# Институт Транспорта и Связи

Курсовой проект

по дисциплине “Технологии конструирования ПО”

Тема: “Система регистрации клиентов турфирмы”

Выполнили: Смирнов Валентин (4001BD)

Фёдоров Денисс (4001BV)

Проверил(-а): Тарасенко Карина

Рига

2013

## Цель работы

Целью данной курсовой работы является разработка программного обеспечения для заказа путёвок 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

Для каждого клиента турфирмы оператором создаётся заявка по заполненому шаблону, который клиент заполняет на месте. Так-же клиенту не обязательно приезжать в агенство, он может сам зарегестрироваться в системе через интернет и заказать путёвку самостоятельно.

После того как заявка была зарегестрирована, клиент должен её оплатить. Если клиент регистрировался в агенстве турфирмы, то оплату производит на месте наличными, либо карточкой, если же клиент регистрировался самостоятельно, то оплату производит через интернет банк. Как клиент оплатил заявку, так сразу она становится действительной.

Для всех пользователей система даёт возможность заказа путёвки, просмотр своей истории, просмотр “горячих” путёвок, изменение личных данных и пароля. А для операторов система даёт возможность удалять заказанные путёвки любого пользователя и заказывать путёвку клиенту.

Также в системе есть возможность просмотреть полезную справочную информацию, например: посещённые туры клиентом, информацию о клиенте, список возможных стран для путишествия, список возможных гостиниц и экскурсий в той или иной стране.

# Этап «Начало»

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

Программный продукт представляет собой систему заказов путёвок. Данная система длолжна следующие функции:

1. Заказ туров через интернет.
2. Заказ туров через оператора.

**Идентификация актёров и вариантов использования:**

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

**Описание актёров:**

Актёр **Клиент** - описывает пользователя работающего с системой заказов путёвок, производящий регистрацию в системе, авторизацию, заказ путёвки, изменение личных данных и пароля, просмотра “горячих” путёвок.

Актёр **Оператор** - описывает пользователя работающего с системой заказов путёвок, производящего все теже действия что и актёр клиент, но также имеет право производить удаление заказа для любого пользователя и заказ путёвки от лица актёра клиента.

Для актеров Клиент, Оператор определим соответствующие USE-CASE’ы, путем рассмотрения актеров и их взаимодействии с системой. На рисунке 1 представлена Use Case диаграмма построенная на основании требований заказчика.

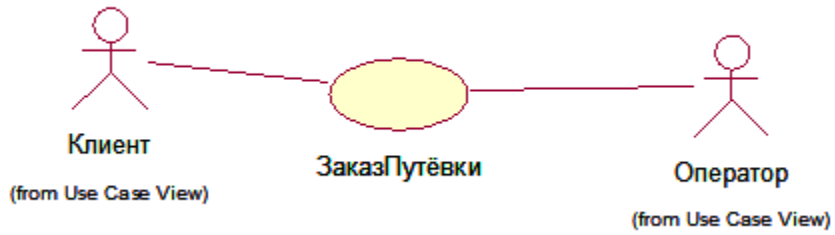



Рис.1 Начальная  Use Case диаграмма (20% от полного представления)

**Описание Use Case’ов:**

- ЗаказПутёвки - USE-CASE элемент предоставляет собой функциональную возможность системы заказа путёвок.

## 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:

- Добавление пользователя в систему.
- Проверка существования пользователя в системе.
- Изменение информации о пользователе.
- Заказ путёвки.
- Просмотр доступных путёвок.
- Удаление заказаной путёвки.
- Просмотр истории заказых путёвок пользователем.

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

**Описание Use Case'ов:**

- **Регистрирование** - функция добавления пользователя в систему.
- **Авторизация** - функция проверки на существования пользователя в системе.
- **ЗаказПутёвки** - функция заказа путёвки пользователем.
- **ОплатаЗаказа** - функция выбора вида оплаты и оплаты заказа
- **ИзменениеПерсональныхДанных** - функция изменения персональных данных.
- **ПросмотрИсторииПутёвок** - функция просмотра истории выполненых заказов пользователя.
- **ОформлениеЗаказаКлиента** - функция заказа путёвки оператором от лица пользователя.
- **ОтменаПутвёки** - функция удаления заказа путёвки оператором.

В итоге на этапе развитие конечная Use Case диаграммы имеют вид (рис.2, рис.3):

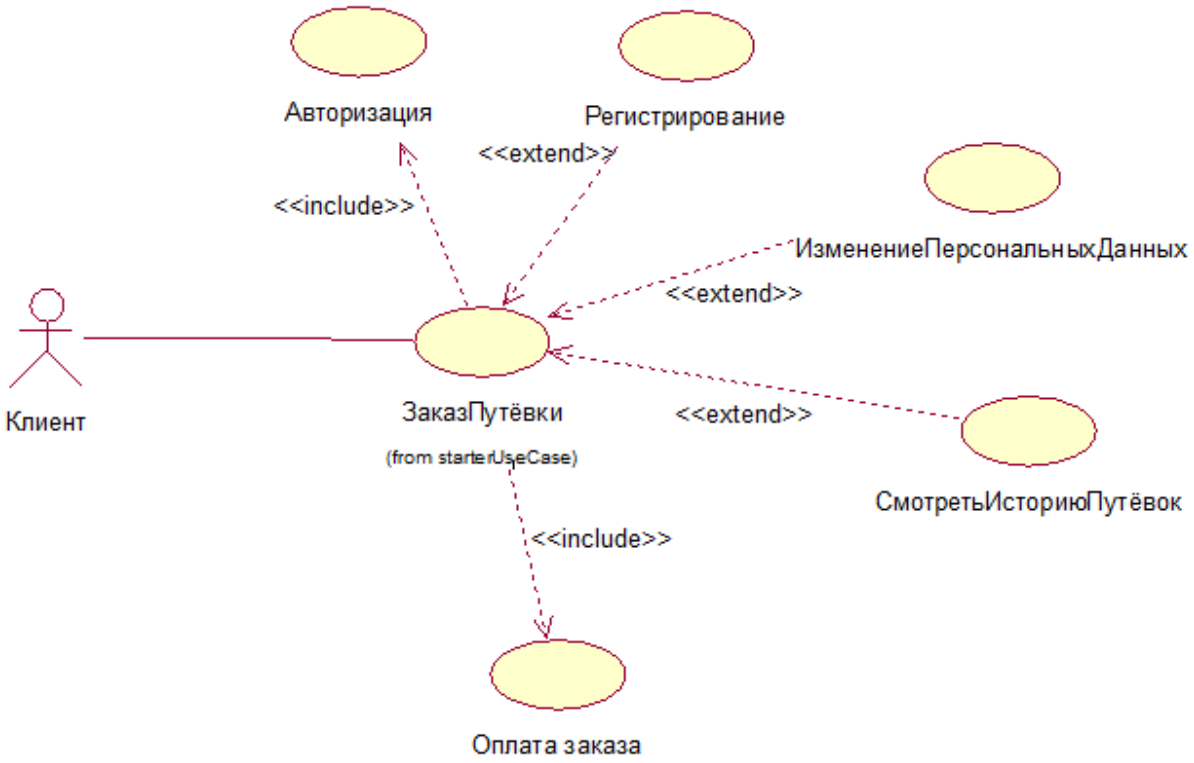



рис.2 Конечная Use Case диаграмма для актёра Клиента(80% от полного представления)

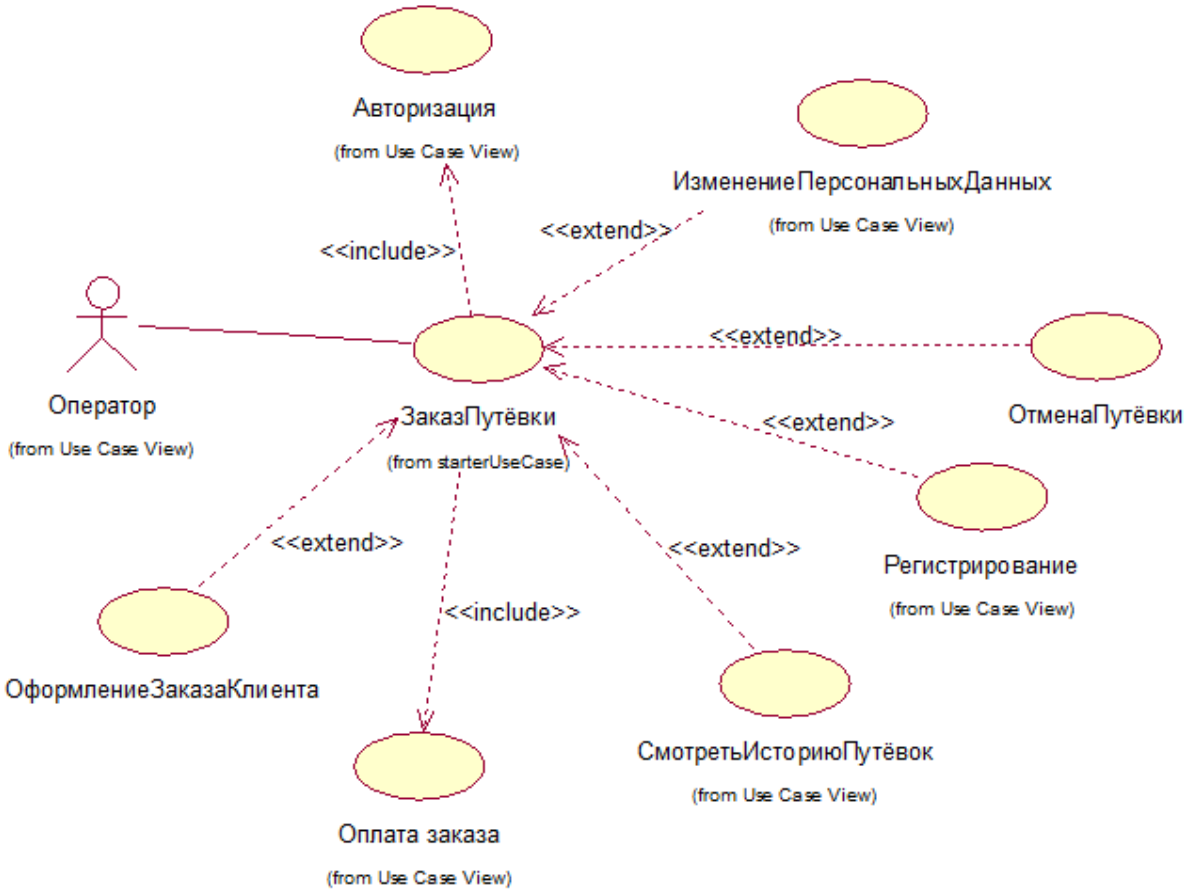



рис.3 Конечная Use Case диаграмма актёра Оператора(80% от полного представления)

**Определение сценариев:**

- Сценарий для Use-Case'а ОформлениеЗаказаКлиента:

Открытие страницы инициализирует запрос к менеджеру базы данных, затем выбираются все доступные заказы и заполняются пустые поля на странице. После выбора страны, отеля и экскурсии, а также ввода имени пользователя и нажатия кнопки "заказать", происходит запрос к менеджеру базы данных, где происходит выборка данных по имени пользователя, и если она прошла удачно, то происходит запрос к менеджеру базы данных, после которого происходит запись заказа, если же выборка данных по имени пользователя не удалась, то выполняется запрос к менеджеру базы данных и создаётся новый пользователь с таким именем и уже после этого происходит запрос к менеджеру базы данных, и после запроса происходит запись заказа нового пользователя.

- Сценарий для Use-Case'а ОплатаЗаказа:

Сценарий Use-Case’а ОплатаЗаказа становится доступным после нажатия кнопки “заказать” на страницах ОформлениеЗаказаКлиента и ЗаказКлиента, в случае нажатия кнопки “заказать” на странице ЗаказКлиента, в запросе к менеджеру базы данных добавляется значение, обозначающие что оплата была проведена карточкой. В случае нажатия кнопки “заказать” на странице ОформлениеЗаказаКлиента, в запросе к менеджеру базы данных добавляется значение которое было выбрано оператором, обозначающие что оплата была проведена карточкой, либо наличными.

- Сценарий для Use-Case’а СмотретьИсториюПутёвок:

Открытие страницы инициализирует запрос к менеджеру базы данных по имени пользователя, затем выбираются все совершонные заказы пользователя и после эгого на странице отображаются данные в виде таблицы.

- Сценарий для Use-Case’а Регистрирование:

При нажатии на кнопку “регистрация” открывается страница тем самым инициализирует процесс регистрации, после того как новый пользователь заполнит все поля регистрации и нажмет на кнопку «подтвердить» выполняется запрос к менеджеру базы данных и производится выборка данных из базы, затем сравниваются данные из базы с введёнными пользователем и если найдено совпадение, то выводится сообщение об ошибке, если совпадений не найдено то выполняется запрос к менеджеру базы данных и создаётся новый пользователь с введёнными данными.

- Сценарий для Use-Case’а ОтменаПутёвки:

Открытие страницы инициализирует запрос к менеджеру базы данных по имени пользователя, затем выбираются все совершонные заказы пользователя и после эгого, на странице отображаются данные в виде таблицы с кнопкой “удалить” напротив каждого заказа. Принажатии кнопки “удалить” напротив конкретного заказа, происходит запрос к менеджеру базы данных и после которого удаляется этот заказ. После удаления пользователь видит подтверждение удаления и происходит запрос к менеджеру базы данных, после которого выбираются все совершонные заказы пользователя и после эгого, на странице отображаются оставшиеся заказы в виде таблицы с кнопкой “удалить”.

- Сценарий для Use-Case’а ИзменениеПерсональныхДанных:

При нажатии на кнопку “Имя Пользователя” пользователь попадает на страницу на которой происходит запрос к менеджеру базы данных и делается выборка данных по

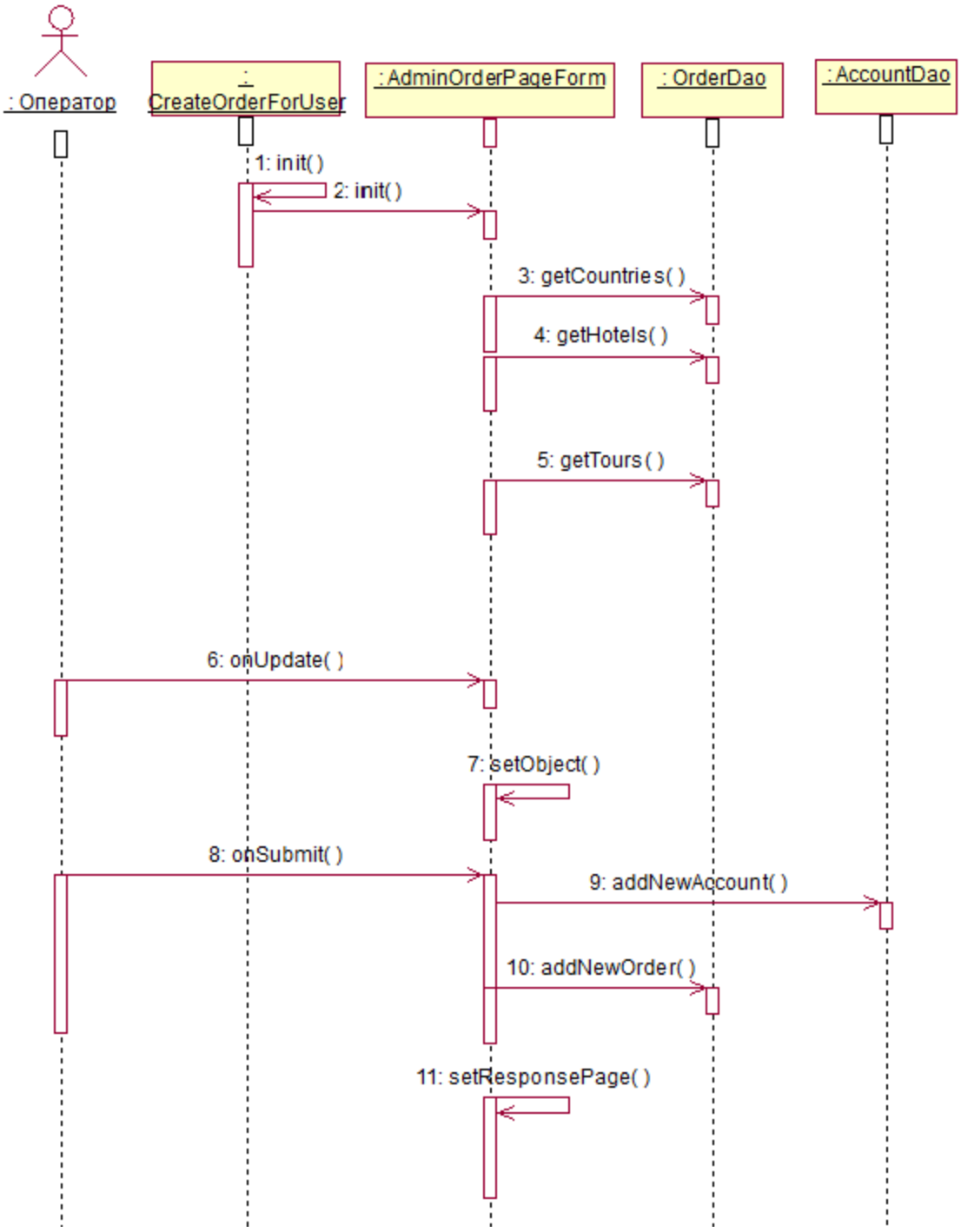
имени пользователя, и заполняются поля: имя, фамилия, возраст, персональные данные и генерируются пустые поля: старый пароль, новый пароль и повторение нового пароля. После изменения данных в полях и нажатии на кнопку “изменить”, происходит запрос к менеджеру базы данных и записываются новые введённые данные для этого пользователя. После заполнения полей старый пароль, новый пароль и повторение нового пароля, происходит запрос к менеджеру базы данных, после чего делается выборка по имени пользователя и в случае удачной выборки, данные введённые в поле новый пароль записываются поверх старого пароль, если выборка была неудачной то выводится сообщение об ошибке.

