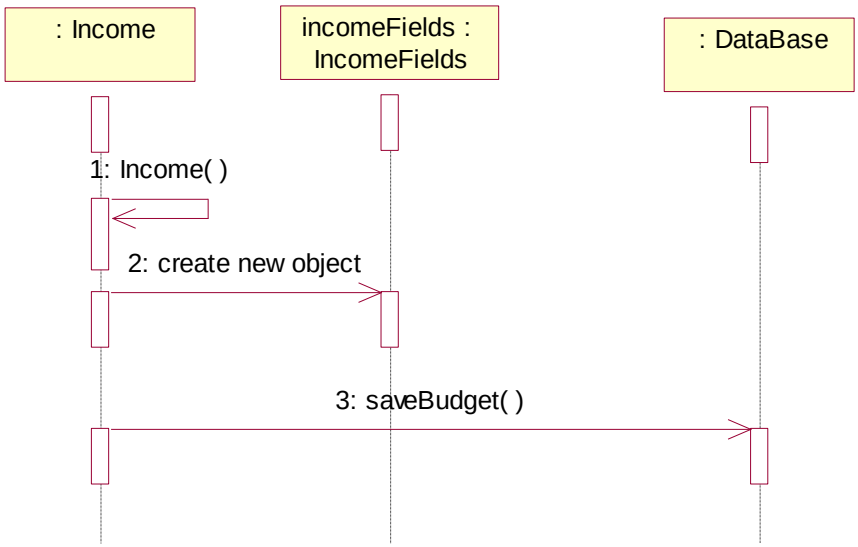


рис.5

Диаграмма последовательности для Use-Case'а OutcomeCalculation выглядит следующим образом (рис.6):

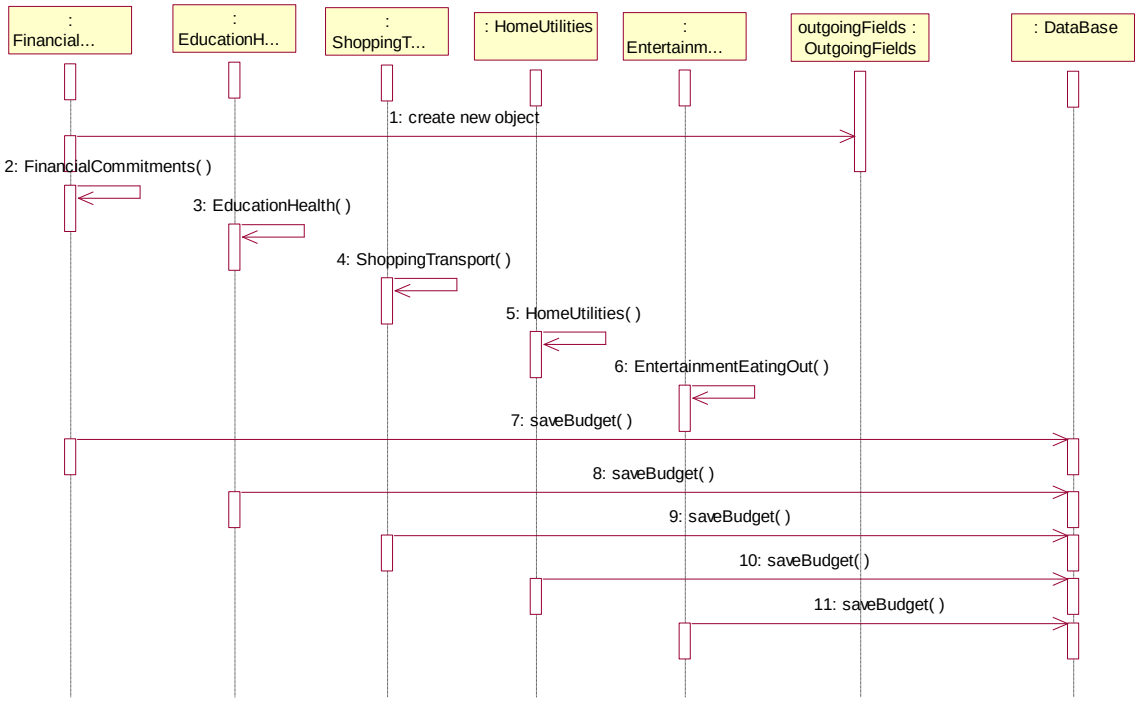


рис.6

Диаграмма последовательности для Use-Case’a InformationInput выглядит следующим образом (рис.7):

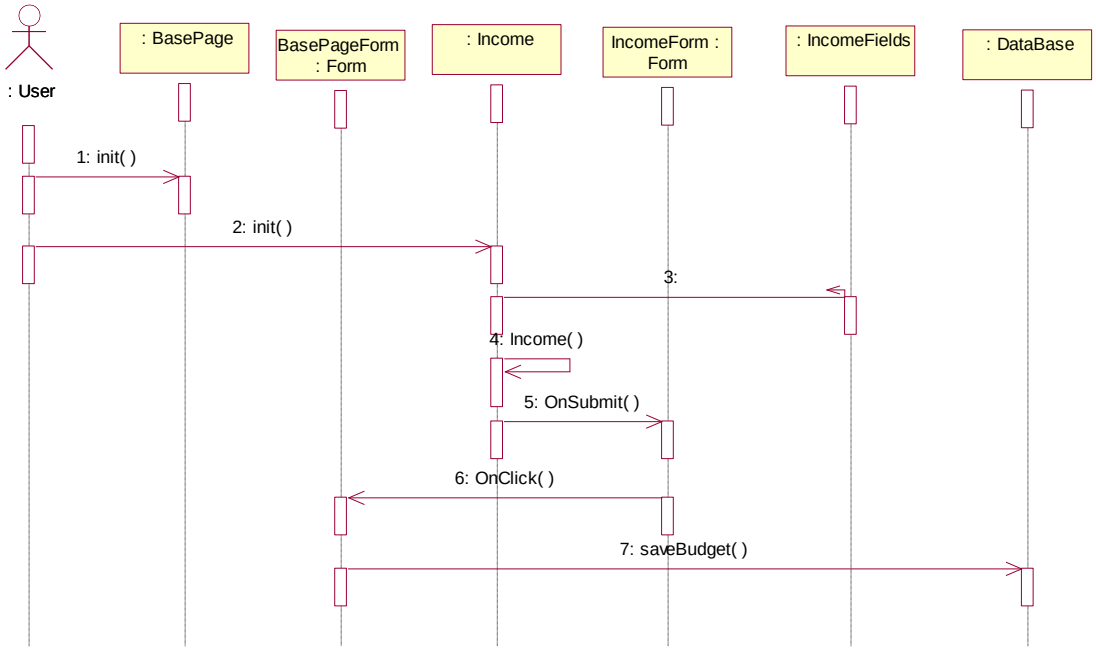


рис.7

Диаграмма последовательности для Use-Case’a InformationEditting выглядит следующим образом (рис.8):

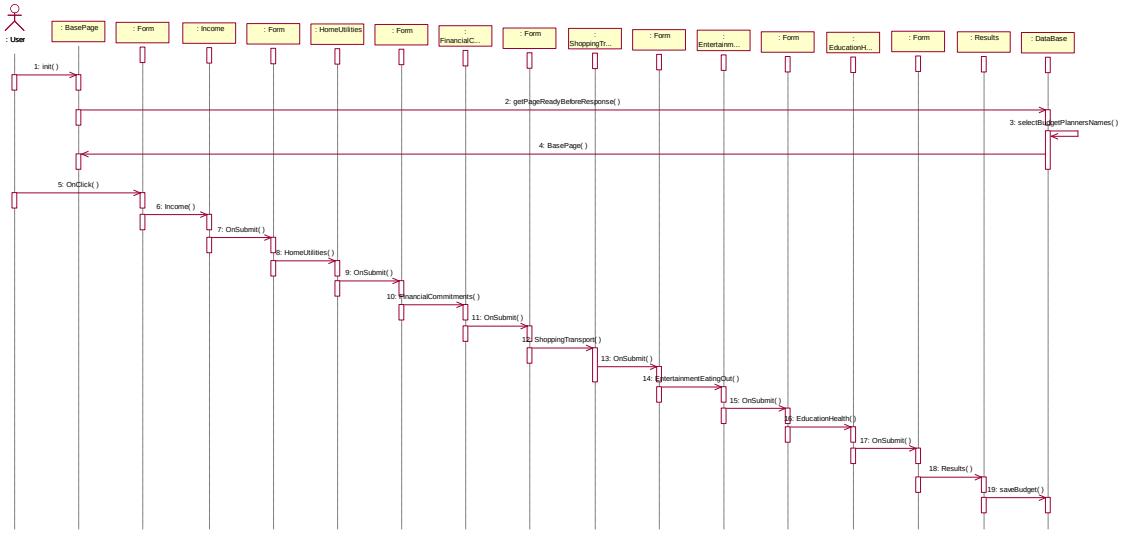


рис.8

Диаграмма последовательности для Use-Case’a InformationOutput выглядит следующим образом (рис.9):

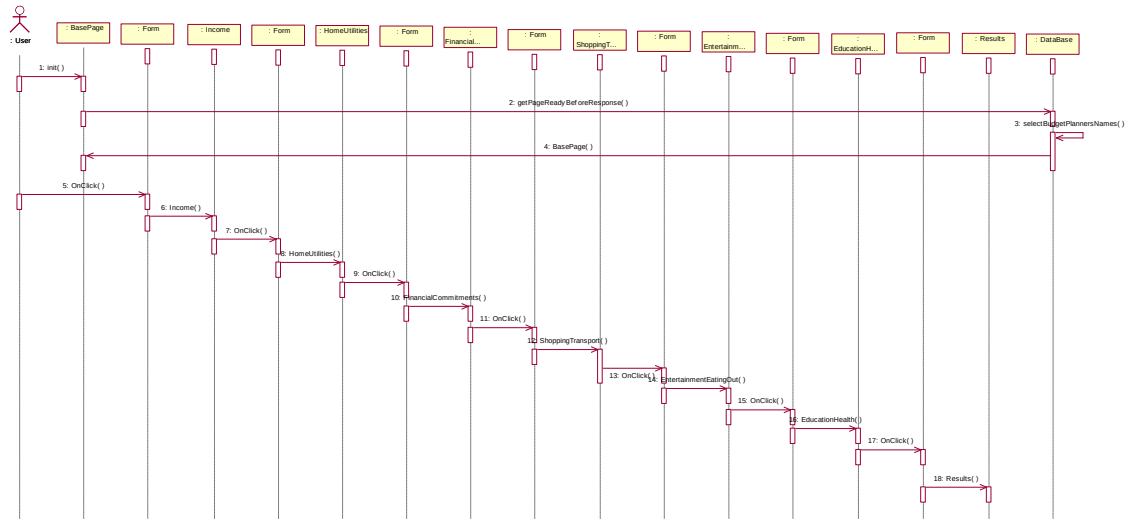


рис.9

Примечание: все представленные диаграммы последовательностей являются конечными.

Диаграмма классов

На основании разработанных диаграмм последовательностей возможно выделить следующие классы:

- **BasePage**-объектом класса является главная страница.
- **SuccessPage –** объектом класса является страница удачно проведенной регистрации.
- **BudgetCreationPage –** объектом класса является страница создания нового бюджета.
- **RegistrationPage –** объектом класса является страница регистрации пользователя в системе.
- **SignIn –** объектом класса является страница входа в систему.
- **SignOut –** объектом класса является страница выхода из системы.

Все создаваемые классы являются наследниками стандартного класса WebPage языка Java.

Также были созданы следующие классы, являющиеся наследниками стандартного класса BasePage:

- **Results**
- **ShoppingTransport**
- **EntertainmentEatingOut**
- **HomeUtilities**
- **Income**
- **EducationHealth**
- **FinancialCommitments**

Так же важно отметить, что возникла необходимость в создании следующих классов, имеющих стереотип запись:

- **IncomeFields**-класс содержащий такие атрибуты, как название, частоту и значение поля ввода информации схемы доходов.
- **OutgoingFields**- класс содержащий такие атрибуты, как название, частоту и значение поля ввода информации схемы расходов.
- **ResultFields**- класс содержащий конечное значение рассчитанного бюджета.