- Сценарий для Use-Case’а Авторизация:

Открывая страницу, клиент инициализирует процесс авторизации, заполнив поля “логин” и “пароль” и после нажатия кнопки “вход”, выполняется запрос к менеджеру базы данных и производится выборка данных из базы, затем сравниваются данные из базы с введёнными пользователем и если найдено совпадение, то пороисходит переход на главную страницу, в случае если не найдено совпадение то выдаётся сообщение об ошибке авторизации.

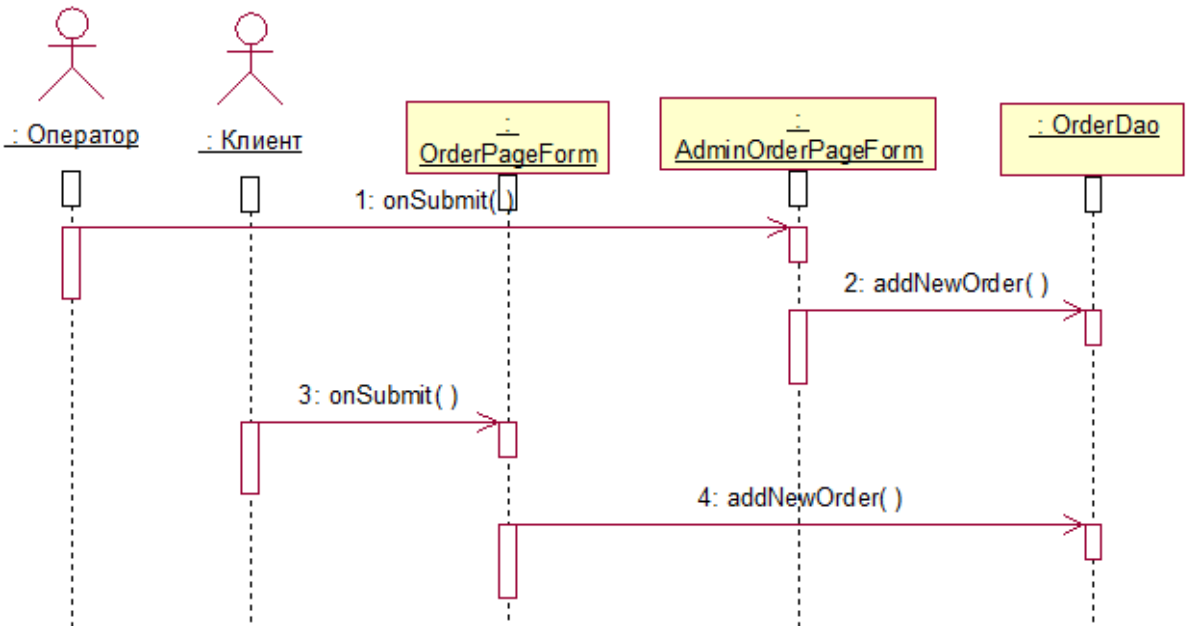
**Разработка диаграмм последовательности**

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



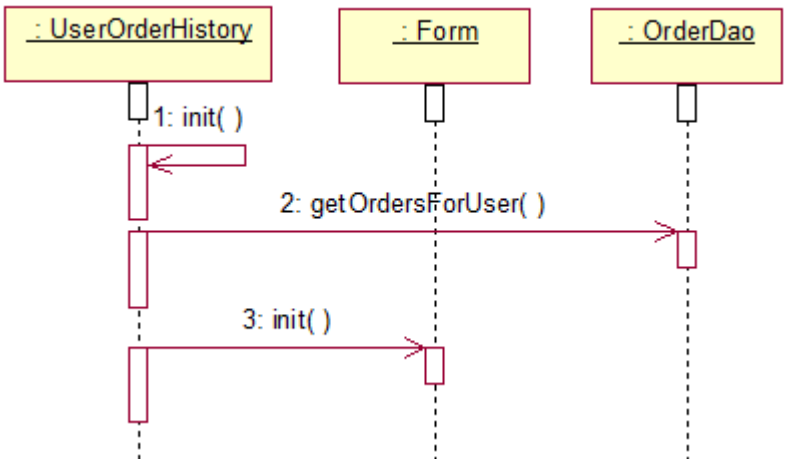

(рис.4)

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



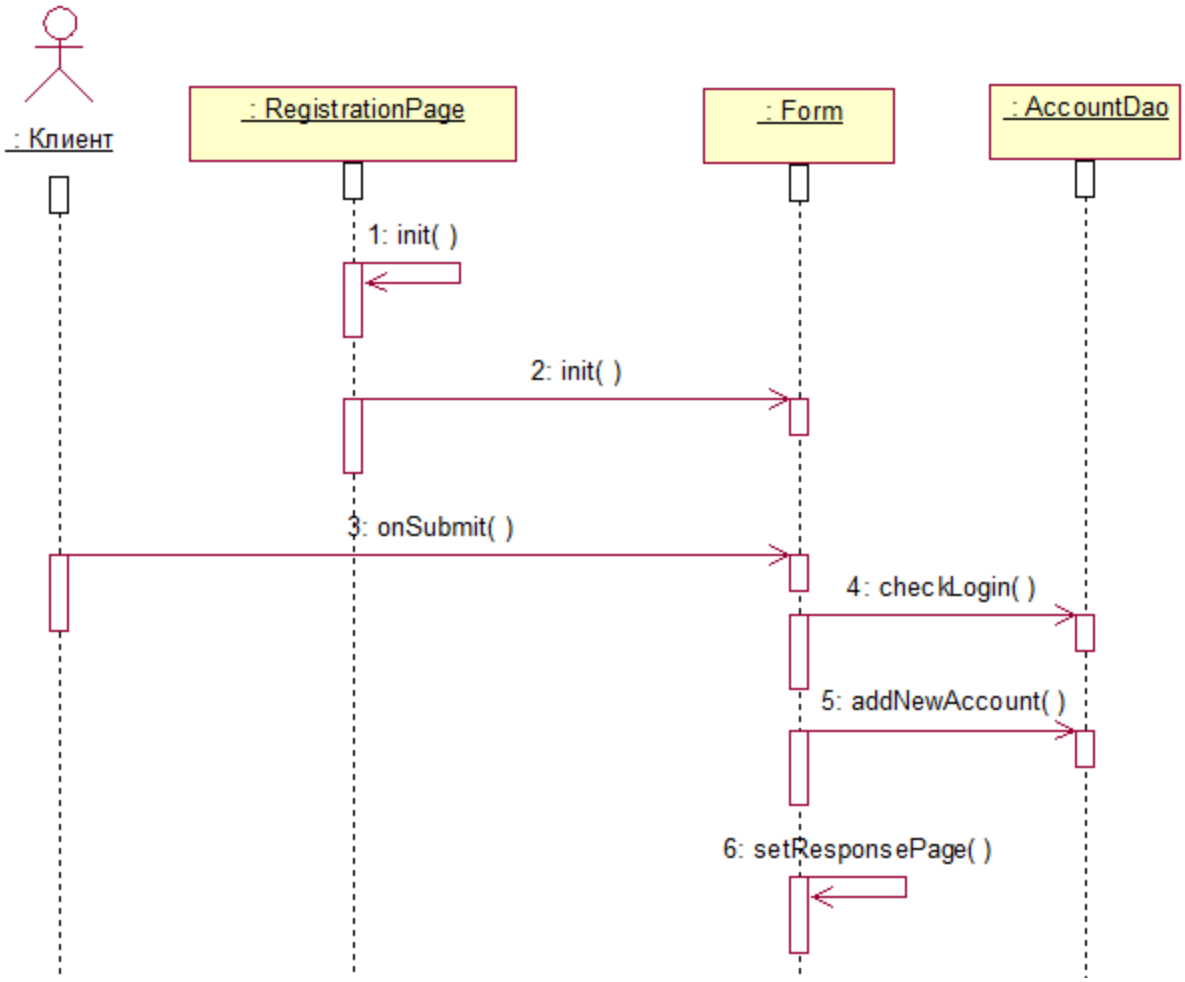

(рис.5)

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



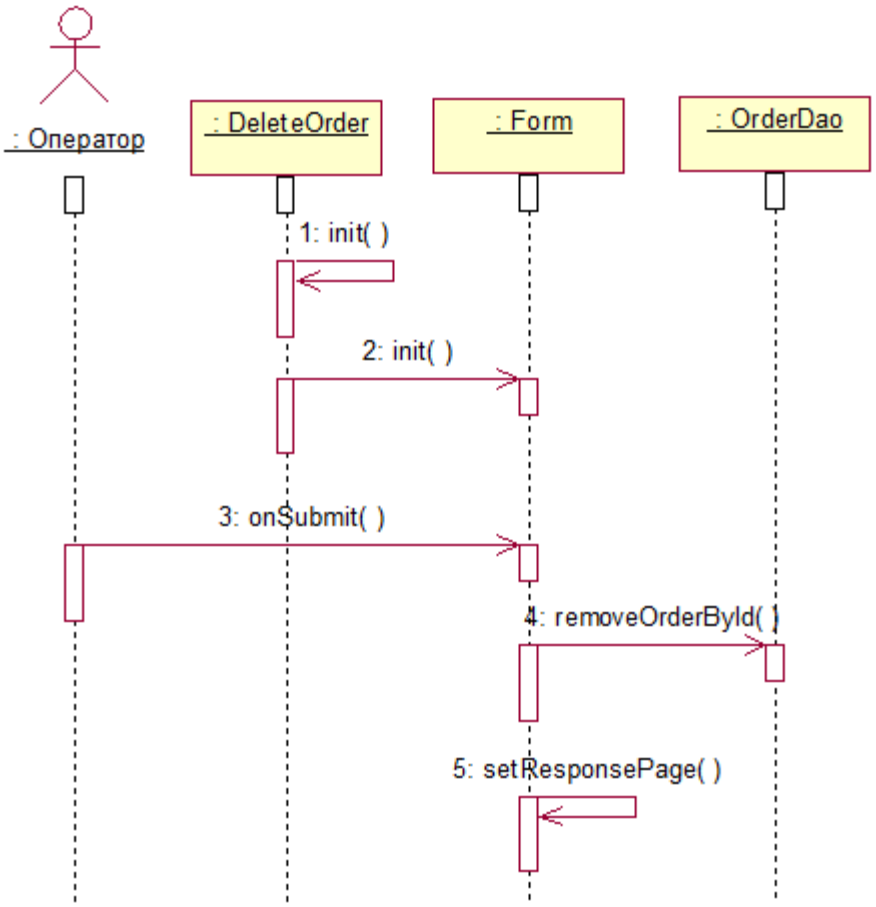

(рис.6)

Диаграмма последовательности для Use-Case'а Регистрирование выглядит следующим образом (рис.7):



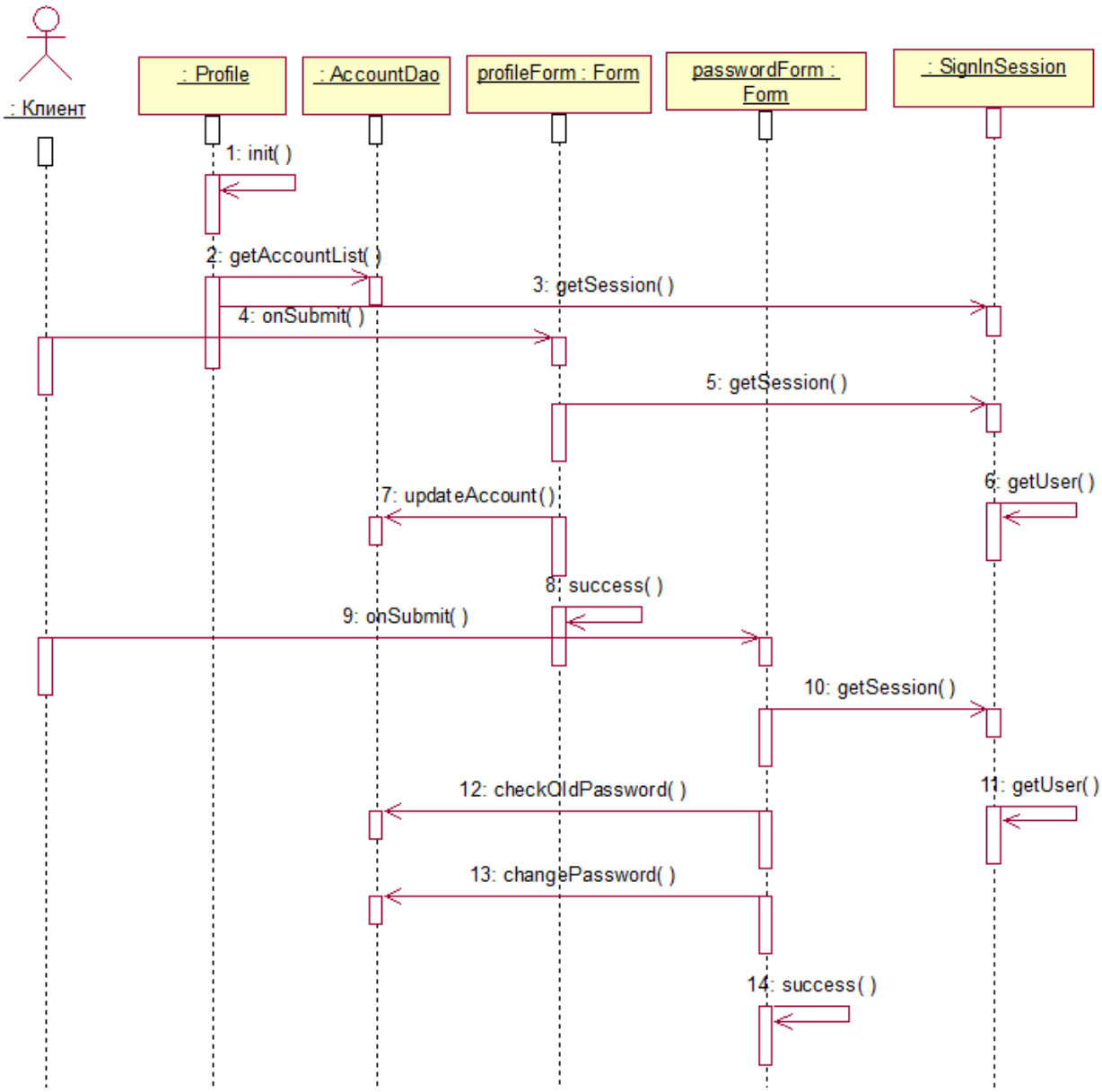

(рис.7)

Диаграмма последовательности для Use-Case'а ОтменаПутёвки выглядит следующим образом (рис.8):



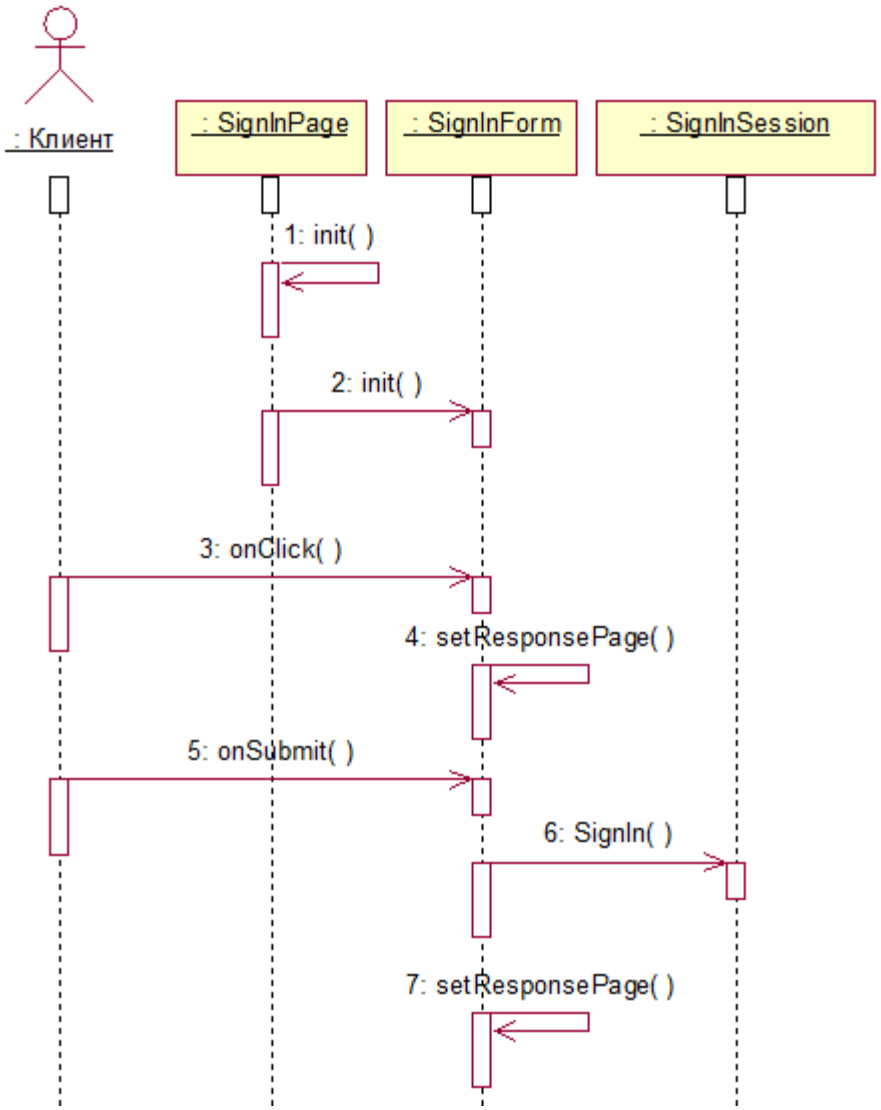

(рис.8)

Диаграмма последовательности для Use-Case'а ИзменениеПерсональныхДанных выглядит следующим образом (рис.9):




(рис.9)

Диаграмма последовательности для Use-Case'а Авторизация выглядит следующим образом (рис.10):




(рис.10)

## Диаграмма классов

На основании разработанных диаграмм последовательностей возможно выделить следующие классы:

- SignInSession - объектом класса является контейнер хранящий в себе текущую сессию.
- SignInForm -объектом класса является форма для авторизации.
- SignInPage - объектом класса является страница авторизации.
- OrderPage - объектом класса является страница заказа путёвки.
- OrderPageForm - объектом класса является форма для заказа путёвки.
    - RegistrationPage - объектом класса является страница регистрации.
- DeleteOrder - объектом класса является страница удаления заказа.

- Form - объектом класса является форма для смены пароля и персональнных данных.
- AccountDao - объектом класса является сервис обращения к базе данных для модели Account.
- OrderDao - объектом класса является сервис обращения к базе данных для модели Order.
- UserOrderHistory - объектом класса является страница отображающая историю пользователя.
- AdminOrderPageForm - объектом класса является форма для заказа путёвок оператором от лица пользователя.
- Profile - объектом класса является страница с личными данными пользователя.
- CreateOrderPageForm - объектом класса является форма для заказа путёвки пользователем.
- CreateOrderForUser - объектом класса является страница для заказа путёвок оператором от лица пользователя.

Также были созданы следующие классы, являющиеся моделями в системе :

- Account
- Country
- Hotel
- Order
- OrderObject
- Tour

Так же важно отметить, что возникла необходимость в создании следующих классов для навигации по страницам:

- HeaderPanel - класс содержащий “шапку страницы”.
- MenuPanel - класс содержащий ссылки на другие страницы.

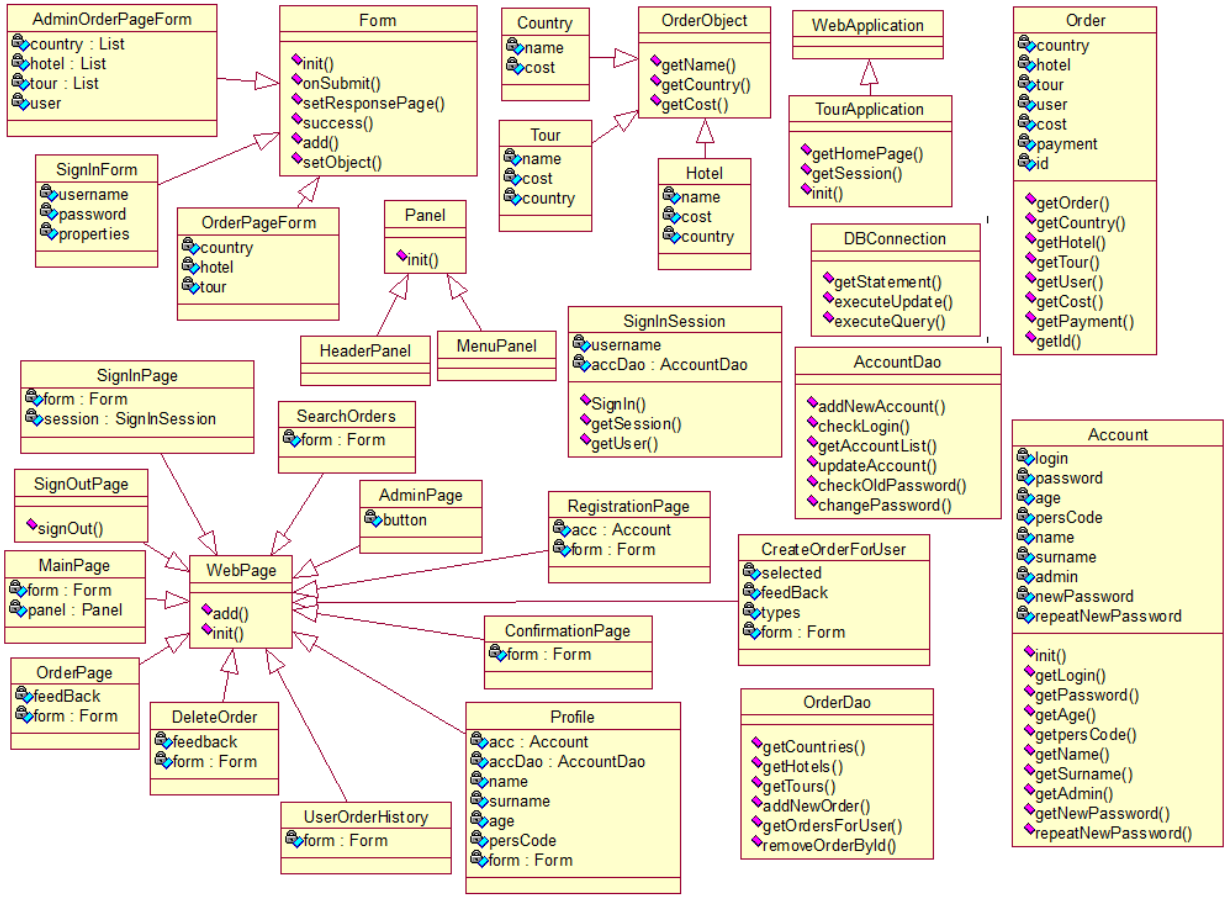
И другие вспомогательные классы:

- WebPage
- Panel
- MainPage
- ConfirmationPage
- SearchOrders
- AdminPage
- SignOutPage
- DBConnection
- TourApplication
- 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.11)

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

| Класс\Метрика | WMC | NOC | DIT | NC | NOM |
|---|---|---|---|---|---|
| WebPage | 2 | 12 | | | |
| Panel | 1 | 2 | | | |
| MainPage | 0 | 0 | | | |
| ConfirmationPage | 0 | 0 | | | |
| SearchOrders | 0 | 0 | | | |
| SignOutPage | 1 | 0 | | | |
| DBConnection | 3 | 0 | | | |
| TourApplication | 3 | 0 | | | |
| WebApplication | 0 | 1 | | | |
| HeaderPanel | 0 | 0 | | | |
| MenuPanel | 0 | 0 | | | |
| Account | 10 | 0 | | | |
| Country | 0 | 0 | | | |
| Hotel | 0 | 0 | | | |
| Order | 8 | 0 | | | |
| OrderObject | 3 | 3 | 1 | 33 | 52 |
| Tour | 0 | 0 | | | |
| SignInSession | 3 | 0 | | | |
| SignInForm | 0 | 0 | | | |
| SignInPage | 0 | 0 | | | |
| OrderPage | 0 | 0 | | | |
| OrderPageForm | 0 | 0 | | | |
| RegistrationPage | 0 | 0 | | | |
| DeleteOrder | 0 | 0 | | | |
| Form | 6 | 3 | | | |
| AccountDao | 6 | 0 | | | |
| OrderDao | 6 | 0 | | | |
| UserOrderHistory | 0 | 0 | | | |
| AdminOrderPageForm | 0 | 0 | | | |
| Profile | 0 | 0 | | | |
| CreateOrderForUser | 0 | 0 | | | |
| Среднее значение | 1.68 | 0.68 | | | |

(табл.1)

Составим начальный план итераций с учётом риска реализации use case’ов:

Итерация №1

1. Регистрация пользователя.
2. Авторизация пользователя.

Итерация №2

1. Изменение персональных данных
2. Смотретьисторию путёвок.
3. Заказ путёвки
4. Оплата заказа

Итерация №3

1. Изменение персональных данных
2. Смотреть историю путёвок.
3. Заказ путёвки
4. Оплата заказа
5. Оформление заказа клиента
6. Отмена путёвки

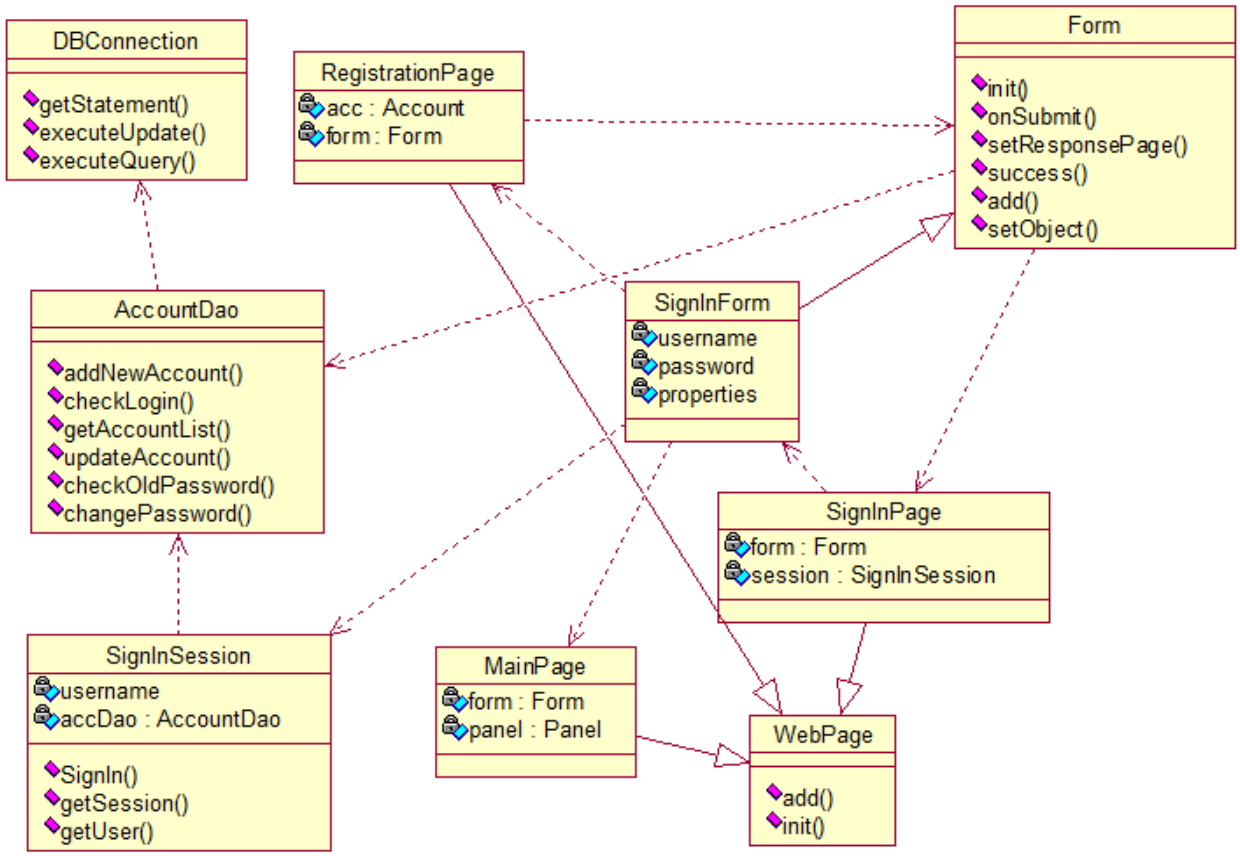
# Этап Конструирование

Итерация 1:

Для реализации сценариев

- Use Case'a Авторизация
- Use Case'a Регистрация

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




(рис.12)

# Оценка качества логической структуры модели с помощью метрик Чидамбера – Кемерера

**Описание процесса подсчёта метрик на первой итерации:**

**Нахождение WMC:** Для нахождениия этой оценки было посчитано индивидуальное количество методов в каждом классе.

**Нахождение NOC:** Для нахождениия этой оценки было посчитано индивидуальное количество детей для каждого класса.

**Нахождение CBO:** Для нахождениия этой оценки было посчитано индивидуальное количество содружевств для этого класса.

**Нахождение RFC:** Для нахождениия этой оценки было посчитано количество методов в классе, а также методы содружных классов.

**Нахождение DIT:** Для нахождениия этой оценки было посчитано максимальнвя длинна наследования от корня.

**Нахождение NC:** Для нахождениия этой оценки было посчитано количество классов.

**Нахождение NOM:** Для нахождениия этой оценки было посчитано количество всех методов.

**Нахождение LOC:** Для нахождениия этой оценки были посчитано сумарное количество строк кода используемых классов.

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

| Класс\Метрика | WMC | NOC | CBO | RFC | LCOM | DIT | NC | NOM | LOC |
|---|---|---|---|---|---|---|---|---|---|
| DBconnection | 3 | 0 | 0 | 0 | 0 | | | | |
| AccountDao | 6 | 0 | 1 | 9 | 0 | | | | |
| SignInSession | 3 | 0 | 1 | 9 | 0 | | | | |
| RegistrationPage | 0 | 0 | 1 | 6 | 0 | | | | |
| MainPage | 0 | 0 | 0 | 0 | 0 | 1 | 9 | 20 | 376 |
| WebPage | 2 | 3 | 0 | 2 | 0 | | | | |
| SignInPage | 0 | 0 | 1 | 0 | 0 | | | | |
| SignInForm | 0 | 0 | 3 | 3 | 0 | | | | |
| Form | 6 | 1 | 2 | 12 | 0 | | | | |
| Среднее значение | 2.22 | 0.44 | 1 | 4.56 | 0 | | | | |

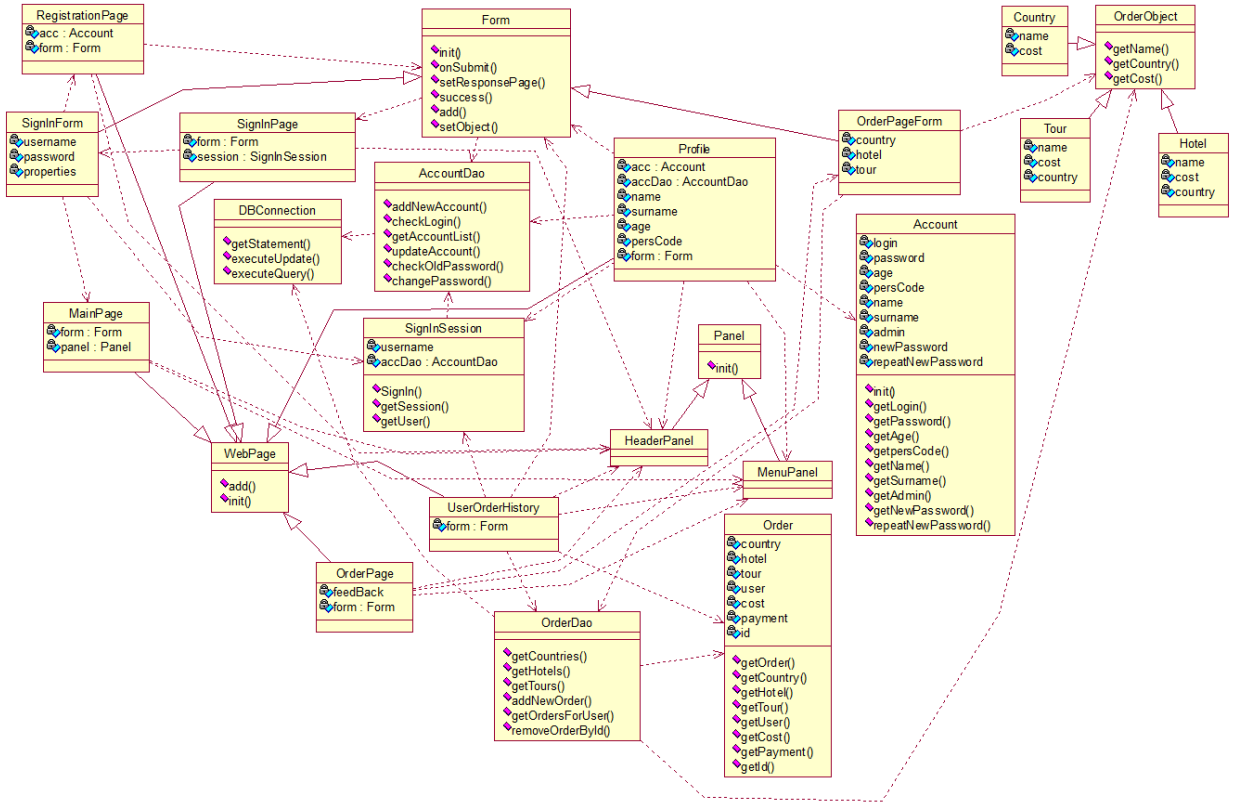
(табл.2)

**Итерация 2:**

Для реализации сценариев

- Use Case’a Изменение персональных данных
- Use Case’a Смотретьисторию путёвок.
- Use Case’a Заказ путёвки
- Use Case’a Оплата заказа

необходимо создать программный код для следующих методов классов представленных на диаграмме ниже :




(рис.13)