И другие вспомогательные классы:

- **UsernameValidator**
- **PasswordValidator**
- **SignInSession**
- **AuthenticatedWebSession**
- **MyDropDownChoice**
- **BudgetPlannerApplication**
- **WebApplication**

Планирование итераций конструирования

На данной стадии выполнения работ производится разбиение процесса конструирования на отдельные итерации с целью обеспечения возможности управления риском в процессе разработки, а также для определения очередности выполнения работ с точки зрения архитектурного построения программной системы. Обычно планирование итераций производится путем разбиения общего объема работ по Use-case'ам и, если требуется, далее по сценариям, входящим в отдельные Use-case'ы.

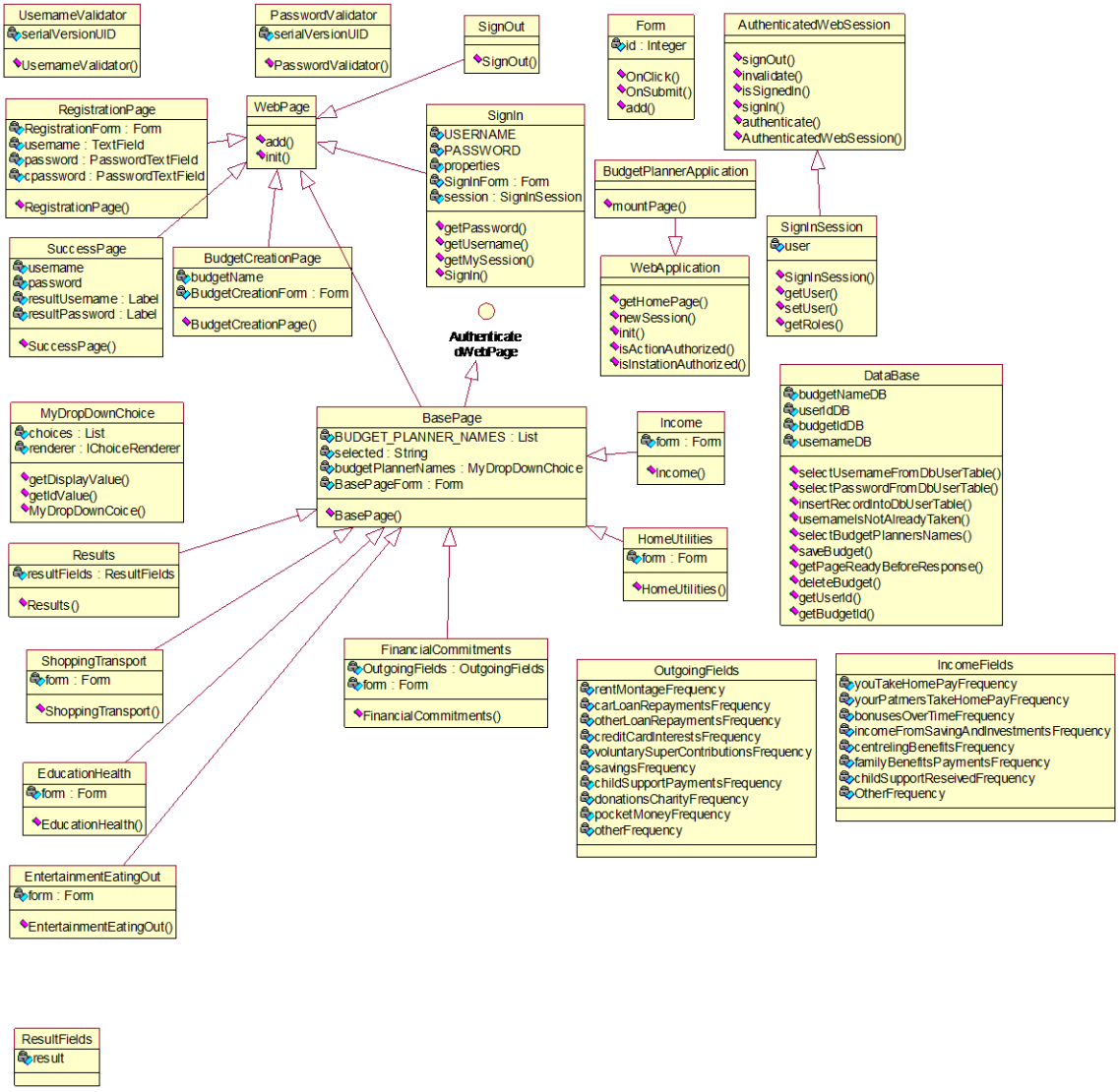


Рис. 10. Предварительная диаграмма классов

Приведем предварительную таблицу значений метрик, характеризующих качество разрабатываемой программной системы:

Метрика	BudgetCreationPage	BasePage	RegistrationPage	SuccessPage	SignIn	SignOut	MyDropDownChoice	BudgetPlannerApplication	SignInSession	DataBase	Results	Income	HomeUtilities	FinancialCommitments	ShoppingTransport	EducationHealth	EntertainmentEatingOut	ResultFields	OutgoingFields	IncomeFields	UsernameValidator	PasswordValidator	Среднее значение
WMC	1	1	1	1	4	1	3	1	4	10	1	1	1	1	1	1	1	0	0	0	1	1	1,7
NOC	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,3
	Метрики, вычисляемые для системы																						
DIT	3																						
NC	26																						
NOM	34																						

Составим начальный план итераций с учётом риска реализации usecase’ов:

Итерация №1

1. Аутентификация пользователя
2. Регистрация пользователя

Итерация №2

1. Добавление информации доходов\расходов
2. Изменение информации доходов\расходов
3. Создание/удаление бюджета
4. Вывод информации доходов\расходов

Итерация №3

1. Добавление информации доходов\расходов
2. Изменение информации доходов\расходов
3. Подсчет информации доходов\расходов
4. Вывод информации доходов\расходов
5. Создание/удаление бюджета

Этап Конструирование

Итерация 1:

Для реализации сценариев

- Use Case’aLogin
- Use Case’aRegistration

необходимо создать программный код для следующих методов классов, представленных на диаграмме ниже (Рис.13):

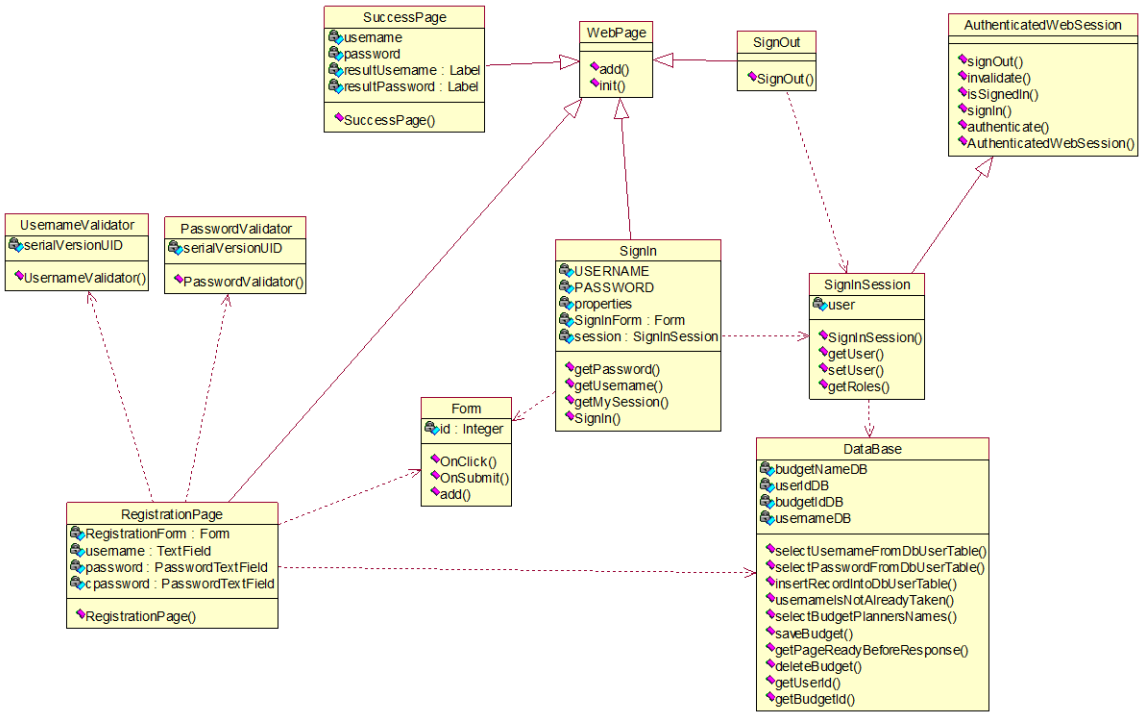


рис.11 Диаграмма классов на первой итерации

Примечание: классы Form, WebPage, AuthenticatedWebSession –не создавались, это классы языка Java.

Оценка качества логической структуры модели с помощью метрик Чидамбера – Кемерера

Описание процесса подсчёта метрик на первой итерации:

Нахождение WMC: Пусть, начальный вес каждого метода равен единице. Тогда, вес взвешенных методов будет равен их количеству.

Нахождение NOC: Стандартные классы WebPage, AuthenticatedWebSession имеют наследников (WebPage – 4, AuthenticatedWebSession - 1).

Нахождение количества сцеплений между классами объектов (CBO метрика):

CBO- количество сотрудничеств, предусмотренных для класса (количество классов с которыми он соединен). В нашем случае для класс Registration значение будет равно пяти, так как класс используют методы или экземплярные переменные стандартных классов языка Java.

Расчёт метрик RFC, OSavg, Npavg

Класс Registration

Метод	Вызываемые методы(RFC)	Кратность	Значение OSavg	Значение NPavg
RegistrationPage	**-**	**0**	**0**	**1**
OnSubmit	**Form submit**	**1**	**1**	**1**
UsernameValidation	**-**	**0**	**0**	**1**
PasswordValidation	**-**	**0**	**0**	**1**
InsertRecordIntoDbUserTable	**Database input**	**1**	**1**	**1**
количество/ среднее	**2**		**0,2**	**1**

Расчёт метрики LCOM

Класс Registration

Число пар методов – 0
СВЯЗАНЫ = 0
НЕСВЯЗАНЫ = 0
LCOM = 0

Класс SuccessPage

Число пар методов – 0
СВЯЗАНЫ = 0
НЕСВЯЗАНЫ = 0
LCOM = 0

Класс SignOut

Число пар методов – 0
СВЯЗАНЫ = 0
НЕСВЯЗАНЫ = 0
LCOM = 0

Класс Database

Число пар методов – 0
СВЯЗАНЫ = 0
НЕСВЯЗАНЫ = 0
LCOM = 0

Класс SignInSession

Число пар методов – 0
СВЯЗАНЫ = 0
НЕСВЯЗАНЫ = 0
LCOM = 0

Класс SignIn

Число пар методов – 0
СВЯЗАНЫ = 0
НЕСВЯЗАНЫ = 0
LCOM = 0

Расчёт метрики NOO – количество операций, переопределяемых подклассом.

Переопределения отсутствуют.

Расчёт метрики NOA – количество операций, добавленных подклассом. Подклассы специализируются добавлением приватных операций и свойств. Приватных операций и свойства отсутствуют.

Расчёт метрики SI – индекс специализации. Специализация достигается добавлением, удалением или переопределением операций.

$$SI = (NOO * \text{уровень})/M_{общ} \text{ , где}$$

Уровень - номер уровня в иерархии, на котором находится подкласс.

$M_{общ}$ - общее количество методов класса.