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

| Класс\Метрика | WMC | NOC | CBO | RFC | LCOM | DIT | NC | NOM | LOC |
|---|---|---|---|---|---|---|---|---|---|
| DBconnection | 3 | 0 | 0 | 3 | 0 | | | | |
| AccountDao | 6 | 0 | 1 | 9 | 0 | | | | |
| SignInSession | 3 | 0 | 1 | 9 | 0 | | | | |
| RegistrationPage | 0 | 0 | 3 | 6 | 0 | | | | |
| MainPage | 0 | 0 | 2 | 0 | 0 | | | | |
| WebPage | 2 | 6 | 0 | 2 | 0 | | | | |
| SignInPage | 0 | 0 | 2 | 0 | 0 | | | | |
| SignInForm | 0 | 0 | 3 | 3 | 0 | | | | |
| Form | 6 | 2 | 2 | 12 | 0 | | | | |
| OrderPage | 0 | 0 | 3 | 0 | 0 | | | | |
| OrderDao | 6 | 0 | 3 | 18 | 0 | | | | |
| OrderObject | 3 | 3 | 0 | 3 | 0 | 1 | 22 | 48 | 1343 |
| MenuPanel | 0 | 0 | 0 | 1 | 0 | | | | |
| HeaderPanel | 0 | 0 | 0 | 0 | 0 | | | | |
| Order | 8 | 0 | 0 | 8 | 0 | | | | |
| Country | 0 | 0 | 0 | 0 | 0 | | | | |
| Tour | 0 | 0 | 0 | 0 | 0 | | | | |
| Hotel | 0 | 0 | 0 | 0 | 0 | | | | |
| UserOrderHistory | 0 | 0 | 4 | 23 | 0 | | | | |
| Profile | 0 | 0 | 6 | 25 | 0 | | | | |
| Panel | 1 | 2 | 0 | 1 | 0 | | | | |
| Account | 10 | 0 | 0 | 10 | 0 | | | | |
| OrderPageForm | 0 | 0 | 2 | 3 | 0 | | | | |
| Среднее значение | 2.09 | 0.57 | 1.39 | 5.91 | 0 | | | | |

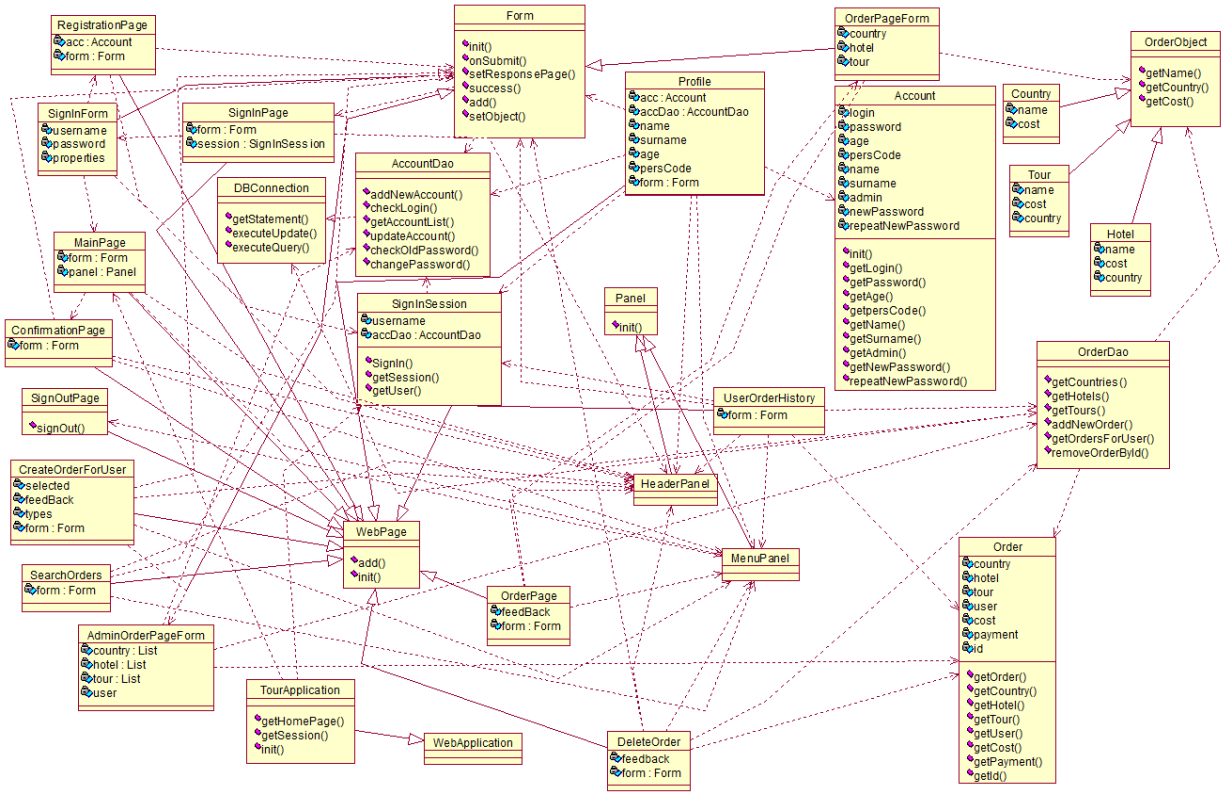
(табл.3)

**Итерация 3:**

Для реализации сценариев

- Use Case’a Оформление заказа клиента
- Use Case’a Отмена путёвки

необходимо создать программный код для следующих методов классов представленных на диаграмме ниже




(рис.14)

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

| Класс\Метрика | WMC | NOC | CBO | RFC | LCOM | DIT | NC | NOM | LOC |
|---|---|---|---|---|---|---|---|---|---|
| DBconnection | 3 | 0 | 0 | 3 | 0 | | | | |
| AccountDao | 6 | 0 | 1 | 9 | 0 | | | | |
| SignInSession | 3 | 0 | 1 | 15 | 0 | | | | |
| RegistrationPage | 0 | 0 | 3 | 6 | 0 | | | | |
| MainPage | 0 | 0 | 2 | 6 | 0 | | | | |
| WebPage | 2 | 11 | 0 | 2 | 0 | | | | |
| SignInPage | 0 | 0 | 2 | 0 | 0 | | | | |
| SignInForm | 0 | 0 | 3 | 6 | 0 | | | | |
| Form | 6 | 3 | 2 | 12 | 0 | | | | |
| OrderPage | 0 | 0 | 3 | 0 | 0 | | | | |
| OrderDao | 6 | 0 | 3 | 17 | 0 | | | | |
| OrderObject | 3 | 3 | 0 | 3 | 0 | | | | |
| MenuPanel | 0 | 0 | 1 | 1 | 0 | | | | |
| HeaderPanel | 0 | 0 | 0 | 0 | 0 | | | | |
| Order | 8 | 0 | 0 | 8 | 0 | | | | |
| Country | 0 | 0 | 0 | 0 | 0 | 1 | 33 | 52 | 1807 |
| Tour | 0 | 0 | 0 | 0 | 0 | | | | |
| Hotel | 0 | 0 | 0 | 0 | 0 | | | | |
| UserOrderHistory | 0 | 0 | 5 | 23 | 0 | | | | |
| Profile | 0 | 0 | 6 | 25 | 0 | | | | |
| Panel | 1 | 2 | 0 | 1 | 0 | | | | |
| Account | 10 | 0 | 0 | 10 | 0 | | | | |
| OrderPageForm | 0 | 0 | 2 | 3 | 0 | | | | |
| ConformationPage | 0 | 0 | 3 | 6 | 0 | | | | |
| SignOutPage | 1 | 0 | 0 | 1 | 0 | | | | |
| CreateOrderForUser | 0 | 0 | 4 | 6 | 0 | | | | |
| SearchOrders | 0 | 0 | 3 | 6 | 0 | | | | |
| AdminOrderPageForm | 0 | 0 | 3 | 20 | 0 | | | | |
| TourApplication | 3 | 0 | 2 | 6 | 0 | | | | |
| WebApplication | 0 | 1 | 0 | 0 | 0 | | | | |
| DeleteOrder | 0 | 0 | 5 | 20 | 0 | | | | |
| Среднее значение | 1.68 | 0.65 | 1.74 | 6.94 | 0 | | | | |

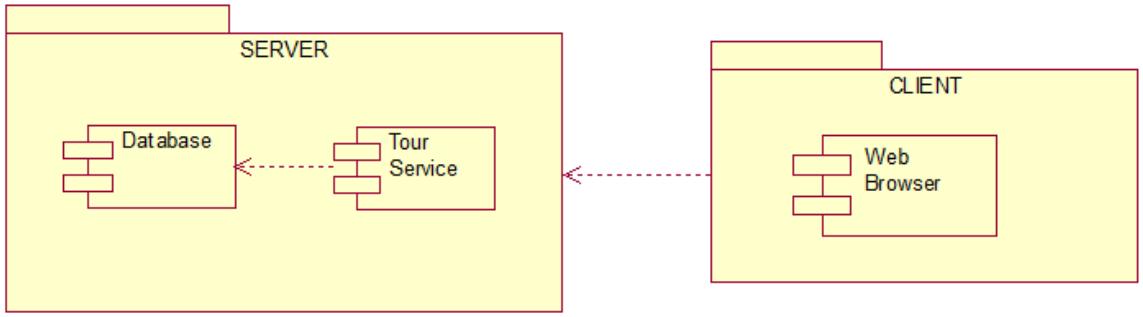
(табл.4)

## Анализ полученных результатов

| Итерация\Метрика | WMC | NOC | CBO | RFC | LCOM | DIT | NC | NOM | LOC |
|---|---|---|---|---|---|---|---|---|---|
| Итерация №1 | 2.22 | 0.44 | 1 | 4.56 | 0 | 1 | 9 | 20 | 376 |
| Итерация №2 | 2.09 | 0.57 | 1.39 | 5.91 | 0 | 1 | 22 | 48 | 1343 |
| Итерация №3 | 1.68 | 0.65 | 1.74 | 6.94 | 0 | 1 | 33 | 52 | 1807 |

Сравнивая полученные значения метрик, можно заметить, что на каждой итерации почти все значения постепенно увеличились, а это в свою очередь свидетельствует о постепенном возрастании сложности продукта, что не является положительным результатом. Чем сложнее система, тем сложнее тестирование и отладка, в будущем так же могут выявиться не нужные классы, которые будут висеть в системе, так же будет сложнее расширять систему.

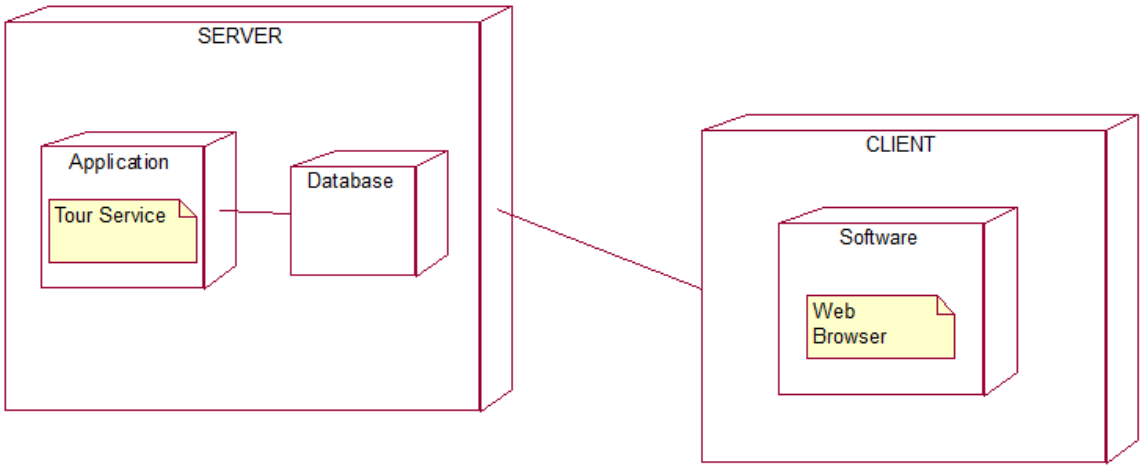
**Диаграмма компонент**
На рисунке представлена компонентная диаграмма (Рис.15).




(Рис.15)

# Диаграмма размещения

На рисунке представлена диаграмма размещения (Рис.16).



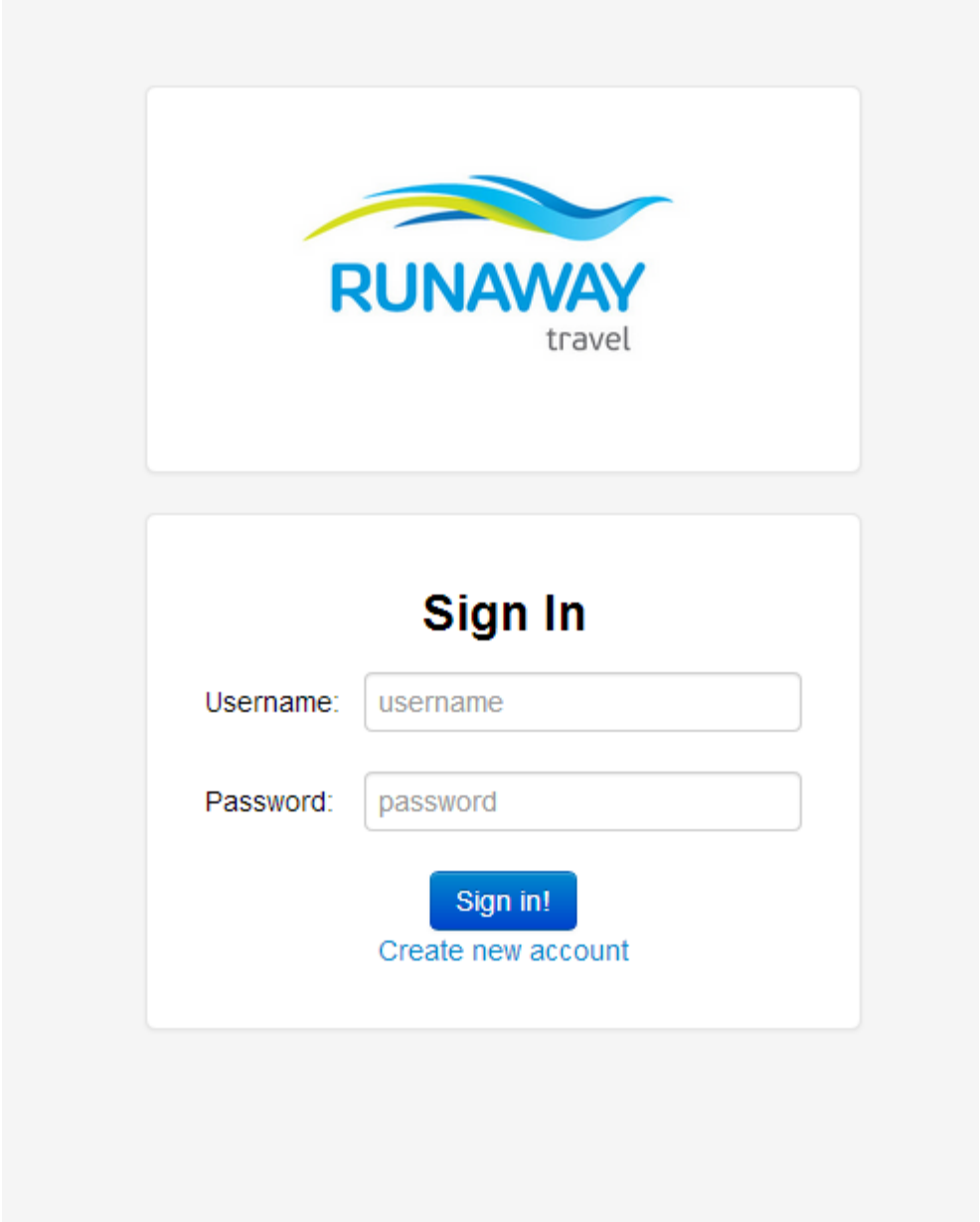

(Рис.16)

# Выводы

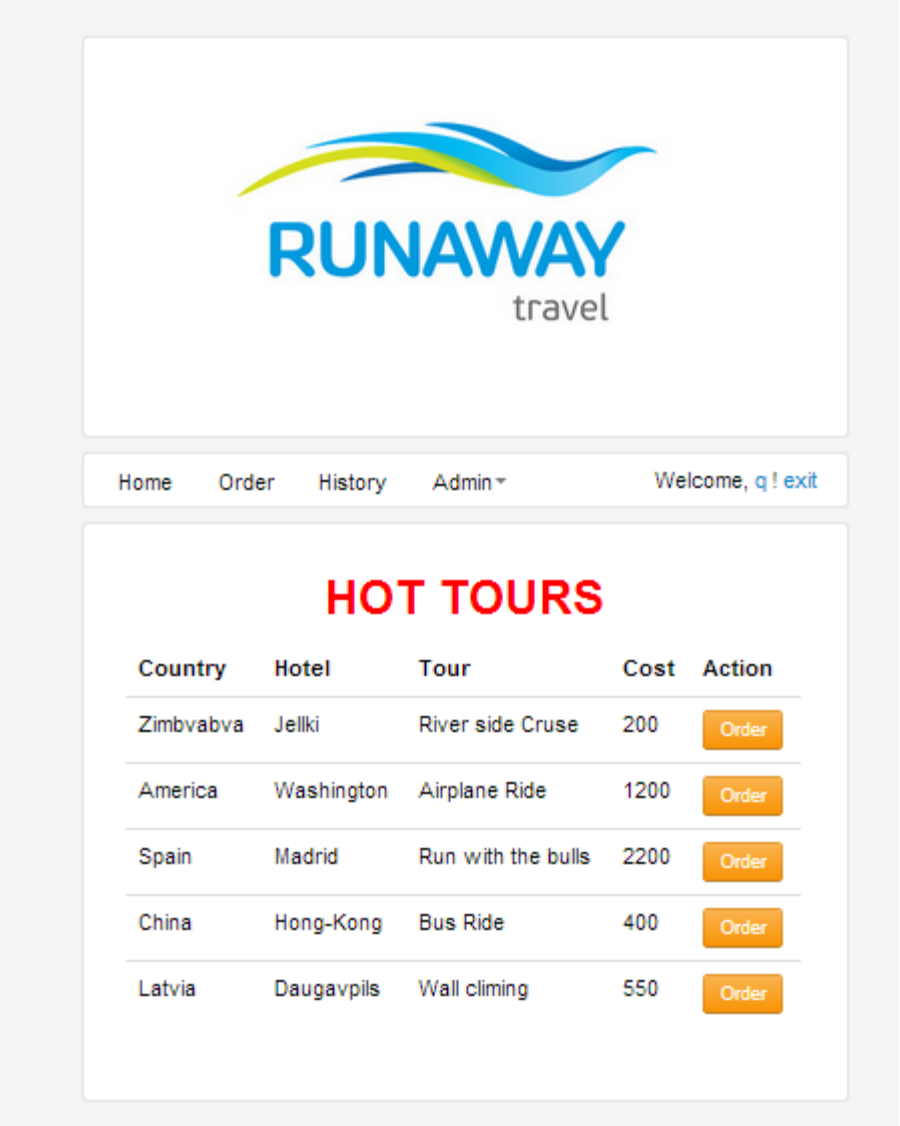
В ходе написания курсового проекта были применены на практике и более подробно изучены технологии визуального конструирования сложных программных систем. Был получен дополнительный опыт в разработке визуальной модели системы и оценивания качества с помощью набора метрик Чидамбера-Кемерера. Итерацеонное планирование конструирования помогло выявить проблемные классы, после чего они были либо изменены, либо устранены полностью. В результате проделанной работы была получена объектно-ориентированная программная система с заданными характеристиками и отвечающая поставленным требованиям.

Приложение:

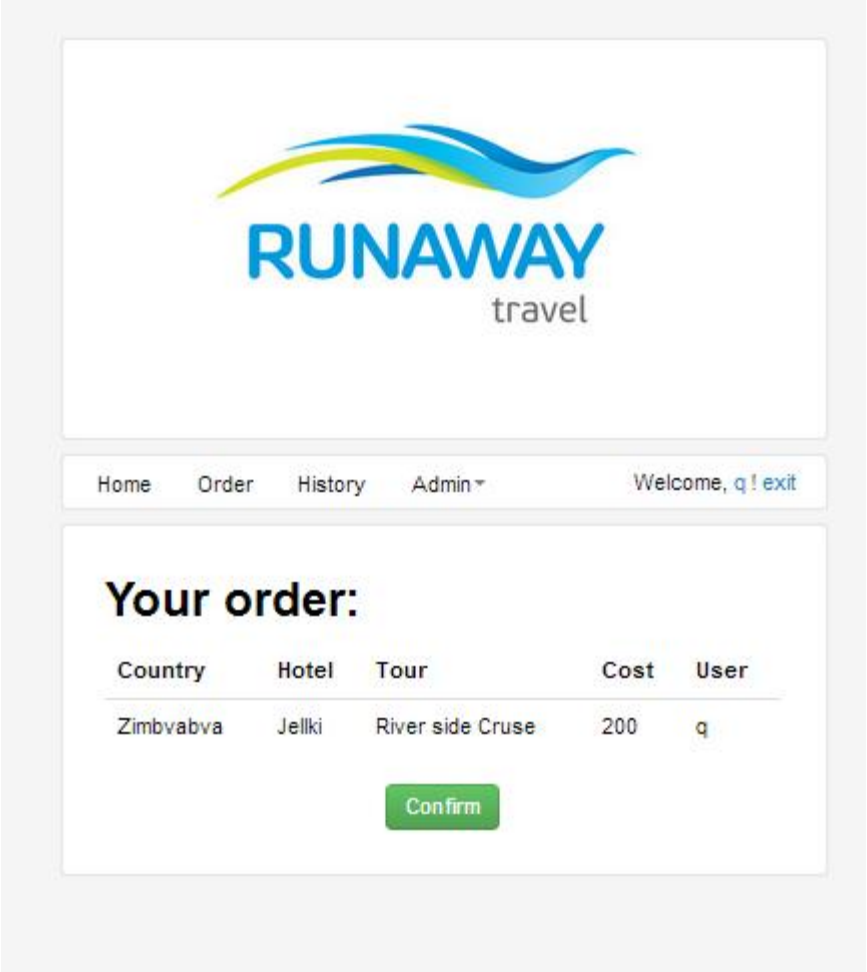
Результат этапа кодирование:



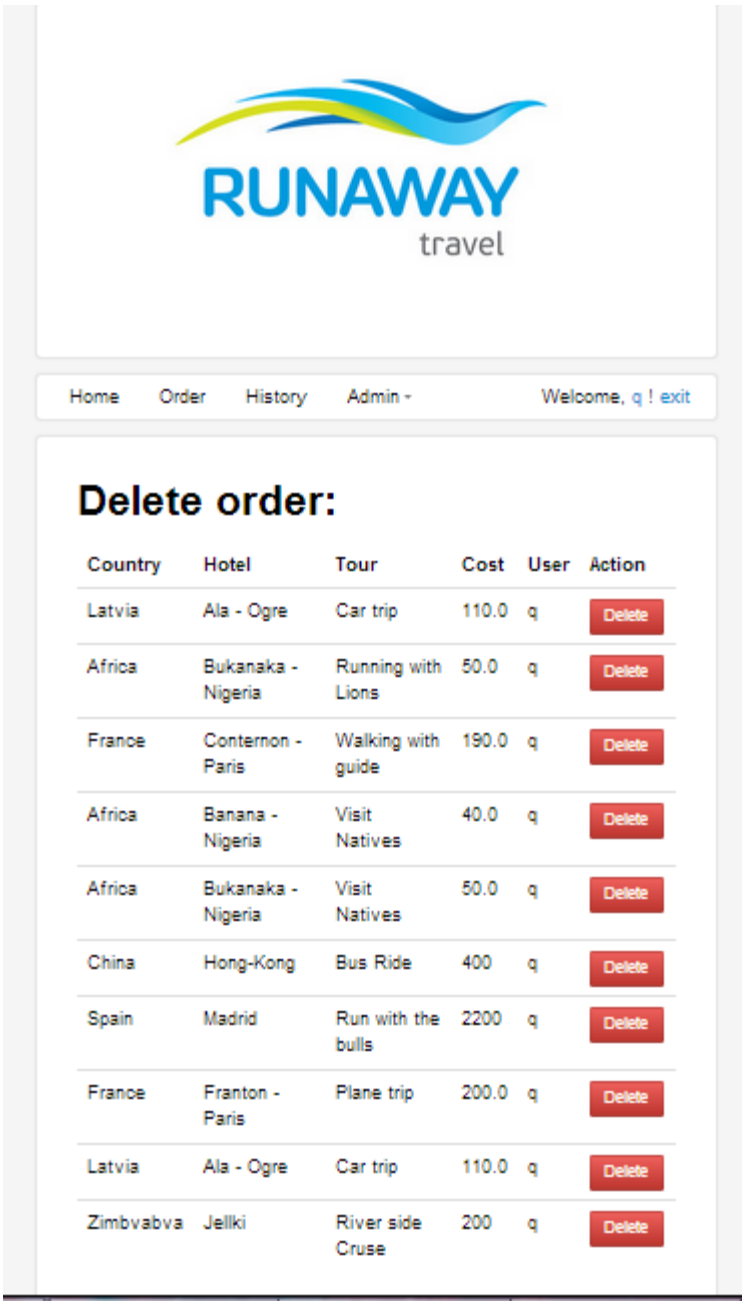

Страница Авторизации



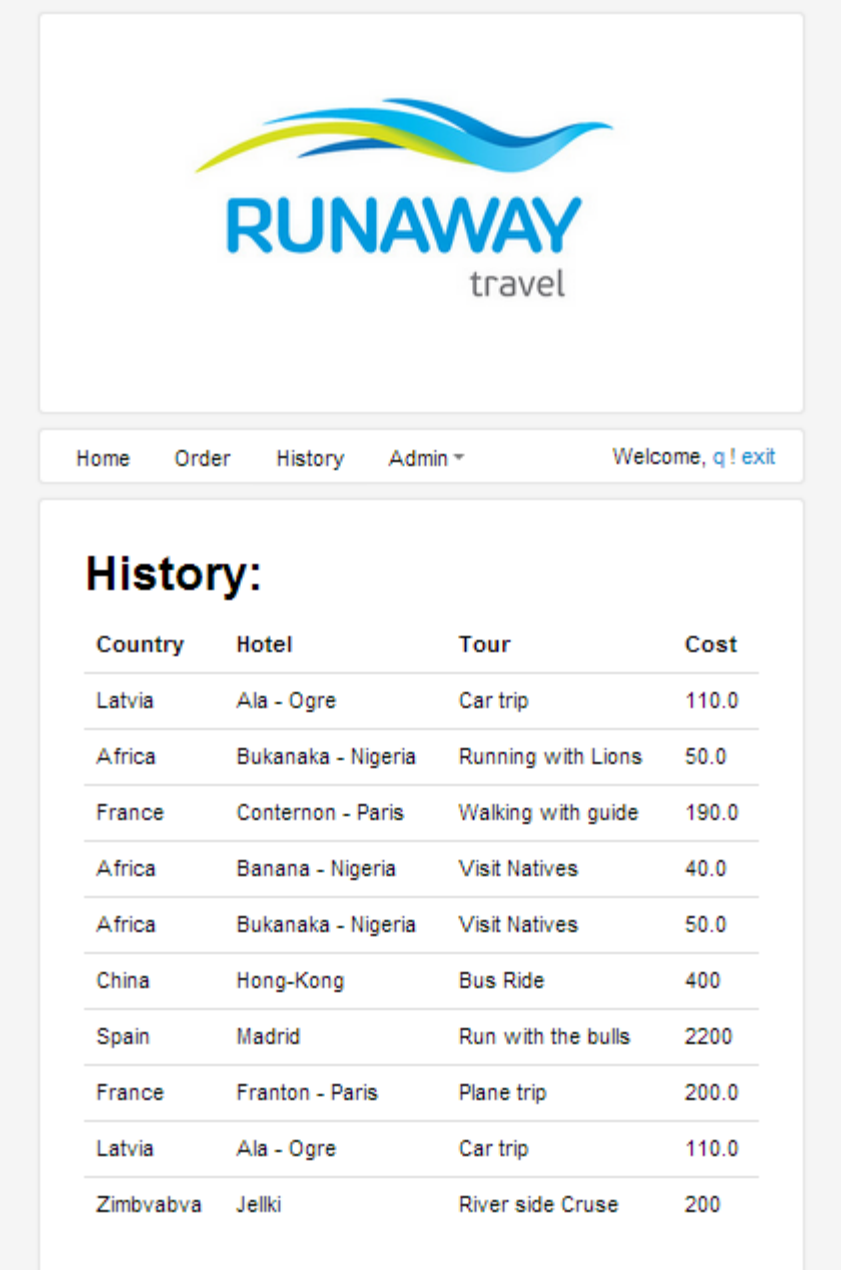

Страница Main



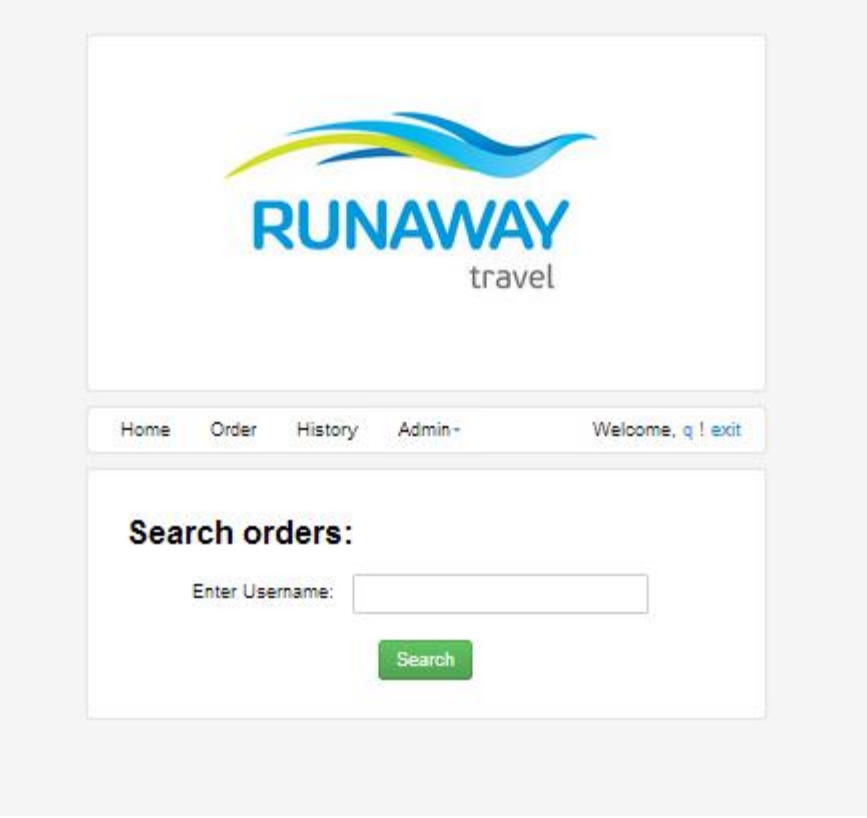

Страница Confirm



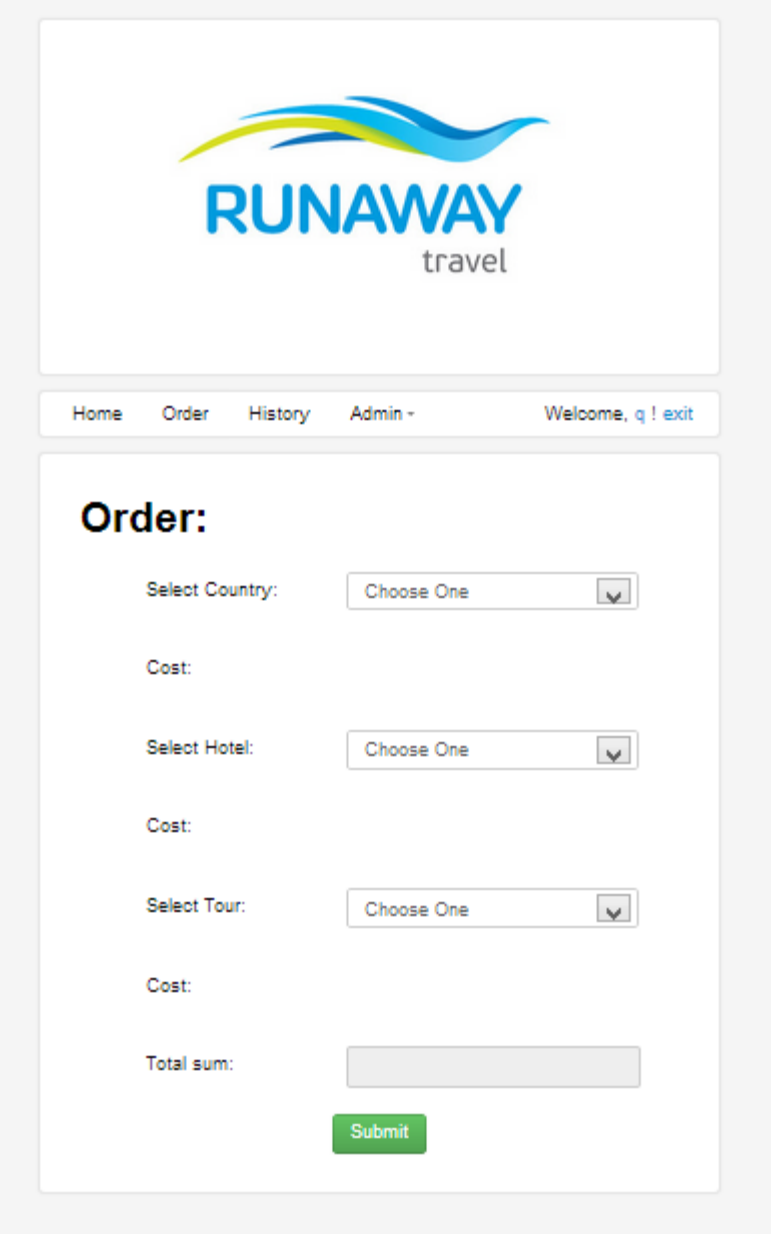

Страница Delete



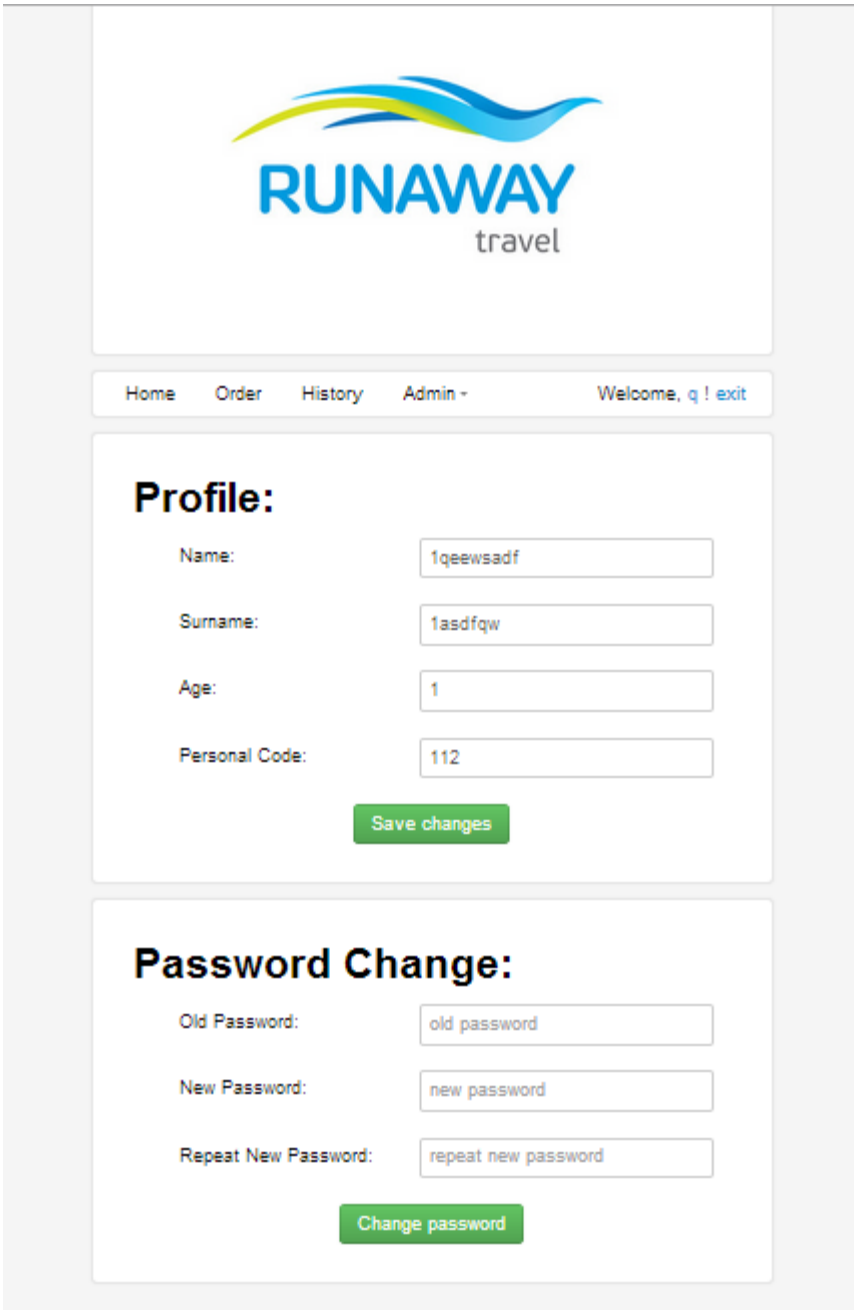

Страница History



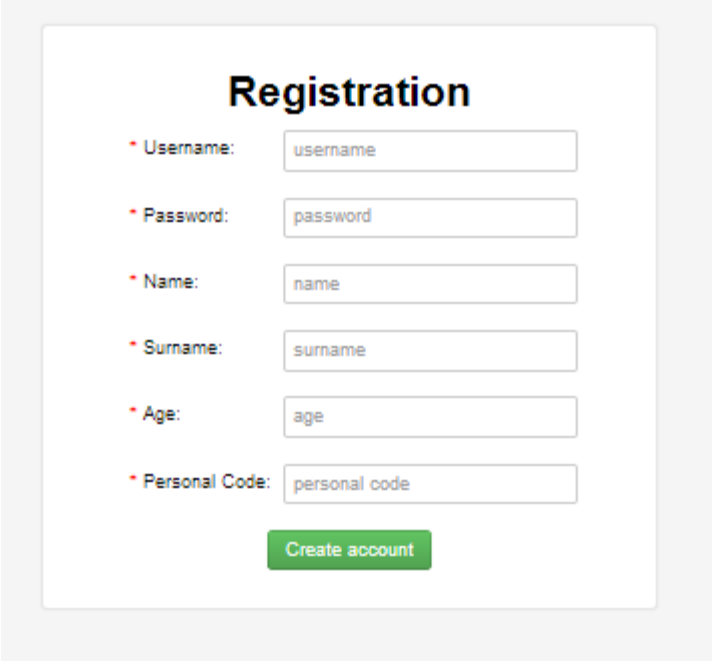

Страница Search



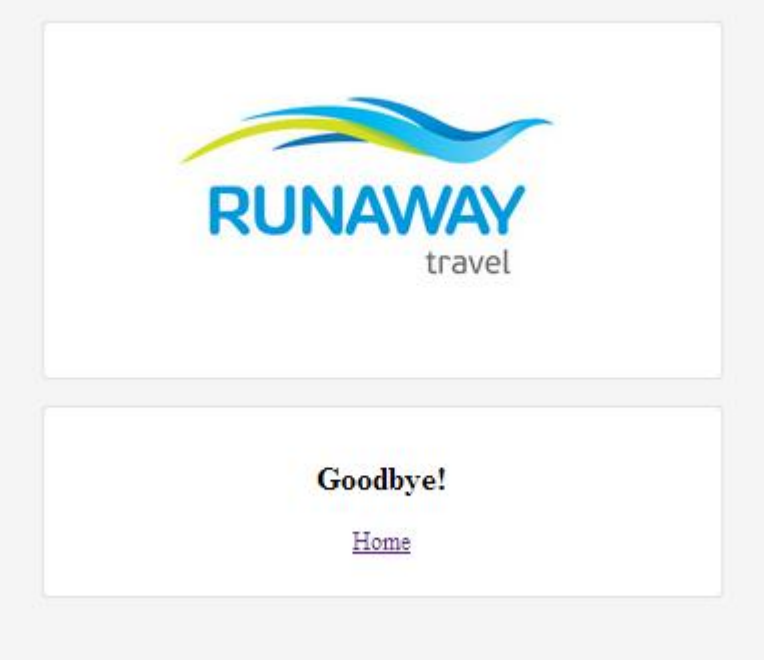

Страница Order/Заказ




Страница Profile




Страница Регистрации




Страница SignOut

*Класс Account*

```
package tour.account;

public class Account {
```

```
    private String login;
    private String password;
    private String age;
    private String persCode;
    private String name;
    private String surname;
    private Boolean admin;
    private String newPassword;
    private String repeatNewPassword;

    public Account() {
    }

    public Account(String login, String password, String name, String surname, String age, String
persCode) {
        this.login = login;
        this.password = password;
        this.name = name;
        this.surname = surname;
        this.age = age;
        this.persCode = persCode;
    }

    public String getLogin() {
        if (login == null) {
            return "";
        }
        return login;
    }

    public String getPassword() {
        if (password == null) {
            return "";
        }
        return password;
    }

    public String getAge() {
        if (age == null) {
            return "";
        }
        return age;
    }
```

```
public void setAge(String age) {
    this.age = age;
}

public String getPersCode() {
    if (persCode == null) {
        return "";
    }
    return persCode;
}

public void setPersCode(String persCode) {
    this.persCode = persCode;
}

public String getName() {
    if (name == null) {
        return "";
    }
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getSurname() {
    if (surname == null) {
        return "";
    }
    return surname;
}

public void setSurname(String surname) {
    this.surname = surname;
}

public Boolean isAdmin() {
    return admin;
}

public void setAdmin(Boolean admin) {
    this.admin = admin;
```

```
    }

    public String getNewPassword() {
        if (newPassword == null) {
            return "";
        }
        return newPassword;
    }

    public String getRepeatNewPassword() {
        if (repeatNewPassword == null) {
            return "";
        }
        return repeatNewPassword;
    }
}
```

*Класс AccountDao*

```
package tour.account;

import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.List;
import tour.dbconnection.DBConnection;

public class AccountDAO {

    public static void addNewAccount(String login, String password, String name, String surname, String
age, String persCode) {
        String insert = "INSERT INTO account VALUES ('" + login + "','" + password + "','" + name + "','"
+ surname + "','" + age + "','" + persCode + "')";
        DBConnection.executeUpdate(insert);
    }

    public static void addNewAccount(String login, String pass){
        String insert = "INSERT INTO account(login,password) VALUES ('" + login + "','"+pass+"');";
        DBConnection.executeUpdate(insert);
    }

    public static void updateAccount(String login, String name, String surname, String age, String
persCode) {
        String insert = "UPDATE account SET name='" + name + "', surname='" + surname + "', age='" +
age + "', persCode='" + persCode + "' WHERE login='" + login + "';";
        DBConnection.executeUpdate(insert);
    }
```

```
    public static void changePassword(String login, String newpassword) {
        String update = "UPDATE account SET password="' + newpassword + "' WHERE login="' + login
+ """;
        DBConnection.executeUpdate(update);
    }

    public static List<Account> getAccountList() {
        try {
            List<Account> list = new ArrayList<Account>();
            String select = "SELECT * FROM account";
            ResultSet query = DBConnection.executeQuery(select);
            while (query.next()) {

                String login = query.getString("login");
                String password = query.getString("password");
                String name = query.getString("name");
                String surname = query.getString("surname");
                String age = query.getString("age");
                String persCode = query.getString("persCode");

                list.add(new Account(login, password, name, surname, age, persCode));
            }
            return list;
        } catch (Exception e) {
            return null;
        }
    }

    public static boolean checkIsUserRegistered(String user) {
        boolean result = false;
        try {
            String select = "SELECT login FROM account where login="' + user + "';";
            ResultSet query = DBConnection.executeQuery(select);
            while (query.next()) {
                String username = query.getString("login");
                if (!username.equals(null)) {
                    result = true;
                    query.close();
                }
            }
            return result;
        } catch (Exception e) {
            return result;
```

```
    }
}

public static boolean checkIsAdmin(String login) {
    boolean result = false;
    try {
        String select = "SELECT login,isAdmin FROM account";
        ResultSet query = DBConnection.executeQuery(select);
        while (query.next()) {
            Boolean isAdmin = query.getBoolean("isAdmin");
            String username = query.getString("login");
            if (login.equals(username)) {
                if (isAdmin) {
                    result = true;
                    query.close();
                }
            }
        }
        return result;
    } catch (Exception e) {
        return result;
    }
}

public static boolean checkLogin(String login) {
    boolean bool = false;
    try {
        String select = "SELECT login FROM account";
        ResultSet query = DBConnection.executeQuery(select);
        while (query.next()) {
            String username = query.getString("login");
            if (login.equals(username)) {
                bool = true;
                query.close();
            }
        }
        return bool;
    } catch (Exception e) {
        return bool;
    }
}

public static boolean checkOldPassword(String login, String password) {
    boolean bool = false;
```

```
        try {
            String select = "SELECT password FROM account WHERE login='" + login + "'";
            ResultSet query = DBConnection.executeQuery(select);
            while (query.next()) {
                String userpassword = query.getString("password");
                if (password.equals(userpassword)) {
                    bool = true;
                    query.close();
                }
            }
            return bool;
        } catch (Exception e) {
            return bool;
        }
    }
}
```

*Класс AdminPage*

```
package tour.admin;

import tour.main.MainPage;
import org.apache.wicket.request.mapper.parameter.PageParameters;
import org.apache.wicket.markup.html.WebPage;
import org.apache.wicket.markup.html.link.BookmarkablePageLink;
import org.apache.wicket.markup.html.link.Link;
import tour.admin.functions.CreateOrderForUser;
import tour.admin.functions.DeleteOrder;
import tour.admin.functions.SearchOrders;

public final class AdminPage extends WebPage {

    public AdminPage() {
        super();
        add(new BookmarkablePageLink<SearchOrders>("SearchOrders", SearchOrders.class));
        add(new BookmarkablePageLink<CreateOrderForUser>("CreateOrderForUser",
CreateOrderForUser.class));
        add(new BookmarkablePageLink<MainPage>("MainPage", MainPage.class));
    }

    public AdminPage(PageParameters params) {
    }
}
```

*Класс CreateOrderForUser*

```
package tour.admin.functions;
```

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import tour.account.AccountDAO;
import tour.menupanel.MenuPanel;
import tour.session.SignInSession;
import org.apache.wicket.ajax.AjaxRequestTarget;
import org.apache.wicket.ajax.form.AjaxFormComponentUpdatingBehavior;
import org.apache.wicket.request.mapper.parameter.PageParameters;
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.form.RadioChoice;
import org.apache.wicket.markup.html.form.TextField;
import org.apache.wicket.markup.html.panel.FeedbackPanel;
import org.apache.wicket.model.AbstractReadOnlyModel;
import org.apache.wicket.model.IModel;
import org.apache.wicket.model.Model;
import org.apache.wicket.model.PropertyModel;
import tour.header.HeaderPanel;
import tour.order.OrderDao;
import tour.order.models.Order;
import tour.order.models.OrderObject;

public final class CreateOrderForUser extends WebPage {

    final List<String> TYPES = Arrays
            .asList(new String[]{"Cash", "Card"});
    String selected = "Card";
    private String feedBack;

    public CreateOrderForUser() {
        super();
        add(new HeaderPanel("headerPanel"));
        add(new MenuPanel("menuPanel"));
        Form<?> form = new AdminOrderPageForm("AdminOrderPageForm");

        RadioChoice<String> paymentType = new RadioChoice<String>(
                "hosting", new PropertyModel<String>(this, "selected"), TYPES);
        form.add(paymentType);
```

```
    add(form);
}

public CreateOrderForUser(PageParameters params) {
    feedBack = params.get("feedBack").toString();
    add(new HeaderPanel("headerPanel"));
    add(new MenuPanel("menuPanel"));
    Form<?> form = new AdminOrderPageForm("AdminOrderPageForm");
    RadioChoice<String> paymentType = new RadioChoice<String>(
            "hosting", new PropertyModel<String>(this, "selected"), TYPES);
    form.add(paymentType);
    add(form);
}

public final class AdminOrderPageForm extends Form<Void> {

    private Label countryCost;
    private Label hotelCost;
    private Label tourCost;
    private TextField totalCostSum;
    private OrderDao orderDao;
    private final Map<String, List<String>> hotelOptions;
    private final Map<String, List<String>> tourOptions;
    private final DropDownChoice<String> countryDropDown;
    private DropDownChoice<String> hotelDropDown;
    private DropDownChoice<String> tourDropDown;
    private List<OrderObject> hotels;
    private List<OrderObject> tours;
    private List<OrderObject> countries;
    private String selectedOption;
    private String selectedHotel;
    private String selectedTour;
    private String calculatedTotalCostSum;
    private Order order;
    private Model<String> countryCostModel;
    private Model<String> hotelCostModel;
    private Model<String> tourCostModel;
    private PropertyModel<String> totalCostSumModel;
    private TextField<String> username;
    private TextField<String> email;
    private FeedbackPanel errorPanel;

    public AdminOrderPageForm(String id) {
        super(id);
```

```
        errorPanel = new FeedbackPanel("feedback");
        username = new TextField<String>("username",
              Model.of(""));
        email = new TextField<String>("email",
              Model.of(""));
        countryCostModel = Model.of("");
        hotelCostModel = Model.of("");
        tourCostModel = Model.of("");

        countryCost = new Label("countryCostLabel", countryCostModel);
        hotelCost = new Label("hotelCostLabel", hotelCostModel);
        tourCost = new Label("tourCostLabel", tourCostModel);

        order = new Order();
        orderDao = new OrderDao();
        hotelOptions = new HashMap<String, List<String>>();
        tourOptions = new HashMap<String, List<String>>();

        countries = orderDao.getCountries();
        hotels = orderDao.getHotels();
        tours = orderDao.getTours();

        for (OrderObject country : countries) {
           hotelOptions.put(country.getName(), getNames(hotels, country.getName()));
           tourOptions.put(country.getName(), getNames(tours, country.getName()));
        }

        IModel<List<? extends String>> makeCountryChoises = new AbstractReadOnlyModel<List<?
extends String>>() {
           @Override
           public List<String> getObject() {
              return new ArrayList<String>(hotelOptions.keySet());
           }
        };

        IModel<List<? extends String>> modelTownChoices = new AbstractReadOnlyModel<List<?
extends String>>() {
           @Override
           public List<String> getObject() {
              List<String> models = hotelOptions.get(selectedOption);
              if (models == null) {
                 models = Collections.emptyList();
              }
```

```
            return models;
        }
    };

    IModel<List<? extends String>> modelTourChoices = new AbstractReadOnlyModel<List<?
extends String>>() {
        @Override
        public List<String> getObject() {
            List<String> models = tourOptions.get(selectedOption);
            if (models == null) {
                models = Collections.emptyList();
            }
            return models;
        }
    };

    countryDropDown = new DropDownChoice<String>("countryDropDown",
            new PropertyModel<String>(this, "selectedOption"), makeCountryChoises);

    hotelDropDown = new DropDownChoice<String>("hotelDropDown",
            new PropertyModel<String>(this, "selectedHotel"), modelTownChoices);

    tourDropDown = new DropDownChoice<String>("tourDropDown",
            new PropertyModel<String>(this, "selectedTour"), modelTourChoices);
    totalCostSumModel = new PropertyModel<String>(this, "calculatedTotalCostSum");
    totalCostSum = new TextField("totalCostSumLabel", totalCostSumModel);

    hotelDropDown.setOutputMarkupId(true);
    tourDropDown.setOutputMarkupId(true);
    countryCost.setOutputMarkupId(true);
    hotelCost.setOutputMarkupId(true);
    tourCost.setOutputMarkupId(true);
    totalCostSum.setOutputMarkupId(true);

    countryDropDown.add(
            new AjaxFormComponentUpdatingBehavior("onchange") {
                @Override
                protected void onUpdate(AjaxRequestTarget target) {
                    target.add(hotelDropDown);
                    target.add(tourDropDown);

                    String cost = getCost(countries, selectedOption);
                    countryCostModel.setObject(cost);
                    hotelCostModel.setObject(getCost(hotels, selectedHotel));
```