Для класса Registration

$SI = (0 * 1)/1 = 0$

Оценка качества на первой итерации представлена в таблице ниже:

Метрика	Registration	Database	SuccessPage	SignOut	SignIn	SignInSession	UsernameValidator	PasswordValidator	Среднее значение
WMC	**1**	**10**	**1**	**1**	**4**	**4**	**1**	**1**	**2,875**
NOC	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
CBO	**4**	**0**	**0**	**1**	**2**	**1**	**0**	**0**	**1**
RFC	**0**	**2**	**0**	**0**	**0**	**2**	**1**	**1**	**0,75**
LCOM	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
CS (a/o)	**4/1**	**4/10**	**4/1**	**0/1**	**5/4**	**1/4**	**1/1**	**1/1**	**2.5/2.9**
NOO	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
NOA	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
SI	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
OSavg	**0,2**	**0,4**	**0,2**	**0**	**1,3**	**0,25**	**1**	**1**	**0,2**
NPavg	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
DIT	**1**								
NC	**8**								
NOM	**23**								
LOCå	**539**								

Итерация 2:

Для реализации сценариев

- UseCase’a InformationInput
- Use Case’aInformationOutput
- Use Case’aInformationEditting

необходимо создать программный код для следующих методов классов, представленных на диаграмме ниже (Рис.14):

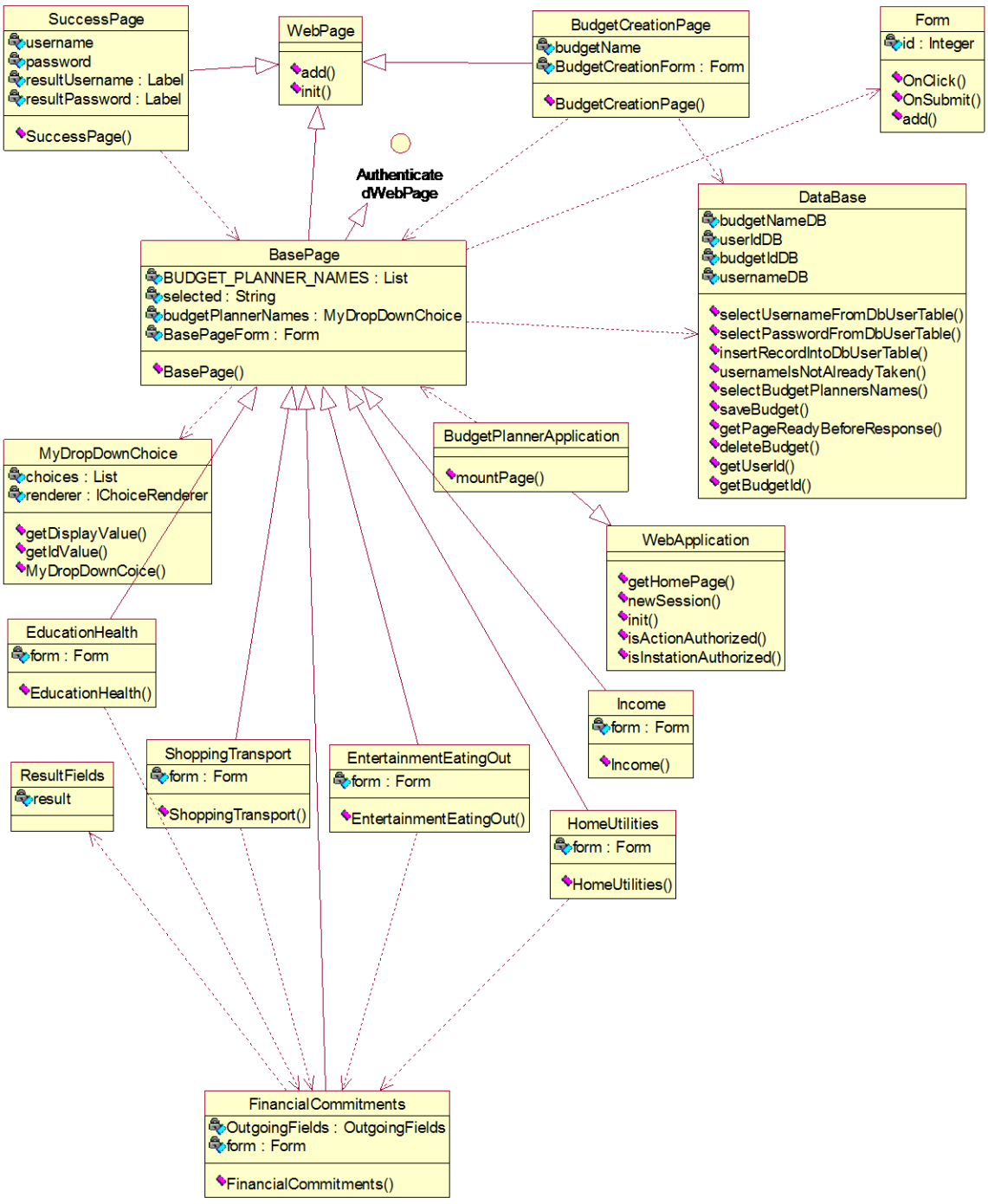


рис.12 Диаграмма классов на второй итерации

Для оценки качества проведенной разработки выполним подсчет значения метрик для реализованных классов и системы в целом:

Метрика	BasePage	BudgetCreationPage	BudgetPlannerApplication	MyDropDownChoice	Database	EducationHealth	ShoppingTransport	EntertainmentEatingOut	HomeUtilities	Income	ResultFields	FinancialCommitments	Среднее значение
WMC	**1**	**1**	**1**	**3**	**10**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**1,833333333**
NOC	**6**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0,5**
CBO	**3**	**2**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**0,916666667**
RFC	**3**	**0**	**0**	**1**	**2**	**0**	**0**	**0**	**0**	**0**	**1**	**4**	**0,916666667**
LCOM	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
CS (a/o)	**4/1**	**2/1**	**0/1**	**2/3**	**4/10**	**1/1**	**1/1**	**1/1**	**1/1**	**1/1**	**1/0**	**2/1**	**1,66667/1,83333**
NOO	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
NOA	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
SI	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
OSavg	**1,75**	**2**	**2**	**2,6**	**2,0875**	**2**	**2**	**2**	**2**	**2**	**2**		**2,039772727**
NPavg	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
DIT	**2**												
NC	**12**												
NOM	**22**												
LOCå	**1044**												

Итерация 3:

Дляреализациисценариев

- Use Case’a IncomeCalculation
- Use Case’aOutcomeCalculation
- UseCase’a InformationInput
- Use Case’a InformationOutput
- Use Case’a InformationEditting

Необходимо создать программный код для следующих методов классов, представленных на диаграмме ниже (Рис.15):

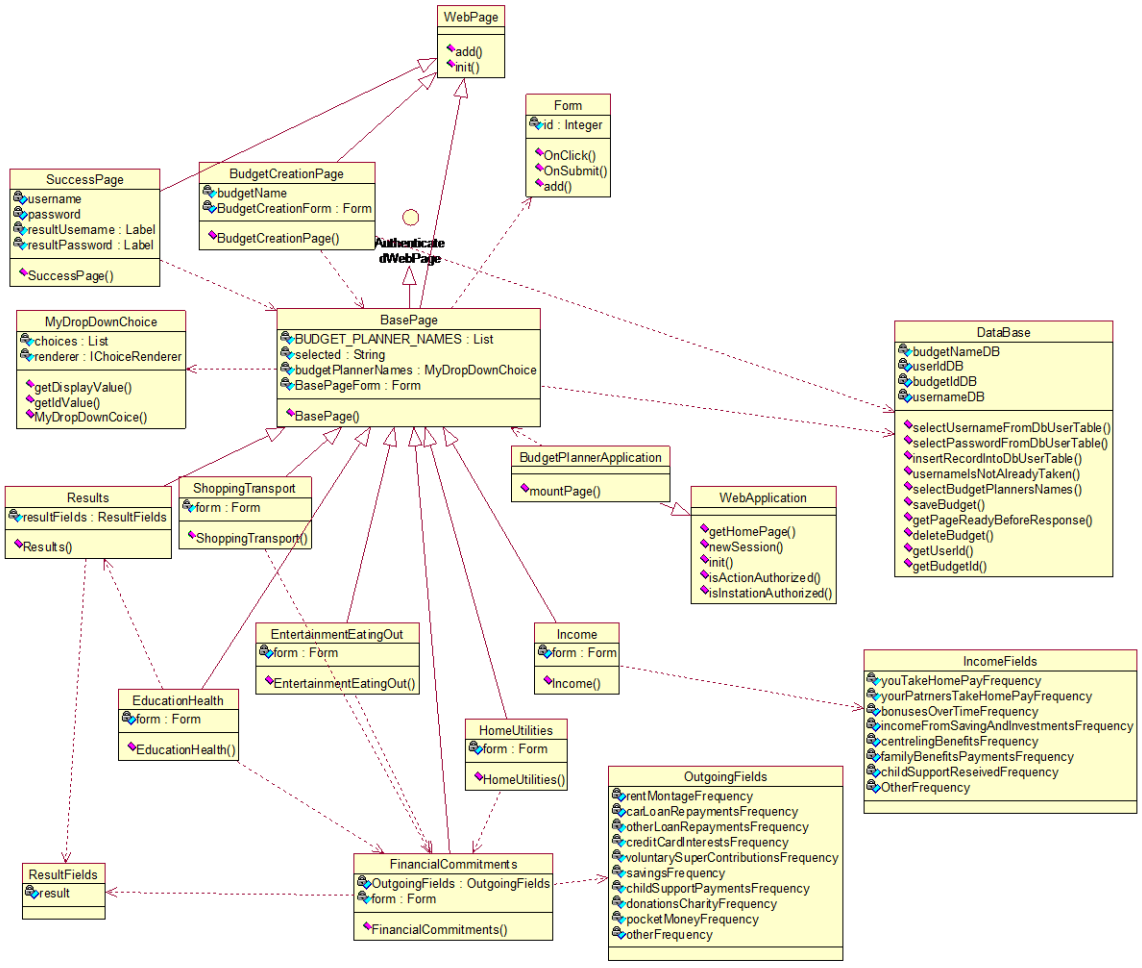


рис.13 Диаграмма классов на третьей итерации

Для оценки качества проведенной разработки выполним подсчет значения метрик для реализованных классов и системы в целом:

Метрика	BasePage	BudgetCreationPage	BudgetPlannerApplication	MyDropDownChoice	Database	EducationHealth	ShoppingTransport	EntertainmentEatingOut	HomeUtilities	Income	ResultFields	FinancialCommitments	OutgoinFields	IncomeFields	Среднее значение
WMC	1	1	1	3	10	1	1	1	1	1	0	1	0	0	1,571428571
NOC	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0,428571429
CBO	3	2	1	0	0	1	1	1	1	0	0	1	0	0	0,785714286
RFC	3	0	0	1	2	0	0	0	0	0	1	4	1	1	0,928571429
LCOM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CS (a/o)	4/1	2/1	0/1	2/3	4/10	1/1	1/1	1/1	1/1	1/1	1/0	2/1	52/0	52/0	5,538492/1,83333
NOO	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
NOA	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SI	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
OSavg	1,75	2	2	2,6	2,0875	2	2	2	2	2	0	1	0	0	1,53125
NPavg	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0,785714286
DIT	2														
NC	14														
NOM	22														
LOCå	1238														

Далее сведем в таблицу полученные значения метрик на разных итерациях и сравним полученные результаты.

Значения метрик на разных итерациях

Метрика	Итерация 1	Итерация 2	Итерация 3
WMC	2,875	1,833333333	1,571428571
NOC	0	0,5	0,428571429
CBO	1	0,916666667	0,785714286
RFC	0,75	0,916666667	0,928571429
LCOM	0	0	0
CS	2.5/2.9	1,66667/1,83333	5,538492/1,83333
NOO	0	0	0

NOA	**0**	**0**	**0**
SI	**0**	**0**	**0**
OSavg	**0,2**	**2,039772727**	**1,53125**
Npavg	**1**	**1**	**0,785714286**
DIT	**2**	**2**	**2**
NC	**4**	**4**	**14**
NOM	**3**	**14**	**22**
LOCsum	**164**	**477**	**1238**

Сравнивая полученные значения метрик, можно заметить, что некоторые их значения увеличились, а это в свою очередь свидетельствует о постепенном возрастании сложности продукта, что в принципе вполне естественно.

Примечание: метрики для классов языка Java не подсчитывались, так как мы их не создавали.

Оценка качества проектирования

С ростом CBO многократность использования класса, вероятно, уменьшается. Высокое значение CBO усложняет модификацию и тестирование. CBO для каждого класса должно иметь разумно низкое значение.

Метрика RFC если в ответ на сообщение может быть вызвано большое количество методов, то усложняется тестирование и отладка класса.

Метрика LCOM показывает насколько методы не связаны друг с другом через свойства (переменные).

CS не превышает рекомендуемого значения CS ≤ 20 методов.

Отсутствие NOO указывают на то, что проблем с проектированием нет, что разработчик не нарушает абстракцию суперкласса. Это не ослабляет иерархию классов, не усложняет тестирование и модификацию программного обеспечения. Рекомендуемое значение ≤3 методов не превышено.

Отсутствие NOA указывают на то, что подкласс не удаляется от абстракции суперкласса. Для рекомендуемых значений CS = 20 и DIT = 6 рекомендуемое значение NOA≤ 4 методов (для класса-листа). Рекомендуемое значение не превышено.

Отсутствие SI указывают на то, что нет вероятности того, что в иерархии классов есть классы, нарушающие абстракцию суперкласса. Рекомендуемое значение SI ≤ 0.15 не превышено.