```
                tourCostModel.setObject(getCost(tours, selectedTour));
                totalCostSumModel.setObject(getTotalCostSum());

                target.add(countryCost);
                target.add(hotelCost);
                target.add(tourCost);
                target.add(totalCostSum);
            }
        }
    );
    hotelDropDown.add(
        new AjaxFormComponentUpdatingBehavior("onchange") {
            @Override
            protected void onUpdate(AjaxRequestTarget target) {
                hotelCostModel.setObject(getCost(hotels, selectedHotel));
                totalCostSumModel.setObject(getTotalCostSum());
                target.add(hotelCost);
                target.add(totalCostSum);
            }
        });
    tourDropDown.add(
        new AjaxFormComponentUpdatingBehavior("onchange") {
            @Override
            protected void onUpdate(AjaxRequestTarget target) {
                tourCostModel.setObject(getCost(tours, selectedTour));
                totalCostSumModel.setObject(getTotalCostSum());
                target.add(tourCost);
                target.add(totalCostSum);
            }
        });
    add(new Label("feedBack", feedBack));
    add(errorPanel);
    add(username);
    add(email);
    add(countryDropDown);
    add(hotelDropDown);
    add(tourDropDown);
    add(countryCost);
    add(hotelCost);
    add(tourCost);
    add(totalCostSum);

}
```

```
public String getSelectedOption() {
   return selectedOption;
}

public void setSelectedOption(String selectedOption) {
   this.selectedOption = selectedOption;
}

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

public String getSelectedHotel() {
   return selectedHotel;
}

public void setSelectedHotel(String selectedHotel) {
   this.selectedHotel = selectedHotel;
}

public String getSelectedTour() {
   return selectedTour;
}

public void setSelectedTour(String selectedTour) {
   this.selectedTour = selectedTour;
}

public String getCalculatedTotalCostSum() {
   return calculatedTotalCostSum;
}

public void setCalculatedTotalCostSum(String calculatedTotalCostSum) {
   this.calculatedTotalCostSum = calculatedTotalCostSum;
}
final PageParameters pageParams = new PageParameters();

@Override
public final void onSubmit() {
   if (AccountDAO.checkIsUserRegistered(username.getModelObject())) {
      orderDao.addNewOrder(getSelectedOption(),
            getSelectedHotel(),
            getSelectedOption(),
            username.getModelObject(),
```

```
                getCalculatedTotalCostSum(),
                selected);
            pageParams.add("feedBack", "Order has been placed !");
            setResponsePage(new CreateOrderForUser(pageParams));
        }
        if (username.getModelObject() == null || username.getModelObject() == "") {
            error("EnterUserName");
        }

        if (!AccountDAO.checkIsUserRegistered(username.getModelObject()) &&
(!(email.getModelObject() == "") || !(email.getModelObject() == null))) {
            AccountDAO.addNewAccount(username.getModelObject(), "123qwe");
            orderDao.addNewOrder(getSelectedOption(),
                getSelectedHotel(),
                getSelectedOption(),
                username.getModelObject(),
                getCalculatedTotalCostSum(),
                selected);
            setResponsePage(CreateOrderForUser.class);
        }
    }

    private List<String> getNames(List<OrderObject> orderObjects, String criteria) {
        List<String> pickedList = new ArrayList<String>();
        for (OrderObject orderObject : orderObjects) {
            if (orderObject.getCountry().equals(criteria)) {
                pickedList.add(orderObject.getName());
            }
        }
        return pickedList;
    }

    private String getCost(List<OrderObject> orderObjects, String name) {
        Double cost = 0.0;
        for (OrderObject object : orderObjects) {
            if (object.getName().equals(name)) {
                cost = object.getCost();
            }
        }
        return cost.toString();
    }

    private String getTotalCostSum() {
```

```
            Double hotelCost = Double.valueOf(getCost(hotels, selectedHotel));
            Double countryCost = Double.valueOf(getCost(countries, selectedOption));
            Double tourCost = Double.valueOf(getCost(tours, selectedTour));
            Double totalCost = hotelCost + countryCost + tourCost;
            return totalCost.toString();
        }

    }
}
```

*Класс DeleteOrder*

```
package tour.admin.functions;

import java.util.List;
import tour.menupanel.MenuPanel;
import org.apache.wicket.request.mapper.parameter.PageParameters;
import org.apache.wicket.markup.html.WebPage;
import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.markup.html.form.Button;
import org.apache.wicket.markup.html.form.Form;
import org.apache.wicket.markup.html.list.ListItem;
import org.apache.wicket.markup.html.list.ListView;
import org.apache.wicket.model.PropertyModel;
import tour.header.HeaderPanel;
import tour.order.OrderDao;
import tour.order.models.Order;

public final class DeleteOrder extends WebPage {

    private String feedBack;

    public DeleteOrder() {
        super();
    }

    public DeleteOrder(final PageParameters params) {
        feedBack = params.get("feedBack").toString();
        add(new HeaderPanel("headerPanel"));
        add(new MenuPanel("menuPanel"));

        OrderDao orderDao = new OrderDao();
        final PageParameters pageParams = new PageParameters();
        List<Order> orders = OrderDao.getOrdersForUser(params.get("userName").toString());
        Form form = new Form("form");
        form.add(new Label("feedBack", feedBack));
```

```
        form.add(new ListView<Order>("orders", orders) {
            @Override
            protected void populateItem(final ListItem<Order> li) {
                final PropertyModel buttonId = new PropertyModel(li.getModel(), "getId");
                li.add(new Label("country", new PropertyModel(li.getModel(), "getCountry")));
                li.add(new Label("hotel", new PropertyModel(li.getModel(), "getHotel")));
                li.add(new Label("tour", new PropertyModel(li.getModel(), "getTour")));
                li.add(new Label("user", new PropertyModel(li.getModel(), "getUser")));
                li.add(new Label("cost", new PropertyModel(li.getModel(), "getCost")));
                li.add(new Button("deleteButton") {
                    @Override
                    public void onSubmit() {
                        String id = buttonId.getObject().toString();
                        OrderDao.removeOrderById(id);
                        pageParams.add("userName", params.get("userName").toString());
                        pageParams.add("feedBack", "Deleted !");
                        setResponsePage(new DeleteOrder(pageParams));
                    }
                });
            }
        });
        add(form);
    }
}
```

*Класс SearchOrders*

```
package tour.admin.functions;

import tour.menupanel.MenuPanel;
import org.apache.wicket.request.mapper.parameter.PageParameters;
import org.apache.wicket.markup.html.WebPage;
import org.apache.wicket.markup.html.form.Button;
import org.apache.wicket.markup.html.form.Form;
import org.apache.wicket.markup.html.form.TextField;
import org.apache.wicket.markup.html.link.BookmarkablePageLink;
import org.apache.wicket.model.Model;
import tour.header.HeaderPanel;

public final class SearchOrders extends WebPage {

    private TextField<String> username;
    private Form form;
    private Button searchButton;

    public SearchOrders() {
```

```
        super();
        add(new HeaderPanel("headerPanel"));
        add(new MenuPanel("menuPanel"));
        form = new Form("form");
        searchButton = new Button("searchButton") {

            @Override
            public void onSubmit() {
                super.onSubmit(); //To change body of generated methods, choose Tools | Templates.
                PageParameters pageParameters = new PageParameters();
                        pageParameters.add("userName", username.getModelObject());
                        setResponsePage(DeleteOrder.class, pageParameters);
            }

        };
        username = new TextField<String>("usernameText", Model.of(""));
        form.add(username);
        form.add(searchButton);
        add(form);
    }

    public SearchOrders(PageParameters params) {
    }
}
```

*Класс DBConnection*

```
package tour.dbconnection;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class DBConnection {

    public DBConnection() {
    }

    public static Statement getStatement() {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con = DriverManager.getConnection("jdbc:mysql://localhost/test", "root", "root");
            Statement stmt = (Statement) con.createStatement();
            return stmt;
        } catch (Exception e) {
```

```
            return null;
        }
    }

    public static void executeUpdate(String query) {
        try {
            Statement stmt = getStatement();
            stmt.executeUpdate(query);
        } catch (Exception e) {
        }
    }

    public static ResultSet executeQuery(String query) {
        try {
            Statement stmt = getStatement();
            return stmt.executeQuery(query);
        } catch (Exception e) {
            return null;
        }
    }
}
```

*Класс* HeaderPanel

```
package tour.header;

import org.apache.wicket.markup.html.panel.Panel;

public final class HeaderPanel extends Panel {

    public HeaderPanel(String id) {
        super(id);
    }
}
```

*Класс MainPage*

```
package tour.main;

import java.util.ArrayList;
import java.util.List;
import tour.session.AuthenticatedWebPage;
import tour.menupanel.MenuPanel;
import tour.session.SignInSession;
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.Button;
```

```
import org.apache.wicket.markup.html.form.Form;
import org.apache.wicket.markup.html.link.BookmarkablePageLink;
import org.apache.wicket.markup.html.list.ListItem;
import org.apache.wicket.markup.html.list.ListView;
import org.apache.wicket.model.PropertyModel;
import org.apache.wicket.request.mapper.parameter.PageParameters;
import tour.header.HeaderPanel;
import tour.order.ComfirmationPage;
import tour.order.OrderPage;
import tour.order.models.Order;

public final class MainPage extends WebPage implements AuthenticatedWebPage {

   public MainPage() {
      final PageParameters pageParams = new PageParameters();
      add(new HeaderPanel("headerPanel"));
      add(new MenuPanel("menuPanel"));

      Form form = new Form("form");
      List<Order> orders = new ArrayList();
      orders.add(new Order("Zimbvabva", "Jellki", "River side Cruse", ((SignInSession)
Session.get()).getUser().toString(), "200"));
      orders.add(new Order("America", "Washington", "Airplane Ride", ((SignInSession)
Session.get()).getUser().toString(), "1200"));
      orders.add(new Order("Spain", "Madrid", "Run with the bulls", ((SignInSession)
Session.get()).getUser().toString(), "2200"));
      orders.add(new Order("China", "Hong-Kong", "Bus Ride", ((SignInSession)
Session.get()).getUser().toString(), "400"));
      orders.add(new Order("Latvia", "Daugavpils", "Wall climing", ((SignInSession)
Session.get()).getUser().toString(), "550"));
      form.add(new ListView<Order>("orders", orders) {
         @Override
         protected void populateItem(final ListItem<Order> li) {
            li.add(new Label("country", new PropertyModel(li.getModel(), "getCountry")));
            li.add(new Label("hotel", new PropertyModel(li.getModel(), "getHotel")));
            li.add(new Label("tour", new PropertyModel(li.getModel(), "getTour")));
            li.add(new Label("cost", new PropertyModel(li.getModel(), "getCost")));
            li.add(new Button("hotOrderButton") {
               @Override
               public void onSubmit() {
                  pageParams.add("country", new PropertyModel(li.getModel(), "getCountry").getObject());
                  pageParams.add("hotel", new PropertyModel(li.getModel(), "getHotel").getObject());
                  pageParams.add("tour", new PropertyModel(li.getModel(), "getTour").getObject());
                  pageParams.add("user", ((SignInSession) Session.get()).getUser());
```

```
                pageParams.add("cost", new PropertyModel(li.getModel(), "getCost").getObject());
                setResponsePage(new ComfirmationPage(pageParams));
            }
        });
    }
  });
  add(form);
 }
}
```

*Класс MenuPanel*

```
package tour.menupanel;

import tour.account.AccountDAO;
import tour.main.MainPage;
import tour.session.SignInSession;
import tour.signinout.SignOutPage;
import org.apache.wicket.Session;
import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.markup.html.link.Link;
import org.apache.wicket.markup.html.panel.Panel;
import tour.profile.Profile;
import tour.admin.functions.CreateOrderForUser;
import tour.admin.functions.SearchOrders;
import tour.order.OrderPage;
import tour.profile.history.UserOrderHistory;

public class MenuPanel extends Panel {

 public MenuPanel(String id) {
  super(id);

  add(new Link("homePage") {
   @Override
   public void onClick() {
    setResponsePage(MainPage.class);
   }
  });

  add(new Link("orderPage") {
   @Override
   public void onClick() {
    setResponsePage(OrderPage.class);
   }
  });
```

```
add(new Link("historyPage") {
  @Override
  public void onClick() {
     setResponsePage(UserOrderHistory.class);
  }
});

add(new Label("welcome", "Welcome, "));
add(new Link("profileLink") {
   @Override
   public void onClick() {
      setResponsePage(Profile.class);
   }
}.add(new Label("name", ((SignInSession) Session.get()).getUser().toString())));


Link adminMenu = new Link("adminMenu") {
   @Override
   public void onClick() {

   }
};

add(new Link("SearchOrders") {
      @Override
      public void onClick() {
         setResponsePage(SearchOrders.class);
      }
});

add(new Link("CreateOrderForUser") {
      @Override
      public void onClick() {
         setResponsePage(CreateOrderForUser.class);
      }
});

if (AccountDAO.checkIsAdmin(((SignInSession) Session.get()).getUser().toString())) {
   adminMenu.setVisible(Boolean.FALSE);

}

add(adminMenu);
```

```
        add(new Link("exit") {
            @Override
            public void onClick() {
                setResponsePage(SignOutPage.class);
            }
        });
    }
}
```

*Класс ConfirmationPage*

```
package tour.order;

import tour.main.MainPage;
import tour.menupanel.MenuPanel;
import org.apache.wicket.request.mapper.parameter.PageParameters;
import org.apache.wicket.markup.html.WebPage;
import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.markup.html.form.Button;
import org.apache.wicket.markup.html.form.Form;
import org.apache.wicket.behavior.* ;
import tour.header.HeaderPanel;


public final class ComfirmationPage extends WebPage {

    private OrderDao orderDao;

    public ComfirmationPage() {
        super();
    }

    public ComfirmationPage(final PageParameters params) {
        orderDao = new OrderDao();
        add(new MenuPanel("menuPanel"));
        add(new HeaderPanel("headerPanel"));
        Form form = new Form("form");
        form.add(new Label("country", params.get("country").toString()));
        form.add(new Label("hotel", params.get("hotel").toString()));
        form.add(new Label("tour", params.get("tour").toString()));
        form.add(new Label("user", params.get("user").toString()));
        form.add(new Label("cost", params.get("cost").toString()));
        Button confirmButton = new Button("confirmButton") {
            @Override
            public void onSubmit() {
```

```
            orderDao.addNewOrder(params.get("country").toString(),
                    params.get("hotel").toString(),
                    params.get("tour").toString(),
                    params.get("user").toString(),
                    params.get("cost").toString(),
                    "Card");
            setResponsePage(MainPage.class);
        }
    };
    form.add(confirmButton);

    add(form);
  }
}
```

*Класс OrderDao*

```
package tour.order;

import tour.order.models.Hotel;
import tour.order.models.Country;
import tour.order.models.OrderObject;
import tour.order.models.Tour;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.List;
import tour.dbconnection.DBConnection;
import tour.order.models.Order;

public class OrderDao {

  public static void removeOrderById(String id) {
    String insert = "delete from orders where idorders = '" + id + "';";
    DBConnection.executeUpdate(insert);
  }

  public void addNewOrder(String country, String hotel, String tour, String user, String cost, String
payment) {
    String delete = "INSERT INTO orders(country,hotel,tour,user,cost,payment) VALUES "
        + "('" + country + "','" + hotel + "','" + tour + "','" + user + "','" + cost + "','"+payment+"')";
    DBConnection.executeUpdate(delete);
  }

  public static List<Order> getOrdersForUser(String user){
    try {
```

```
        List<Order> list = new ArrayList<Order>();
        String select = "SELECT * FROM orders where user = '"+user+"';";
        ResultSet query = DBConnection.executeQuery(select);
        while (query.next()) {
            list.add(new Order(query.getString("country"),
                    query.getString("hotel"),
                    query.getString("tour"),
                    query.getString("user"),
                    query.getString("cost"),
                    query.getInt("idorders")));
        }
        return list;
    } catch (Exception e) {
        return null;
    }
}

public static List<Order> getOrders() {
    try {
        List<Order> list = new ArrayList<Order>();
        String select = "SELECT * FROM orders";
        ResultSet query = DBConnection.executeQuery(select);
        while (query.next()) {
            list.add(new Order(query.getString("country"),
                    query.getString("hotel"),
                    query.getString("tour"),
                    query.getString("user"),
                    query.getString("cost"),
                    query.getInt("idorders")));
        }
        return list;
    } catch (Exception e) {
        return null;
    }
}

public List<OrderObject> getCountries() {
    try {
        List<OrderObject> list = new ArrayList<OrderObject>();
        String select = "SELECT * FROM countrys";
        ResultSet query = DBConnection.executeQuery(select);
        while (query.next()) {
            list.add(new Country(query.getString("name"), query.getDouble("cost")));
        }
```

```
            return list;
        } catch (Exception e) {
            return null;
        }
    }

    public List<OrderObject> getHotels() {
        try {
            List<OrderObject> list = new ArrayList<OrderObject>();
            String select = "SELECT * FROM hotels";
            ResultSet query = DBConnection.executeQuery(select);
            while (query.next()) {
                list.add(new Hotel(query.getString("name"), query.getDouble("cost"),
query.getString("countryFk")));
            }
            return list;
        } catch (Exception e) {
            return null;
        }
    }

    public List<OrderObject> getTours() {
        try {
            List<OrderObject> list = new ArrayList<OrderObject>();
            String select = "SELECT * FROM tours";
            ResultSet query = DBConnection.executeQuery(select);
            while (query.next()) {
                list.add(new Tour(query.getString("name"), query.getDouble("cost"),
query.getString("countryFk")));
            }
            return list;
        } catch (Exception e) {
            return null;
        }
    }