Чем больше параметров у операции, тем сложнее сотрудничество между объектами. Поэтому значение NP_{avg}должно быть, как можно меньшим. Рекомендуемое значение $NP_{avg} = 0.7$. немного превышено, но не значительно.

Ниже представлена общая диаграмма классов для всего проекта (Рис.16):

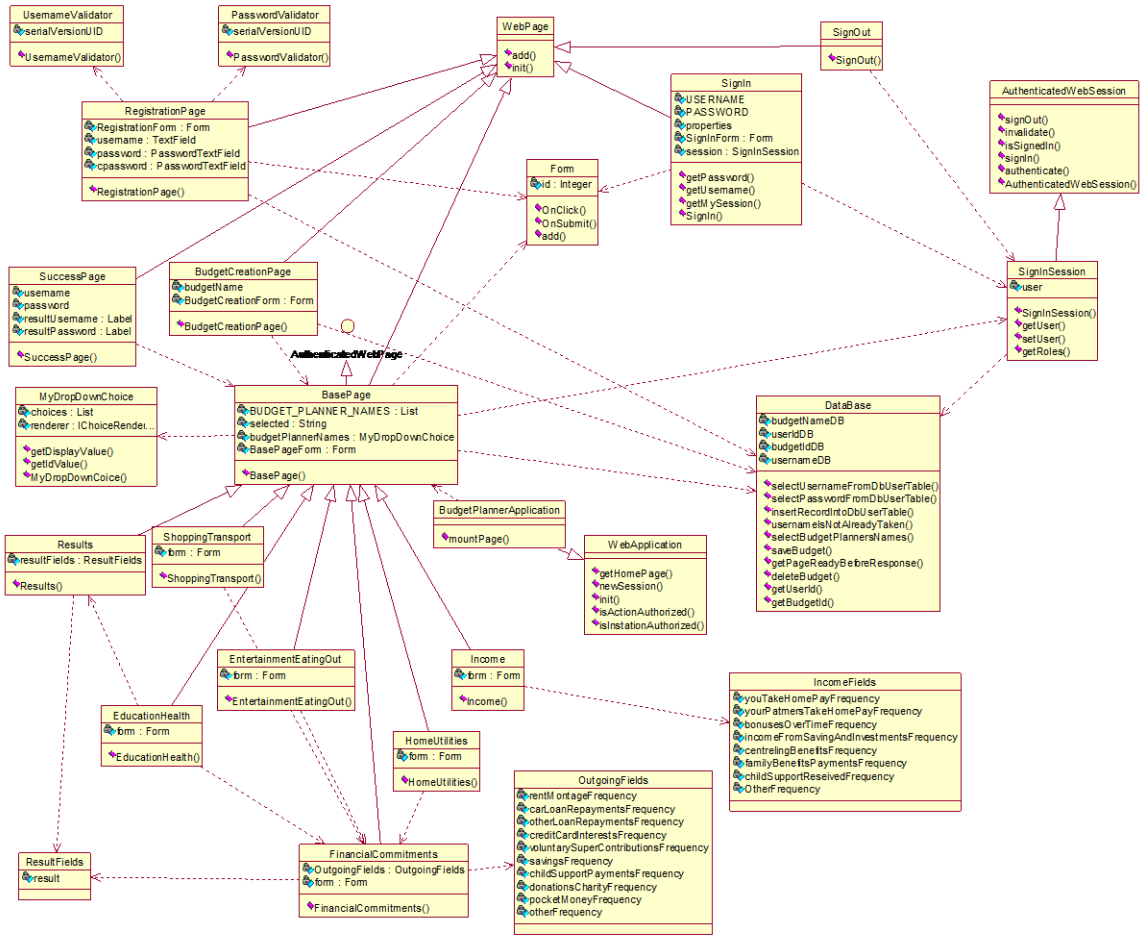


рис.14Общая диаграмма классов для всего проекта

Диаграмма компонент

На рисунке представлена компонентная диаграмма (Рис.17).

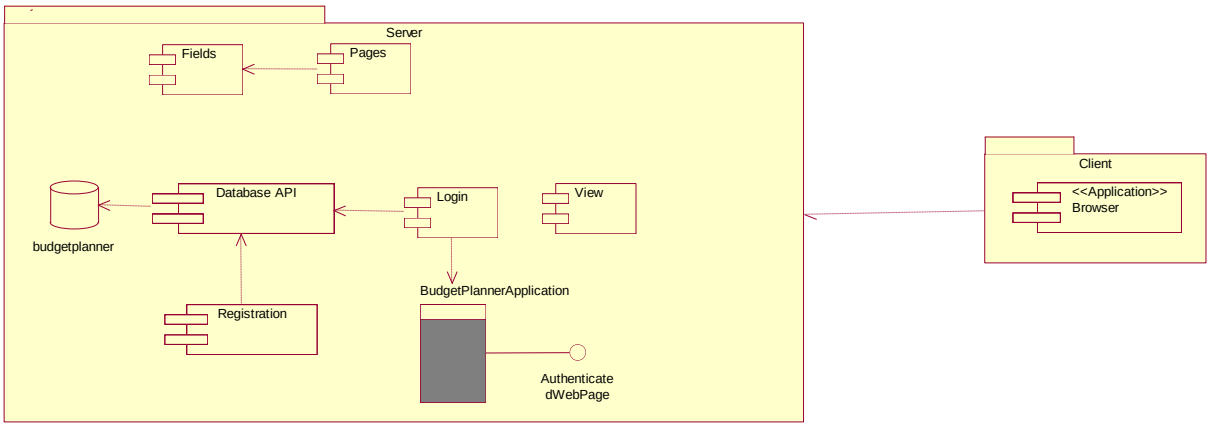


рис.15 Компонентная диаграмма

Диаграмма размещения

В конечном итоге, программа представляет собой единственный запускаемый файл, рассчитанный на работу в локальном режиме на стандартном однопроцессорном компьютере. Диаграмма размещения приведена на рисунке 16.

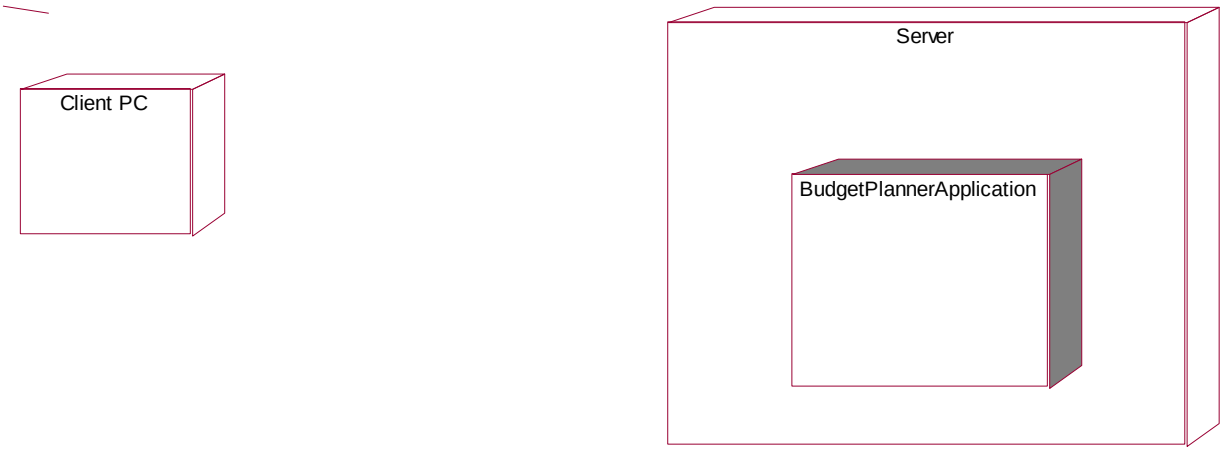


рис.15 Диаграмма размещения

Выводы

В ходе написания курсового проекта были применены на практике и более подробно изучены технологии визуального конструирования сложных программных систем. Был получен дополнительный опыт разработки визуальной модели системы, её оценивание качества с помощью набора метрик Чидамбера и Кемерера. Ознакомились с программной средой Rational Rose. Можно сделать вывод о том, что визуальное проектирование сложных ОО программ, на первых этапах разработки, существенно облегчает труд разработчика, предоставляет возможность зрительного восприятия и начального анализа будущего проекта.

В результате проделанной работы была получена объектно-ориентированная программная система с заданными характеристиками и отвечающая поставленным требованиям.

В ходе планирования итераций конструирования удалось найти решение, позволяющее получать после каждой завершенной итерации не только законченные с точки зрения программной реализации классы, конструирование которых планировалось на данной итерации, но и работоспособную (в рамках разрабатываемого Use-case'а) систему, что позволяет не только значительно уменьшить риск разработки посредством обеспечения возможности отладки кода на каждой итерации, но и снизить накладные расходы, связанные с достаточно трудоемким процессом пересчета метрик, в случае если разработка методов класса ведется в ходе нескольких итераций.

Приложение

Программный код

Класс BasePage

```
/*
* To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package lv.budgetplanner.app;

import lv.budgetplanner.login.SignInSession;
import lv.budgetplanner.error404.ErrorPage404;
import lv.budgetplanner.db.DataBase;
import java.util.List;
import lv.budgetplanner.pages.BudgetCreationPage;
import lv.budgetplanner.pages.EducationHealth;
import lv.budgetplanner.pages.EntertainmentEatingOut;
import lv.budgetplanner.pages.FinancialCommitments;
import lv.budgetplanner.pages.HomeUtilities;
import lv.budgetplanner.pages.Income;
import lv.budgetplanner.pages.Results;
import lv.budgetplanner.pages.ShoppingTransport;
import org.apache.wicket.Session;
import org.apache.wicket.markup.html.WebPage;
import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.markup.html.form.DropDownChoice;
import org.apache.wicket.markup.html.form.Form;
import org.apache.wicket.markup.html.link.BookmarkablePageLink;
import org.apache.wicket.markup.html.link.Link;
import org.apache.wicket.markup.html.panel.FeedbackPanel;
import org.apache.wicket.model.PropertyModel;

/**
 *
 * @author Maxim
 */
public class BasePage extends WebPage implements AuthenticatedWebPage {

private static List<String> BUDGET_PLANNER_NAMES =
DataBase.selectBudgetPlannersNames(null);
private String selected = BUDGET_PLANNER_NAMES.get(0);

public BasePage() {

add(new BookmarkablePageLink("income", Income.class));
add(new BookmarkablePageLink("financialCommitments", FinancialCommitments.class));
add(new BookmarkablePageLink("homeUtilities", HomeUtilities.class));
add(new BookmarkablePageLink("educationHealth", EducationHealth.class));
add(new BookmarkablePageLink("shoppingTransport", ShoppingTransport.class));
add(new BookmarkablePageLink("entertainmentEatingOut", EntertainmentEatingOut.class));
add(new BookmarkablePageLink("results", Results.class));

        Form<?> form = new Form("basePageForm");
```

```
form.add(new Label("incomeLabel", new PropertyModel<Integer>(Income.incomeFields,
"incomeTotal")));
form.add(new Label("financialCommLabel", new
PropertyModel<Integer>(FinancialCommitments.outgoingFields, "financialCommitmentsTotal")));
form.add(new Label("homeUtilitiesLabel", new
PropertyModel<Integer>(FinancialCommitments.outgoingFields, "homeUtilitiesTotal")));
form.add(new Label("educationHealthLabel", new
PropertyModel<Integer>(FinancialCommitments.outgoingFields, "educationHealthTotal")));
form.add(new Label("shoppingTransportLabel", new
PropertyModel<Integer>(FinancialCommitments.outgoingFields, "shoppingTransportTotal")));
form.add(new Label("entertainmentLabel", new
PropertyModel<Integer>(FinancialCommitments.outgoingFields, "entertainmentEatingOutTotal")));
form.add(new Label("whatsLeftLabel", new PropertyModel<Integer>(Results.resultFields, "result")));
form.add(new Label("per", "Year!"));

add(form);

if (((SignInSession) Session.get()).getUser().toString().isEmpty()) {
setResponsePage(ErrorPage404.class);
	} else {
	  BUDGET_PLANNER_NAMES = DataBase.selectBudgetPlannersNames(((SignInSession)
Session.get()).getUser().toString());
	}

	DropDownChoice<String> budgetPlannerNames = new
DropDownChoice<String>("budgetPlannersNamesSelector", new PropertyModel<String>(this,
"selected"), BUDGET_PLANNER_NAMES);
add(new Link("createBudgetButton") {
	  @Override
public void onClick() {
setResponsePage(BudgetCreationPage.class);
	  }
	});
add(new Link("saveBudgetButton") {
	  @Override
public void onClick() {
if (DataBase.saveBudget(selected) == true) {
success("Budget has been saved successfully!");
	    } else {
error("Please be sure you entered correct name of new budget.");
	    }
	  }
	});
add(new Link("deleteBudgetbutton") {
	  @Override
public void onClick() {
if (DataBase.deleteBudget(selected) == true) {
success("Budget has been deleted successfully!");
	    } else {
error("Please be sure you entered correct name of new budget.");
	    }
```

```
            }
        });
add(budgetPlannerNames);
add(new Label("whoiam", ((SignInSession) Session.get()).getUser().toString()));
add(new FeedbackPanel("feedback"));
add(new Label("footer", "All copyrights are reserved by © "));
    }
}
```

Класс DataBase

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package lv.budgetplanner.db;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import lv.budgetplanner.pages.FinancialCommitments;
import lv.budgetplanner.pages.Income;
import lv.budgetplanner.login.SignInSession;
import org.apache.wicket.Session;

/**
 *
 * @author Maxim
 */
public class DataBase {

public static boolean selectUsernameFromDbUserTable(String username) {
        String usernameFromDb = "";
boolean bool = false;
try {
Class.forName("com.mysql.jdbc.Driver");
            Connection con = DriverManager.getConnection("jdbc:mysql://localhost/budgetplanner",
"root", "admin");
            Statement st = (Statement) con.createStatement();
            String selectTableSQL = "SELECT * FROM `user`";
            ResultSet rs = st.executeQuery(selectTableSQL);
while (rs.next()) {
usernameFromDb = rs.getString("Username");
if (username.equals(usernameFromDb)) {
bool = true;
                }
            }
        } catch (Exception e) {
```

```
        }
return bool;
    }

public static boolean selectPasswordFromDbUserTable(String password) {
        String passwordFromDb = "";
boolean bool = false;
try {
Class.forName("com.mysql.jdbc.Driver");
            Connection con = DriverManager.getConnection("jdbc:mysql://localhost/budgetplanner",
"root", "admin");
            Statement st = (Statement) con.createStatement();
            String selectTableSQL = "SELECT * FROM `user`";
            ResultSet rs = st.executeQuery(selectTableSQL);
while (rs.next()) {
passwordFromDb = rs.getString("Password");
if (password.equals(passwordFromDb)) {
bool = true;
                }
            }
        } catch (Exception e) {
        }
return bool;
    }

public static void insertRecordIntoDbUserTable(String usernameValue, String passwordValue) {
try {
Class.forName("com.mysql.jdbc.Driver");
            Connection con = DriverManager.getConnection("jdbc:mysql://localhost/budgetplanner",
"root", "admin");
            Statement st = (Statement) con.createStatement();
            String insertTableSQL = "INSERT INTO budgetplanner.`user` (`Username`, `Password`)
VALUES ('" + usernameValue + "', '" + passwordValue + "');";
st.executeUpdate(insertTableSQL);
st.close();
        } catch (Exception e) {
        }
    }

public static boolean usernameIsNotAlreadyTaken(String usernameValue) {
        String usernameFromDb = "";
boolean bool = true;
try {
Class.forName("com.mysql.jdbc.Driver");
            Connection con = DriverManager.getConnection("jdbc:mysql://localhost/budgetplanner",
"root", "admin");
            Statement st = (Statement) con.createStatement();
            String selectTableSQL = "SELECT * FROM `user`";
            ResultSet rs = st.executeQuery(selectTableSQL);
while (rs.next()) {
usernameFromDb = rs.getString("Username");
if (usernameValue.equals(usernameFromDb)) {
```

```
bool = false;
                }
            }
        } catch (Exception e) {
        }
return bool;
    }

public static List<String> selectBudgetPlannersNames(String username) {
        ArrayList<String> budgetPlannersNamesList = new ArrayList<String>();
        String budgetPlannerName;
try {
Class.forName("com.mysql.jdbc.Driver");
            Connection con = DriverManager.getConnection("jdbc:mysql://localhost/budgetplanner",
"root", "admin");
            Statement st = (Statement) con.createStatement();
            String selectTableSQL = "SELECT `BudgetName` FROM budgetplace WHERE
`User_idUser` = '" + getUserId(username) + "';";
            ResultSet rs = st.executeQuery(selectTableSQL);
while (rs.next()) {
budgetPlannerName = rs.getString("BudgetName");
budgetPlannersNamesList.add(budgetPlannerName);
            }
        } catch (Exception e) {
        }
if (budgetPlannersNamesList.isEmpty()) {
budgetPlannersNamesList.add("You don't have saved budgets");
        }
return budgetPlannersNamesList;
    }
public static String budgetNameDB;
public static String userIdDB;
public static String budgetIdDB;
public static String usernameDB;

public static boolean saveBudget(String budgetName) {

        DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
        Date date = new Date();

boolean bool = true;
try {
Class.forName("com.mysql.jdbc.Driver");
            Connection con = DriverManager.getConnection("jdbc:mysql://localhost/budgetplanner",
"root", "admin");
            Statement st = (Statement) con.createStatement();
            String selectTableSQL = "SELECT * from budgetplace";
            ResultSet rs = st.executeQuery(selectTableSQL);
while (rs.next()) {
userIdDB = rs.getString("User_idUser");
budgetNameDB = rs.getString("BudgetName");
budgetIdDB = rs.getString("idBudgetPlace");
```

```
        }
if (budgetName.equals(budgetNameDB)) {
try {
            String insertIncomeTableSQL = "UPDATE budgetplanner.budgetincome SET
`BudgetIncomecol` = " + Income.incomeFields.youTakeHomePayInput + ", `BudgetIncomecol1` = " +
Income.incomeFields.yourPartnersTakeHomePayInput + ", `BudgetIncomecol2` = " +
Income.incomeFields.bonusesOvertimeInput + ", `BudgetIncomecol3` = " +
Income.incomeFields.incomeFromSavingsAndInvestmentsInput + ", `BudgetIncomecol4` = " +
Income.incomeFields.centrelingBenefitsInput + ", `BudgetIncomecol5` = " +
Income.incomeFields.familyBenefitsPaymentsInput + ", `BudgetIncomecol6` = " +
Income.incomeFields.childSupportReseivedInput + ", `BudgetIncomecol7` = " +
Income.incomeFields.otherInput + ", `BudgetIncomeResult` = " + Income.incomeFields.incomeTotal
+ " WHERE `BudgetPlace_idBudgetPlace` = '" + budgetIdDB + "' AND `BudgetPlace_User_idUser`
= '" + userIdDB + "';";
            String insertOutcomeTableSQL = "UPDATE budgetplanner.budgetoutcome SET
`BudgetOutcomecol` = " + FinancialCommitments.outgoingFields.rentMontageInput + ",
`BudgetOutcomecol1` = " + FinancialCommitments.outgoingFields.carLoanRepaymentsInput + ",
`BudgetOutcomecol2` = " + FinancialCommitments.outgoingFields.otherLoanRepaymentsInput + ",
`BudgetOutcomecol3` = " + FinancialCommitments.outgoingFields.creditCardInterestsInput + ",
`BudgetOutcomecol4` = " + FinancialCommitments.outgoingFields.voluntarySuperContributionsInput
+ ", `BudgetOutcomecol5` = " + FinancialCommitments.outgoingFields.savingsInput + ",
`BudgetOutcomecol6` = " + FinancialCommitments.outgoingFields.childSupportPaymentsInput + ",
`BudgetOutcomecol7` = " + FinancialCommitments.outgoingFields.donationsCharityInput + ",
`BudgetOutcomecol8` = " + FinancialCommitments.outgoingFields.pocketMoneyInput + ",
`BudgetOutcomecol9` = " + FinancialCommitments.outgoingFields.otherInput + ",
`BudgetOutcomeResult` = " + FinancialCommitments.outgoingFields.financialCommitmentsTotal + "
WHERE `BudgetPlace_idBudgetPlace` = '" + budgetIdDB + "' AND `BudgetPlace_User_idUser` = '"
+ userIdDB + "';";
st.executeUpdate(insertIncomeTableSQL);
st.executeUpdate(insertOutcomeTableSQL);
st.close();
            } catch (Exception e) {
            }
        } else {
try {
            String insertBudgetPlaceTableSQL = "INSERT INTO budgetplanner.budgetplace
(`BudgetName`, `BudgetDate`, `User_idUser`) VALUES ('" + budgetName + "','" +
dateFormat.format(date) + "','" + getUserId(((SignInSession) Session.get()).getUser().toString()) +
"');";
            String insertTableSQL = "INSERT INTO budgetplanner.budgetincome
(`BudgetIncomecol`, `BudgetIncomecol1`, `BudgetIncomecol2`, `BudgetIncomecol3`,
`BudgetIncomecol4`, `BudgetIncomecol5`, `BudgetIncomecol6`, `BudgetIncomecol7`,
`BudgetIncomeResult`, `BudgetPlace_idBudgetPlace`,`BudgetPlace_User_idUser`) VALUES ('" +
Income.incomeFields.youTakeHomePayInput + "', '" +
Income.incomeFields.yourPartnersTakeHomePayInput + "', '" +
Income.incomeFields.bonusesOvertimeInput + "', '" +
Income.incomeFields.incomeFromSavingsAndInvestmentsInput + "', '" +
Income.incomeFields.centrelingBenefitsInput + "', '" +
Income.incomeFields.familyBenefitsPaymentsInput + "', '" +
Income.incomeFields.childSupportReseivedInput + "', '" + Income.incomeFields.otherInput + "', '" +
getBudgetId(budgetName) + "','" + getUserId(((SignInSession) Session.get()).getUser().toString()) +
"');";
```

```
st.executeUpdate(insertBudgetPlaceTableSQL);
st.executeUpdate(insertTableSQL);
st.close();
            } catch (Exception e) {
            }
        }
      } catch (Exception e) {
bool = false;
      }
return bool;
   }

public static void getPageReadyBeforeResponse() {
try {
Class.forName("com.mysql.jdbc.Driver");
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost/budgetplanner",
"root", "admin");
        Statement st = (Statement) con.createStatement();
        String selectTableSQL = "SELECT * FROM budgetincome WHERE
`BudgetPlace_User_idUser`='" + getUserId(((SignInSession) Session.get()).getUser().toString()) +
"';";
        ResultSet rs = st.executeQuery(selectTableSQL);
while (rs.next()) {
            Income.incomeFields.youTakeHomePayInput = rs.getInt("BudgetIncomecol");
            Income.incomeFields.incomeTotal = rs.getInt("BudgetIncomeResult");
        }
      } catch (Exception e) {
      }
   }

public static boolean deleteBudget(String budgetName) {
boolean bool = true;
try {
Class.forName("com.mysql.jdbc.Driver");
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost/budgetplanner",
"root", "admin");
        Statement st = (Statement) con.createStatement();
        String selectTableSQL = "SELECT * from budgetplace";
        ResultSet rs = st.executeQuery(selectTableSQL);
while (rs.next()) {
usernameDB = rs.getString("Username");
budgetNameDB = rs.getString("Budget name");
        }
if (budgetName.equals(budgetNameDB) && usernameDB.equals(((SignInSession)
Session.get()).getUser().toString())) {
try {
            String insertTableSQL = "DELETE FROM budgetplanner.budgetplace WHERE
`User_idUser` = '" + getUserId(((SignInSession) Session.get()).getUser().toString()) + "' AND
`BudgetName` = '" + budgetName + "'";
st.executeUpdate(insertTableSQL);
st.close();
            } catch (Exception e) {
```

```
            }
        } else {
bool = false;
        }
    } catch (Exception e) {
bool = false;
    }
return bool;
  }

private static String getUserId(String username) {
    String userId = "";
try {
Class.forName("com.mysql.jdbc.Driver");
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost/budgetplanner",
"root", "admin");
        Statement st = (Statement) con.createStatement();
        String selectTableSQL = "SELECT idUser FROM `user` WHERE Username = '" + username +
"';";
        ResultSet rs = st.executeQuery(selectTableSQL);
while (rs.next()) {
userId = rs.getString("idUser");
        }
return userId;
    } catch (Exception e) {

    }
return "";
  }

private static String getBudgetId(String budgetName) {
    String budgetId = "";
try {
Class.forName("com.mysql.jdbc.Driver");
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost/budgetplanner",
"root", "admin");
        Statement st = (Statement) con.createStatement();
        String selectTableSQL = "SELECT idBudgetPlace FROM `budgetplace` WHERE
BudgetName = '" + budgetName + "';";
        ResultSet rs = st.executeQuery(selectTableSQL);
while (rs.next()) {
budgetId = rs.getString("idBudgetPlace");
        }
return budgetId;
    } catch (Exception e) {

    }
return "";
  }
}
```

Класс SignIn

```
/*
```

```
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package lv.budgetplanner.login;

import lv.budgetplanner.registration.RegistrationPage;
import lv.budgetplanner.db.DataBase;
import org.apache.wicket.markup.html.WebPage;
import org.apache.wicket.markup.html.form.Form;
import org.apache.wicket.markup.html.form.PasswordTextField;
import org.apache.wicket.markup.html.form.TextField;
import org.apache.wicket.markup.html.link.Link;
import org.apache.wicket.markup.html.panel.FeedbackPanel;
import org.apache.wicket.model.PropertyModel;
import org.apache.wicket.util.value.ValueMap;

/**
 *
 * @author Maxim
 */
public final class SignIn extends WebPage {

public SignIn() {
add(new SignInForm("signInForm"));
    }

public final class SignInForm extends Form<Void> {

private static final String USERNAME = "username";
private static final String PASSWORD = "password";
private final ValueMap properties = new ValueMap();

public SignInForm(final String id) {
super(id);
add(new FeedbackPanel("feedback"));
add(new Link("registrationPage") {
            @Override
public void onClick() {
setResponsePage(RegistrationPage.class);
            }
        });
add(new TextField<String>(USERNAME, new PropertyModel<String>(properties, USERNAME)));
add(new PasswordTextField(PASSWORD, new PropertyModel<String>(properties, PASSWORD)));
    }

    @Override
public final void onSubmit() {
        SignInSession session = getMySession();

if (session.signIn(getUsername(), getPassword())) {
DataBase.getPageReadyBeforeResponse();
setResponsePage(getApplication().getHomePage());
```

```
            } else {
                String errmsg = getString("loginError", null, "Incorrect username or password.");

error(errmsg);
            }
        }

private String getPassword() {
return properties.getString(PASSWORD);
        }

private String getUsername() {
return properties.getString(USERNAME);
        }

private SignInSession getMySession() {
return (SignInSession) getSession();
        }
    }
}
```

Клacс BudgetCreationPage

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package lv.budgetplanner.pages;

import lv.budgetplanner.app.BasePage;
import lv.budgetplanner.db.DataBase;
import org.apache.wicket.markup.html.WebPage;
import org.apache.wicket.markup.html.form.Form;
import org.apache.wicket.markup.html.form.TextField;
import org.apache.wicket.markup.html.link.Link;
import org.apache.wicket.markup.html.panel.FeedbackPanel;
import org.apache.wicket.model.PropertyModel;

/**
 *
 * @author Maxim
 */
public final class BudgetCreationPage extends WebPage {

private String budgetName;

public BudgetCreationPage() {
        Form<?> form = new Form("budgetCreationForm") {
            @Override
protected void onSubmit() {
if (DataBase.saveBudget(budgetName) == true) {
setResponsePage(BasePage.class);
                }
            }
```

```
        };
form.add(new FeedbackPanel("feedback"));
form.add(new Link("homePage") {
            @Override
public void onClick() {
setResponsePage(BasePage.class);
            }
        });
form.add(new TextField<String>("budgetName", new PropertyModel<String>(this, "budgetName")));
add(form);
    }
}
```

Класс FinancialCommitments

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package lv.budgetplanner.pages;

import lv.budgetplanner.app.BasePage;
import lv.budgetplanner.app.MyDropDownChoice;
import lv.budgetplanner.fields.OutgoingFields;
import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.markup.html.form.Form;
import org.apache.wicket.markup.html.form.TextField;
import org.apache.wicket.markup.html.link.Link;
import org.apache.wicket.model.PropertyModel;

/**
 *
 * @author Maxim
 */
public final class FinancialCommitments extends BasePage {

public static OutgoingFields outgoingFields = new OutgoingFields();

public FinancialCommitments() {
        Form<?> form = new Form("form") {
            @Override
protected void onSubmit() {
                outgoingFields.rentMontageLabel = outgoingFields.rentMontageInput *
outgoingFields.rentMontageFrequency;
                outgoingFields.carLoanRepaymentsLabel = outgoingFields.carLoanRepaymentsInput *
outgoingFields.carLoanRepaymentsFrequency;
                outgoingFields.otherLoanRepaymentsLabel = outgoingFields.otherLoanRepaymentsInput *
outgoingFields.otherLoanRepaymentsFrequency;
                outgoingFields.creditCardInterestsLabel = outgoingFields.creditCardInterestsInput *
outgoingFields.creditCardInterestsFrequency;
                outgoingFields.voluntarySuperContributionsLabel =
outgoingFields.voluntarySuperContributionsInput *
outgoingFields.voluntarySuperContributionsFrequency;
                outgoingFields.savingsLabel = outgoingFields.savingsInput *
```

```
outgoingFields.savingsFrequency;
            outgoingFields.childSupportPaymentsLabel = outgoingFields.childSupportPaymentsInput *
outgoingFields.childSupportPaymentsFrequency;
            outgoingFields.donationsCharityLabel = outgoingFields.donationsCharityInput *
outgoingFields.donationsCharityFrequency;
            outgoingFields.pocketMoneyLabel = outgoingFields.pocketMoneyInput *
outgoingFields.pocketMoneyFrequency;
            outgoingFields.otherLabel = outgoingFields.otherInput * outgoingFields.otherFrequency;
            outgoingFields.financialCommitmentsTotal = outgoingFields.rentMontageLabel
                + outgoingFields.carLoanRepaymentsLabel
                + outgoingFields.otherLoanRepaymentsLabel
                + outgoingFields.creditCardInterestsLabel
                + outgoingFields.voluntarySuperContributionsLabel
                + outgoingFields.savingsLabel
                + outgoingFields.childSupportPaymentsLabel
                + outgoingFields.donationsCharityLabel
                + outgoingFields.pocketMoneyLabel
                + outgoingFields.otherLabel;
            Results.resultFields.result = Income.incomeFields.incomeTotal -
FinancialCommitments.outgoingFields.financialCommitmentsTotal -
FinancialCommitments.outgoingFields.homeUtilitiesTotal -
FinancialCommitments.outgoingFields.educationHealthTotal -
FinancialCommitments.outgoingFields.shoppingTransportTotal -
FinancialCommitments.outgoingFields.entertainmentEatingOutTotal;
        }
    };

form.add(new Label("rentMontageLabel", new PropertyModel<Integer>(outgoingFields,
"rentMontageLabel")));
form.add(new TextField<Integer>("rentMontageInput", new PropertyModel<Integer>(outgoingFields,
"rentMontageInput")));

form.add(new Label("carLoanRepaymentsLabel", new PropertyModel<Integer>(outgoingFields,
"carLoanRepaymentsLabel")));
form.add(new TextField<Integer>("carLoanRepaymentsInput", new
PropertyModel<Integer>(outgoingFields, "carLoanRepaymentsInput")));

form.add(new Label("otherLoanRepaymentsLabel", new PropertyModel<Integer>(outgoingFields,
"otherLoanRepaymentsLabel")));
form.add(new TextField<Integer>("otherLoanRepaymentsInput", new
PropertyModel<Integer>(outgoingFields, "otherLoanRepaymentsInput")));

form.add(new Label("creditCardInterestsLabel", new PropertyModel<Integer>(outgoingFields,
"creditCardInterestsLabel")));
form.add(new TextField<Integer>("creditCardInterestsInput", new
PropertyModel<Integer>(outgoingFields, "creditCardInterestsInput")));

form.add(new Label("voluntarySuperContributionsLabel", new
PropertyModel<Integer>(outgoingFields, "voluntarySuperContributionsLabel")));
form.add(new TextField<Integer>("voluntarySuperContributionsInput", new
PropertyModel<Integer>(outgoingFields, "voluntarySuperContributionsInput")));
```

```
form.add(new Label("savingsLabel", new PropertyModel<Integer>(outgoingFields, "savingsLabel")));
form.add(new TextField<Integer>("savingsInput", new PropertyModel<Integer>(outgoingFields,
"savingsInput")));

form.add(new Label("childSupportPaymentsLabel", new PropertyModel<Integer>(outgoingFields,
"childSupportPaymentsLabel")));
form.add(new TextField<Integer>("childSupportPaymentsInput", new
PropertyModel<Integer>(outgoingFields, "childSupportPaymentsInput")));

form.add(new Label("donationsCharityLabel", new PropertyModel<Integer>(outgoingFields,
"donationsCharityLabel")));
form.add(new TextField<Integer>("donationsCharityInput", new
PropertyModel<Integer>(outgoingFields, "donationsCharityInput")));

form.add(new Label("pocketMoneyLabel", new PropertyModel<Integer>(outgoingFields,
"pocketMoneyLabel")));
form.add(new TextField<Integer>("pocketMoneyInput", new
PropertyModel<Integer>(outgoingFields, "pocketMoneyInput")));

form.add(new Label("otherLabel", new PropertyModel<Integer>(outgoingFields, "otherLabel")));
form.add(new TextField<Integer>("otherInput", new PropertyModel<Integer>(outgoingFields,
"otherInput")));

add(form);

form.add(new MyDropDownChoice("rentMontageSelect", new
PropertyModel<Integer>(outgoingFields, "rentMontageFrequency")));

form.add(new MyDropDownChoice("carLoanRepaymentsSelect", new
PropertyModel<Integer>(outgoingFields, "carLoanRepaymentsFrequency")));

form.add(new MyDropDownChoice("otherLoanRepaymentsSelect", new
PropertyModel<Integer>(outgoingFields, "otherLoanRepaymentsFrequency")));

form.add(new MyDropDownChoice("creditCardInterestsSelect", new
PropertyModel<Integer>(outgoingFields, "creditCardInterestsFrequency")));

form.add(new MyDropDownChoice("voluntarySuperContributionsSelect", new
PropertyModel<Integer>(outgoingFields, "voluntarySuperContributionsFrequency")));

form.add(new MyDropDownChoice("savingsSelect", new PropertyModel<Integer>(outgoingFields,
"savingsFrequency")));

form.add(new MyDropDownChoice("childSupportPaymentsSelect", new
PropertyModel<Integer>(outgoingFields, "childSupportPaymentsFrequency")));

form.add(new MyDropDownChoice("donationsCharitySelect", new
PropertyModel<Integer>(outgoingFields, "donationsCharityFrequency")));

form.add(new MyDropDownChoice("pocketMoneySelect", new
PropertyModel<Integer>(outgoingFields, "pocketMoneyFrequency")));
```

```
form.add(new MyDropDownChoice("otherSelect", new PropertyModel<Integer>(outgoingFields,
"otherFrequency")));

form.add(new Link("previousPage") {
        @Override
public void onClick() {
setResponsePage(Income.class);
        }
      });
form.add(new Link("nextPage") {
        @Override
public void onClick() {
setResponsePage(HomeUtilities.class);
        }
      });
   }
}
```

Класс Income

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package lv.budgetplanner.pages;

import lv.budgetplanner.app.BasePage;
import lv.budgetplanner.fields.IncomeFields;
import lv.budgetplanner.app.MyDropDownChoice;
import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.markup.html.form.Form;
import org.apache.wicket.markup.html.form.TextField;
import org.apache.wicket.markup.html.link.Link;
import org.apache.wicket.model.PropertyModel;

/**
 *
 * @author Maxim
 */
public final class Income extends BasePage {

public static IncomeFields incomeFields = new IncomeFields();

public Income() {
      Form<?> form = new Form("form") {
        @Override
protected void onSubmit() {
            incomeFields.youTakeHomePayLabel = incomeFields.youTakeHomePayInput *
incomeFields.youTakeHomePayFrequency;
            incomeFields.yourPartnersTakeHomePayLabel =
incomeFields.yourPartnersTakeHomePayInput * incomeFields.yourPartnersTakeHomePayFrequency;
            incomeFields.bonusesOvertimeLabel = incomeFields.bonusesOvertimeInput *
incomeFields.bonusesOvertimeFrequency;
            incomeFields.incomeFromSavingsAndInvestmentsLabel =
```

```
incomeFields.incomeFromSavingsAndInvestmentsInput *
incomeFields.incomeFromSavingsAndInvestmentsFrequency;
            incomeFields.centrelingBenefitsLabel = incomeFields.centrelingBenefitsInput *
incomeFields.centrelingBenefitsFrequency;
            incomeFields.familyBenefitsPaymentsLabel = incomeFields.familyBenefitsPaymentsInput *
incomeFields.familyBenefitsPaymentsFrequency;
            incomeFields.childSupportReseivedLabel = incomeFields.childSupportReseivedInput *
incomeFields.childSupportReseivedFrequency;
            incomeFields.otherLabel = incomeFields.otherInput * incomeFields.otherFrequency;
            incomeFields.incomeTotal = incomeFields.youTakeHomePayLabel
                  + incomeFields.yourPartnersTakeHomePayLabel
                  + incomeFields.bonusesOvertimeLabel
                  + incomeFields.incomeFromSavingsAndInvestmentsLabel
                  + incomeFields.centrelingBenefitsLabel
                  + incomeFields.familyBenefitsPaymentsLabel
                  + incomeFields.childSupportReseivedLabel
                  + incomeFields.otherLabel;
            Results.resultFields.result = Income.incomeFields.incomeTotal -
FinancialCommitments.outgoingFields.financialCommitmentsTotal -
FinancialCommitments.outgoingFields.homeUtilitiesTotal -
FinancialCommitments.outgoingFields.educationHealthTotal -
FinancialCommitments.outgoingFields.shoppingTransportTotal -
FinancialCommitments.outgoingFields.entertainmentEatingOutTotal;
          }
      };
form.add(new Label("youTakeHomePayLabel", new PropertyModel<Integer>(incomeFields,
"youTakeHomePayLabel")));
form.add(new TextField<Integer>("youTakeHomePayInput", new
PropertyModel<Integer>(incomeFields, "youTakeHomePayInput")));

form.add(new Label("yourPartnersTakeHomePayLabel", new PropertyModel<Integer>(incomeFields,
"yourPartnersTakeHomePayLabel")));
form.add(new TextField<Integer>("yourPartnersTakeHomePayInput", new
PropertyModel<Integer>(incomeFields, "yourPartnersTakeHomePayInput")));

form.add(new Label("bonusesOvertimeLabel", new PropertyModel<Integer>(incomeFields,
"bonusesOvertimeLabel")));
form.add(new TextField<Integer>("bonusesOvertimeInput", new
PropertyModel<Integer>(incomeFields, "bonusesOvertimeInput")));

form.add(new Label("incomeFromSavingsAndInvestmentsLabel", new
PropertyModel<Integer>(incomeFields, "incomeFromSavingsAndInvestmentsLabel")));
form.add(new TextField<Integer>("incomeFromSavingsAndInvestmentsInput", new
PropertyModel<Integer>(incomeFields, "incomeFromSavingsAndInvestmentsInput")));

form.add(new Label("centrelingBenefitsLabel", new PropertyModel<Integer>(incomeFields,
"centrelingBenefitsLabel")));
form.add(new TextField<Integer>("centrelingBenefitsInput", new
PropertyModel<Integer>(incomeFields, "centrelingBenefitsInput")));

form.add(new Label("familyBenefitsPaymentsLabel", new PropertyModel<Integer>(incomeFields,
"familyBenefitsPaymentsLabel")));
```

```
form.add(new TextField<Integer>("familyBenefitsPaymentsInput", new
PropertyModel<Integer>(incomeFields, "familyBenefitsPaymentsInput")));

form.add(new Label("childSupportReseivedLabel", new PropertyModel<Integer>(incomeFields,
"childSupportReseivedLabel")));
form.add(new TextField<Integer>("childSupportReseivedInput", new
PropertyModel<Integer>(incomeFields, "childSupportReseivedInput")));

form.add(new Label("otherLabel", new PropertyModel<Integer>(incomeFields, "otherLabel")));
form.add(new TextField<Integer>("otherInput", new PropertyModel<Integer>(incomeFields,
"otherInput")));

add(form);

form.add(new MyDropDownChoice("youTakeHomePaySelect", new
PropertyModel<Integer>(incomeFields, "youTakeHomePayFrequency")));

form.add(new MyDropDownChoice("yourPartnersTakeHomePaySelect", new
PropertyModel<Integer>(incomeFields, "yourPartnersTakeHomePayFrequency")));

form.add(new MyDropDownChoice("bonusesOvertimeSelect", new
PropertyModel<Integer>(incomeFields, "bonusesOvertimeFrequency")));

form.add(new MyDropDownChoice("incomeFromSavingsAndInvestmentsSelect", new
PropertyModel<Integer>(incomeFields, "incomeFromSavingsAndInvestmentsFrequency")));

form.add(new MyDropDownChoice("centrelingBenefitsSelect", new
PropertyModel<Integer>(incomeFields, "centrelingBenefitsFrequency")));

form.add(new MyDropDownChoice("familyBenefitsPaymentsSelect", new
PropertyModel<Integer>(incomeFields, "familyBenefitsPaymentsFrequency")));

form.add(new MyDropDownChoice("childSupportReseivedSelect", new
PropertyModel<Integer>(incomeFields, "childSupportReseivedFrequency")));

form.add(new MyDropDownChoice("otherSelect", new PropertyModel<Integer>(incomeFields,
"otherFrequency")));

form.add(new Link("nextPage") {
        @Override
public void onClick() {
setResponsePage(FinancialCommitments.class);
        }
    });
  }
}
```

Класс Results

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package lv.budgetplanner.pages;
```

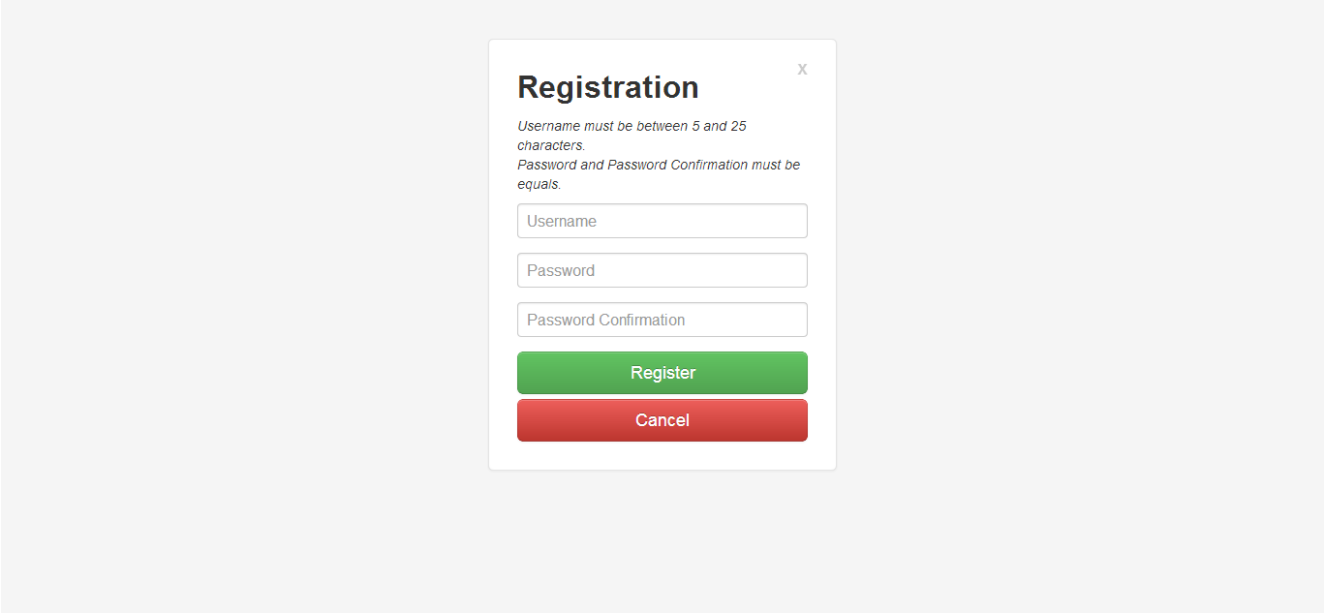
```
import lv.budgetplanner.app.BasePage;
import lv.budgetplanner.fields.ResultFields;
import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.model.PropertyModel;

/**
 *
 * @author Maxim
 */
public final class Results extends BasePage {

public static ResultFields resultFields = new ResultFields();

public Results() {
if (resultFields.result == 0) {
add(new Label("resultTotal", new PropertyModel<Integer>(resultFields, "result")));
add(new Label("lessOrMore", "not more and not less"));
add(new Label("summary", "Congradulations!"));
       } else {
if (resultFields.result > 0) {
add(new Label("resultTotal", new PropertyModel<Integer>(resultFields, "result")));
add(new Label("lessOrMore", "less"));
add(new Label("summary", "Congratulations!"));
           } else {
add(new Label("resultTotal", new PropertyModel<Integer>(resultFields, "result")));
add(new Label("lessOrMore", "more"));
add(new Label("summary", "We have bad news for you! :("));
           }
       }
   }
}
```

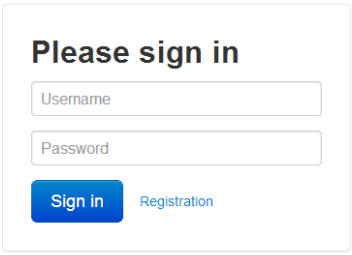
Класс RegistrationPage

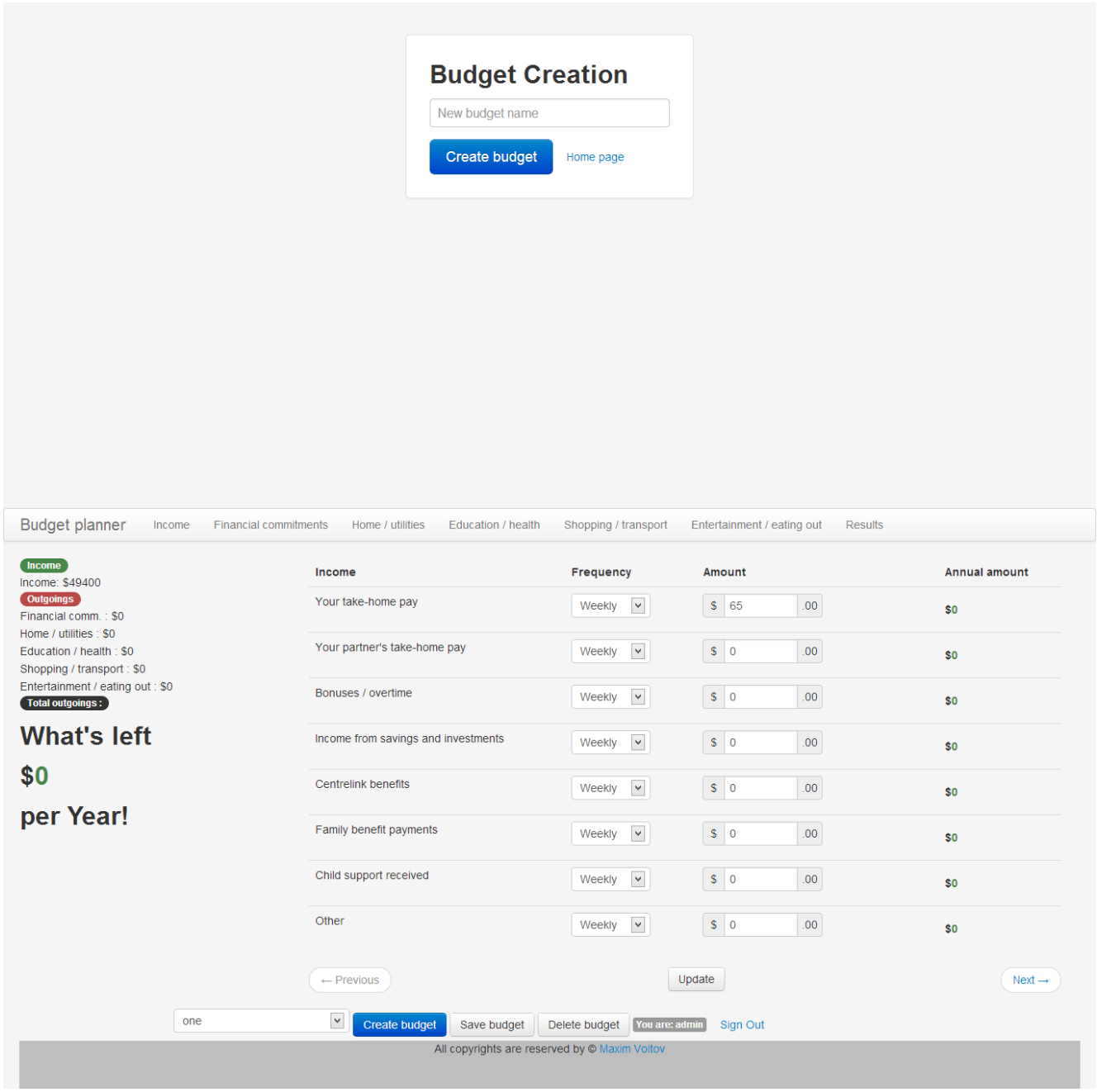
```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package lv.budgetplanner.registration;

import lv.budgetplanner.validators.PasswordValidator;
import lv.budgetplanner.validators.UsernameValidator;
import lv.budgetplanner.db.DataBase;
import lv.budgetplanner.login.SignIn;
import org.apache.wicket.markup.html.WebPage;
import org.apache.wicket.markup.html.form.Form;
import org.apache.wicket.markup.html.form.PasswordTextField;
import org.apache.wicket.markup.html.form.TextField;
import org.apache.wicket.markup.html.form.validation.EqualPasswordInputValidator;
import org.apache.wicket.markup.html.link.Link;
import org.apache.wicket.markup.html.panel.FeedbackPanel;
import org.apache.wicket.model.Model;
import org.apache.wicket.request.mapper.parameter.PageParameters;
```

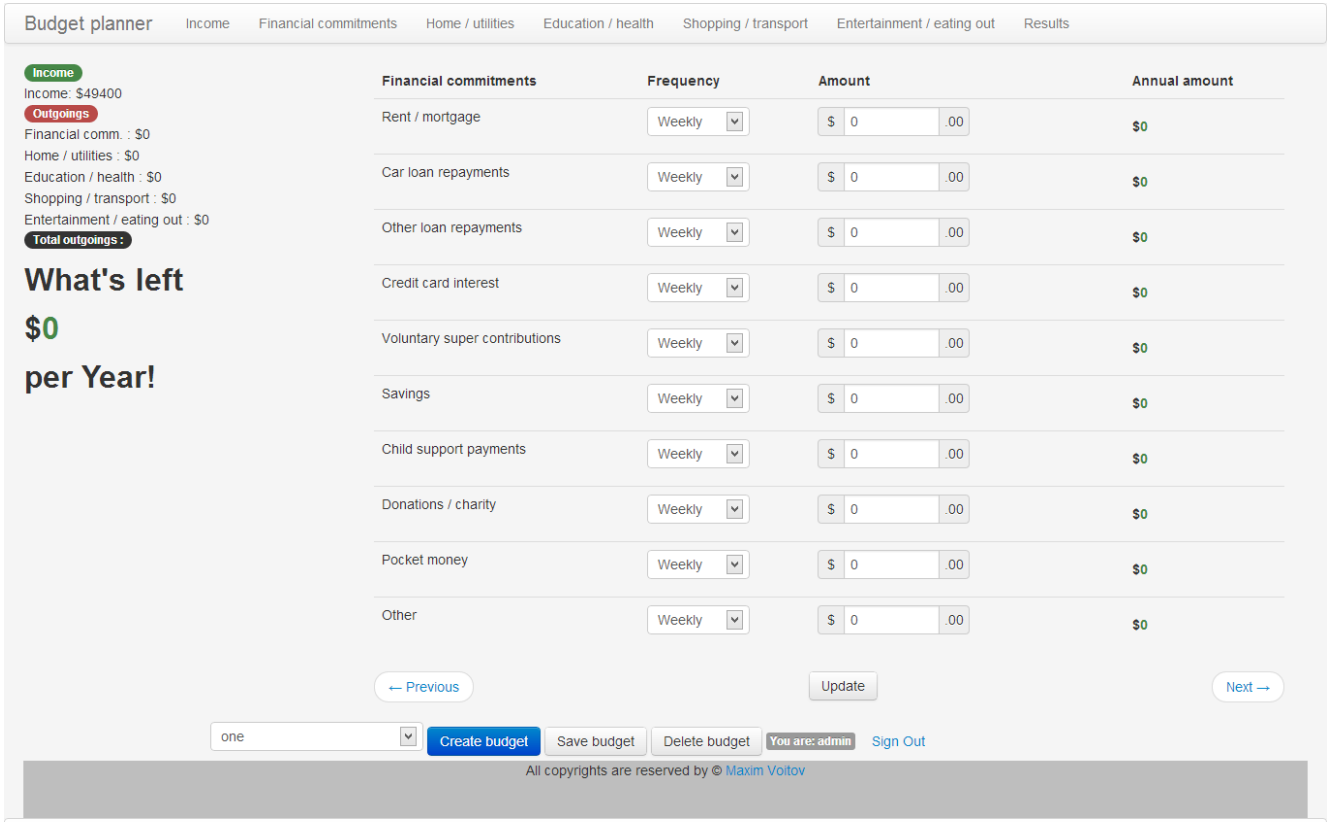
```
/**
 *
 * @author Maxim
 */
public final class RegistrationPage extends WebPage {

public RegistrationPage(final PageParameters parameters) {



final TextField<String> username = new TextField<String>("username", Model.of(""));
final PasswordTextField password = new PasswordTextField("password", Model.of(""));
final PasswordTextField cpassword = new PasswordTextField("cpassword", Model.of(""));
username.setRequired(true);
password.setRequired(true);
username.add(new UsernameValidator());
password.add(new PasswordValidator());
cpassword.add(new PasswordValidator());
       Form<?> form = new Form<Void>("registrationForm") {
          @Override
protected void onSubmit() {
final String usernameValue = username.getModelObject();
final String passwordValue = password.getModelObject();

             PageParameters pp = new PageParameters();
pp.add("username", usernameValue);
pp.add("password", passwordValue);
if (DataBase.usernameIsNotAlreadyTaken(usernameValue)) {
DataBase.insertRecordIntoDbUserTable(usernameValue, passwordValue);
setResponsePage(SuccessPage.class, pp);
             } else {
                String erromsg = "Username you entered is already used. Plase choose another one.";
error(erromsg);
             }

          }
       };
add(form);
form.add(new FeedbackPanel("feedback"));
form.add(new Link("loginPage") {
             @Override
public void onClick() {
setResponsePage(SignIn.class);
             }
          });
form.add(username);
form.add(password);
form.add(cpassword);
form.add(new EqualPasswordInputValidator(password, cpassword));
   }
}
```

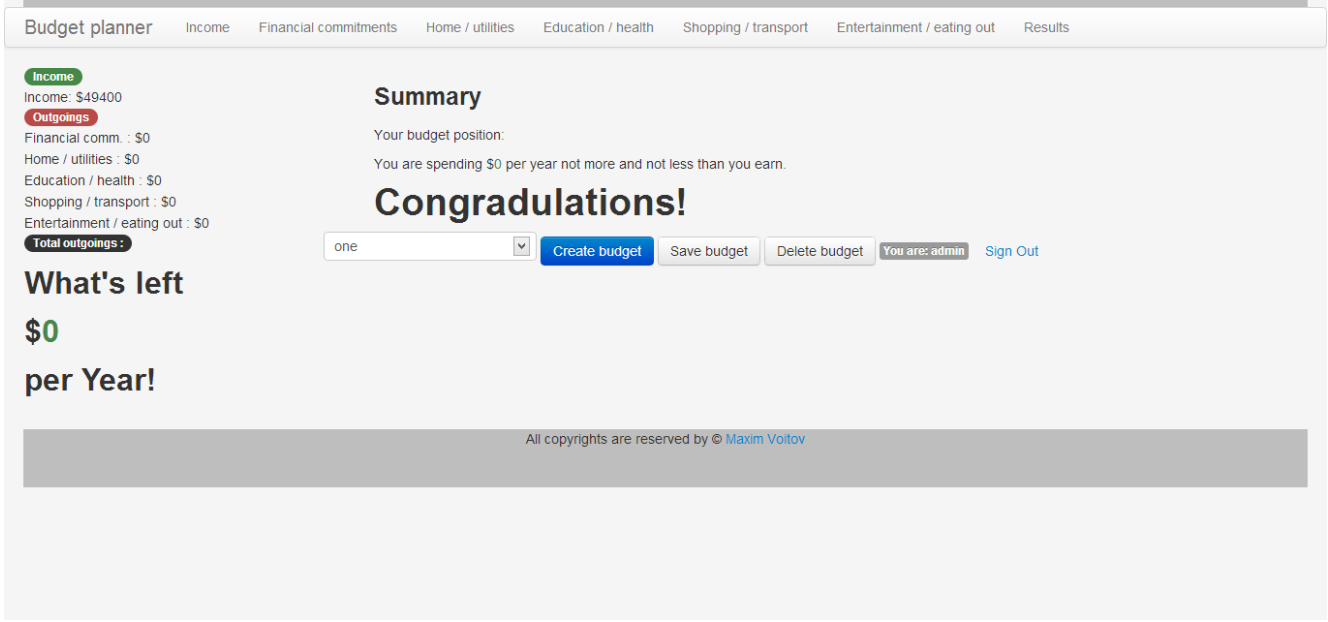
Интерфейс


Registration
x
Username must be between 5 and 25 characters.
Password and Password Confirmation must be equals.
Username
Password
Password Confirmation
Register
Cancel


Please sign in
Username
Password
Sign in
Registration


Budget Creation
New budget name
Create budget
Home page
Budget planner
Income
Financial commitments
Home / utilities
Education / health
Shopping / transport
Entertainment / eating out
Results
Income
Income: $49400
Outgoings
Financial comm. : $0
Home / utilities : $0
Education / health : $0
Shopping / transport : $0
Entertainment / eating out : $0
Total outgoings :
What's left
$0
per Year!
Income
Frequency
Amount
Annual amount
Your take-home pay
Weekly
$ 65 .00
$0
Your partner's take-home pay
Weekly
$ 0 .00
$0
Bonuses / overtime
Weekly
$ 0 .00
$0
Income from savings and investments
Weekly
$ 0 .00
$0
Centrelink benefits
Weekly
$ 0 .00
$0
Family benefit payments
Weekly
$ 0 .00
$0
Child support received
Weekly
$ 0 .00
$0
Other
Weekly
$ 0 .00
$0
← Previous
Update
Next →
one
Create budget
Save budget
Delete budget
You are: admin
Sign Out
All copyrights are reserved by © Maxim Voitov


Budget planner
Income
Financial commitments
Home / utilities
Education / health
Shopping / transport
Entertainment / eating out
Results
Income
Income: $49400
Outgoings
Financial comm. : $0
Home / utilities : $0
Education / health : $0
Shopping / transport : $0
Entertainment / eating out : $0
Total outgoings :
What's left
$0
per Year!
Financial commitments
Frequency
Amount
Annual amount
Rent / mortgage
Car loan repayments
Other loan repayments
Credit card interest
Voluntary super contributions
Savings
Child support payments
Donations / charity
Pocket money
Other
Weekly
$ 0 .00
$0
← Previous
Update
Next →
one
Create budget
Save budget
Delete budget
You are: admin
Sign Out
All copyrights are reserved by © Maxim Voitov


Budget planner
Income
Financial commitments
Home / utilities
Education / health
Shopping / transport
Entertainment / eating out
Results
Income
Income: $49400
Outgoings
Financial comm. : $0
Home / utilities : $0
Education / health : $0
Shopping / transport : $0
Entertainment / eating out : $0
Total outgoings :
What's left
$0
per Year!
Summary
Your budget position:
You are spending $0 per year not more and not less than you earn.
Congradulations!
one
Create budget
Save budget
Delete budget
You are: admin
Sign Out
All copyrights are reserved by © Maxim Voitov

Список используемой литературы

1. Орлов С.А., Современные технологии конструирования сложных программных систем. – Рига: ИТС, 1999.-289 с.:ил.
2. Орлов С.А., Цилькер Б.Я., Технология разработки программного обеспечения – Питер, 2012. – 608.: ил.
3. Орлов С.А., Введение в визуальное моделирование. – Рига: ИТС, 2001. – 55с.: ил.
4. Орлов С.А. Конспект лекций по технологиям конструирования программных систем.
5. Дарахвелидзе П.Г., Марков Е.П., Программирование в Delphi 4. – СПб.: БХВ – Санкт-Петербург, 1999.- 864с., ил.