}
```

*Класс OrderPage*

```
package tour.order;

import tour.order.models.OrderObject;
import tour.order.models.Order;
import java.util.ArrayList;
import java.util.Arrays;
```

```
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import tour.menupanel.MenuPanel;
import tour.session.SignInSession;
import org.apache.wicket.ajax.AjaxRequestTarget;
import org.apache.wicket.ajax.form.AjaxFormComponentUpdatingBehavior;
import org.apache.wicket.request.mapper.parameter.PageParameters;
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.form.TextField;
import org.apache.wicket.markup.html.link.BookmarkablePageLink;
import org.apache.wicket.model.AbstractReadOnlyModel;
import org.apache.wicket.model.IModel;
import org.apache.wicket.model.Model;
import org.apache.wicket.model.PropertyModel;
import tour.admin.functions.CreateOrderForUser;
import tour.header.HeaderPanel;

public final class OrderPage extends WebPage {

    private String feedBack;

    public OrderPage() {
        super();
        add(new HeaderPanel("headerPanel"));
        add(new MenuPanel("menuPanel"));
        Form<?> form = new OrderPageForm("OrderPageForm");
        add(form);
    }

    public OrderPage(PageParameters params) {
        super();
        feedBack = params.get("feedBack").toString();
        add(new HeaderPanel("headerPanel"));
        add(new MenuPanel("menuPanel"));
        Form<?> form = new OrderPageForm("OrderPageForm");
        add(form);
    }

    public final class OrderPageForm extends Form<Void> {
```

```
private Label countryCost;
private Label hotelCost;
private Label tourCost;
private TextField totalCostSum;
private OrderDao orderDao;
private final Map<String, List<String>> hotelOptions;
private final Map<String, List<String>> tourOptions;
private final DropDownChoice<String> countryDropDown;
private DropDownChoice<String> hotelDropDown;
private DropDownChoice<String> tourDropDown;
private List<OrderObject> hotels;
private List<OrderObject> tours;
private List<OrderObject> countries;
private String selectedOption;
private String selectedHotel;
private String selectedTour;
private String calculatedTotalCostSum;
private Order order;
private SignInSession session;
private Model<String> countryCostModel;
private Model<String> hotelCostModel;
private Model<String> tourCostModel;
private PropertyModel<String> totalCostSumModel;

public OrderPageForm(String id) {
    super(id);

    countryCostModel = Model.of("");
    hotelCostModel = Model.of("");
    tourCostModel = Model.of("");

    countryCost = new Label("countryCostLabel", countryCostModel);
    hotelCost = new Label("hotelCostLabel", hotelCostModel);
    tourCost = new Label("tourCostLabel", tourCostModel);

    session = getMySession();

    order = new Order();
    orderDao = new OrderDao();
    hotelOptions = new HashMap<String, List<String>>();
    tourOptions = new HashMap<String, List<String>>();

    countries = orderDao.getCountries();
```

```
        hotels = orderDao.getHotels();
        tours = orderDao.getTours();

        for (OrderObject country : countries) {
           hotelOptions.put(country.getName(), getNames(hotels, country.getName()));
           tourOptions.put(country.getName(), getNames(tours, country.getName()));
        }

        IModel<List<? extends String>> makeCountryChoises = new AbstractReadOnlyModel<List<?
extends String>>() {
           @Override
           public List<String> getObject() {
              return new ArrayList<String>(hotelOptions.keySet());
           }
        };

        IModel<List<? extends String>> modelTownChoices = new AbstractReadOnlyModel<List<?
extends String>>() {
           @Override
           public List<String> getObject() {
              List<String> models = hotelOptions.get(selectedOption);
              if (models == null) {
                 models = Collections.emptyList();
              }
              return models;
           }
        };

        IModel<List<? extends String>> modelTourChoices = new AbstractReadOnlyModel<List<?
extends String>>() {
           @Override
           public List<String> getObject() {
              List<String> models = tourOptions.get(selectedOption);
              if (models == null) {
                 models = Collections.emptyList();
              }
              return models;
           }
        };

        countryDropDown = new DropDownChoice<String>("countryDropDown",
              new PropertyModel<String>(this, "selectedOption"), makeCountryChoises);

        hotelDropDown = new DropDownChoice<String>("hotelDropDown",
```

```
        new PropertyModel<String>(this, "selectedHotel"), modelTownChoices);

tourDropDown = new DropDownChoice<String>("tourDropDown",
        new PropertyModel<String>(this, "selectedTour"), modelTourChoices);
totalCostSumModel = new PropertyModel<String>(this, "calculatedTotalCostSum");
totalCostSum = new TextField("totalCostSumLabel", totalCostSumModel);

hotelDropDown.setOutputMarkupId(true);
tourDropDown.setOutputMarkupId(true);
countryCost.setOutputMarkupId(true);
hotelCost.setOutputMarkupId(true);
tourCost.setOutputMarkupId(true);
totalCostSum.setOutputMarkupId(true);

countryDropDown.add(
        new AjaxFormComponentUpdatingBehavior("onchange") {
            @Override
            protected void onUpdate(AjaxRequestTarget target) {
                target.add(hotelDropDown);
                target.add(tourDropDown);

                String cost = getCost(countries, selectedOption);
                countryCostModel.setObject(cost);
                hotelCostModel.setObject(getCost(hotels, selectedHotel));
                tourCostModel.setObject(getCost(tours, selectedTour));
                totalCostSumModel.setObject(getTotalCostSum());

                target.add(countryCost);
                target.add(hotelCost);
                target.add(tourCost);
                target.add(totalCostSum);
            }
        }
);
hotelDropDown.add(
        new AjaxFormComponentUpdatingBehavior("onchange") {
            @Override
            protected void onUpdate(AjaxRequestTarget target) {
                hotelCostModel.setObject(getCost(hotels, selectedHotel));
                totalCostSumModel.setObject(getTotalCostSum());
                target.add(hotelCost);
                target.add(totalCostSum);
            }
        });
```

```
    tourDropDown.add(
            new AjaxFormComponentUpdatingBehavior("onchange") {
                @Override
                protected void onUpdate(AjaxRequestTarget target) {
                    tourCostModel.setObject(getCost(tours, selectedTour));
                    totalCostSumModel.setObject(getTotalCostSum());
                    target.add(tourCost);
                    target.add(totalCostSum);
                }
            });

    add(new Label("feedBack", feedBack));
    add(countryDropDown);
    add(hotelDropDown);
    add(tourDropDown);
    add(countryCost);
    add(hotelCost);
    add(tourCost);
    add(totalCostSum);

}

public String getSelectedOption() {
    return selectedOption;
}

public void setSelectedOption(String selectedOption) {
    this.selectedOption = selectedOption;
}

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

public String getSelectedHotel() {
    return selectedHotel;
}

public void setSelectedHotel(String selectedHotel) {
    this.selectedHotel = selectedHotel;
}

public String getSelectedTour() {
    return selectedTour;
```

```
}

public void setSelectedTour(String selectedTour) {
   this.selectedTour = selectedTour;
}

public String getCalculatedTotalCostSum() {
   return calculatedTotalCostSum;
}

public void setCalculatedTotalCostSum(String calculatedTotalCostSum) {
   this.calculatedTotalCostSum = calculatedTotalCostSum;
}

final PageParameters pageParams = new PageParameters();

@Override
public final void onSubmit() {
   orderDao.addNewOrder(getSelectedOption(),
         getSelectedHotel(),
         getSelectedTour(),
         session.getUser(),
         getTotalCostSum(),
         "Card");
   pageParams.add("feedBack", "Order has been placed !");
   setResponsePage(new OrderPage(pageParams));
}

private List<String> getNames(List<OrderObject> orderObjects, String criteria) {
   List<String> pickedList = new ArrayList<String>();
   for (OrderObject orderObject : orderObjects) {
      if (orderObject.getCountry().equals(criteria)) {
         pickedList.add(orderObject.getName());
      }
   }
   return pickedList;
}

private String getCost(List<OrderObject> orderObjects, String name) {
   Double cost = 0.0;
   for (OrderObject object : orderObjects) {
      if (object.getName().equals(name)) {
         cost = object.getCost();
      }
```

```
            }
            return cost.toString();
        }

        private String getTotalCostSum() {

            Double hotelCost = Double.valueOf(getCost(hotels, selectedHotel));
            Double countryCost = Double.valueOf(getCost(countries, selectedOption));
            Double tourCost = Double.valueOf(getCost(tours, selectedTour));
            Double totalCost = hotelCost + countryCost + tourCost;
            return totalCost.toString();
        }

    }
}
```

*Класс Country*

```
package tour.order.models;

public class Country extends OrderObject{

    private String name;
    private Double cost;

    public Country(String name, Double cost) {
        this.name = name;
        this.cost = cost;
    }

    public String getName() {
        if (name == null) {
            return "";
        }
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Double getCost() {
        return cost;
    }

    public void setCost(Double cost) {
```

```
        this.cost = cost;
    }

    @Override
    public String getCountry() {
        return null;
    }


}
```

*Класс Hotel*

```
package tour.order.models;


public class Hotel extends OrderObject{

    private String name;
    private Double cost;
    private String country;

    public Hotel(String name, Double cost, String country) {
        this.name = name;
        this.cost = cost;
        this.country = country;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Double getCost() {
        return cost;
    }

    public void setCost(Double cost) {
        this.cost = cost;
    }

    public String getCountry() {
```

```
        return country;
    }

    public void setCountry(String country) {
        this.country = country;

}
```

*Класс Order*

```
package tour.order.models;

public class Order {

    private String country;
    private String hotel;
    private String tour;
    private String user;
    private String cost;
    private String payment;
    private Integer id;

    public Order() {
    }

    public Order(String country, String hotel, String tour, String user, String cost){
        this.country = country;
        this.hotel = hotel;
        this.tour = tour;
        this.user = user;
        this.cost = cost;
    }

    public Order(String country, String hotel, String tour) {
        this.country = country;
        this.hotel = hotel;
        this.tour = tour;
    }

    public Order(String country, String hotel, String tour, String user, String cost, Integer id) {
        this.country = country;
        this.hotel = hotel;
        this.tour = tour;
        this.user = user;
        this.cost = cost;
        this.id= id;
```

```
}


public String getPayment() {
   return payment;
}

public void setPayment(String payment) {
   this.payment = payment;
}

public void setId(Integer id){
   this.id = id;
}

public Integer getId(){
   return id;
}

public String getCountry() {
   return country == null ? "" : country;
}

public void setCountry(String country) {
   this.country = country;
}

public String getHotel() {
return hotel == null ? "" : hotel;
}

public void setHotel(String hotel) {
   this.hotel = hotel;
}

public String getTour() {
   return tour == null ? "" : tour;
}

public void setTour(String tour) {
   this.tour = tour;
}

public String getUser() {
```

```
        return user == null ? "" : user;
    }

    public void setUser(String user) {
        this.user = user;
    }

    public String getCost() {
        return cost == null ? "" : cost;
    }

    public void setCost(String cost) {
        this.cost = cost;
    }

    public String getOrder(){
        return "    " + getCountry()+
                "    " + getHotel()+
                "    " + getTour()+
                "    " + getUser()+
                "    " + getCost();
    }
}
```

*Класс OrderObject*

```
package tour.order.models;
public abstract class OrderObject {
    public abstract String getName();
    public abstract String getCountry();
    public abstract Double getCost();
}
```

*Класс Tour*

```
package tour.order.models;

public class Tour extends OrderObject{

    private String name;
    private Double cost;
    private String country;

    public Tour(String name, Double cost, String country) {
        this.name = name;
        this.cost = cost;
```

```
        this.country = country;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Double getCost() {
        return cost;
    }

    public void setCost(Double cost) {
        this.cost = cost;
    }

    public String getCountry() {
        return country;
    }

    public void setCountry(String country) {
        this.country = country;
    }


}
```

*Класс Profile*

```
package tour.profile;

import java.util.List;
import tour.account.Account;
import tour.account.AccountDAO;
import tour.menupanel.MenuPanel;
import tour.session.SignInSession;
import org.apache.wicket.Session;
import org.apache.wicket.feedback.ComponentFeedbackMessageFilter;
import org.apache.wicket.markup.html.WebPage;
import org.apache.wicket.markup.html.form.Button;
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.panel.FeedbackPanel;
import org.apache.wicket.model.PropertyModel;
import tour.header.HeaderPanel;

public final class Profile extends WebPage {

   private final Account acc = new Account();
   private final AccountDAO aO = new AccountDAO();
   private String name;
   private String surname;
   private String age;
   private String persCode;

   public Profile() {
      add(new HeaderPanel("headerPanel"));
      add(new MenuPanel("menuPanel"));
      List<Account> accounts = aO.getAccountList();
      SignInSession session = getMySession();
      Account currentAccount = getAccount(accounts, session.getUser());

      setName(currentAccount.getName());
      setSurname(currentAccount.getSurname());
      setAge(currentAccount.getAge());
      setPersCode(currentAccount.getPersCode());



      final TextField yourName = new TextField("yourName", new PropertyModel<String>(this,
"name"));
      final TextField yourSurname = new TextField("yourSurname", new PropertyModel<String>(this,
"surname"));
      final TextField yourAge = new TextField("yourAge", new PropertyModel<String>(this, "age"));
      final TextField yourPerCode = new TextField("yourPerCode", new PropertyModel<String>(this,
"persCode"));

      Button saveButton = new Button("saveChanges") {
         @Override
         public void onSubmit() {
            String login = ((SignInSession) Session.get()).getUser().toString();
            String name = yourName.getValue();
            String surname = yourSurname.getValue();
            String age = yourAge.getValue();
            String persCode = yourPerCode.getValue();
            AccountDAO.updateAccount(login,name,surname,age,persCode);
            String msg = getString("profileChanged", null, "Profile data changed!");
```

```
            success(msg);
        }
    };

    Form profileForm = new Form("profileForm");

    final PasswordTextField oldPassword = new PasswordTextField("oldPassword", new
PropertyModel<String>(acc, "password"));
    final PasswordTextField newPassword = new PasswordTextField("newPassword", new
PropertyModel<String>(acc, "newPassword"));
    final PasswordTextField repeatNewPassword = new PasswordTextField("repeatNewPassword", new
PropertyModel<String>(acc, "repeatNewPassword"));

    Button changeButton = new Button("savePw") {
        @Override
        public void onSubmit() {
            String login = ((SignInSession) Session.get()).getUser().toString();
            String oldpassword = oldPassword.getValue();
            String newpassword = newPassword.getValue();
            String repeatnewpassword = repeatNewPassword.getValue();

            if(AccountDAO.checkOldPassword(login, oldpassword)==true)
            {
                if(newpassword.equals(repeatnewpassword))
                {
                    AccountDAO.changePassword(login, newpassword);
                    String msg = getString("passwordChanged", null, "Password changed!");
                    success(msg);
                }
                else
                {
                    String errmsg = getString("repeatPasswordError", null, "'Repeat new password' do not
match 'New Password'");
                    error(errmsg);
                }
            }
            else
            {
                String errmsg = getString("oldPasswordError", null, "Wrong old password.");
                error(errmsg);
            }
            }
    };
```

```
    Form passwordForm = new Form("passwordForm");
    FeedbackPanel successP = new FeedbackPanel("successP");
    profileForm.add(yourName, yourSurname, yourAge, yourPerCode, saveButton, successP);
    passwordForm.add(oldPassword, newPassword, repeatNewPassword, changeButton);
    add(profileForm, passwordForm);
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getSurname() {
    return surname;
}

public void setSurname(String surname) {
    this.surname = surname;
}

public String getAge() {
    return age;
}

public void setAge(String age) {
    this.age = age;
}

public String getPersCode() {
    return persCode;
}

public void setPersCode(String persCode) {
    this.persCode = persCode;
}

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

private Account getAccount(List<Account> accounts, String user) {
```

```
        Account account = new Account();
        for (Account ac : accounts) {
            if (ac.getLogin().equals(user)) {
                account = ac;
            }
        }
        return account;
    }
}
```

*Класс* UserOrderHistory

```
package tour.profile.history;

import java.util.List;
import tour.menupanel.MenuPanel;
import tour.session.SignInSession;
import org.apache.wicket.Session;
import org.apache.wicket.request.mapper.parameter.PageParameters;
import org.apache.wicket.markup.html.WebPage;
import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.markup.html.form.Form;
import org.apache.wicket.markup.html.list.ListItem;
import org.apache.wicket.markup.html.list.ListView;
import org.apache.wicket.model.PropertyModel;
import tour.header.HeaderPanel;
import tour.order.OrderDao;
import tour.order.models.Order;

public final class UserOrderHistory extends WebPage {

    public UserOrderHistory() {
        super();
        add(new HeaderPanel("headerPanel"));
        add(new MenuPanel("menuPanel"));
        List<Order> orders = OrderDao.getOrdersForUser(((SignInSession)
Session.get()).getUser().toString());
        Form form = new Form("form");
        form.add(new ListView<Order>("orders", orders) {
            @Override
            protected void populateItem(final ListItem<Order> li) {
                li.add(new Label("country", new PropertyModel(li.getModel(), "getCountry")));
                li.add(new Label("hotel", new PropertyModel(li.getModel(), "getHotel")));
                li.add(new Label("tour", new PropertyModel(li.getModel(), "getTour")));
                li.add(new Label("cost", new PropertyModel(li.getModel(), "getCost")));
            }
```

```
        });
        add(form);
    }

    public UserOrderHistory(PageParameters params) {

    }
}
```

*Класс RegistrationPage*

```
package tour.registration;

import tour.account.Account;
import tour.account.AccountDAO;
import tour.signinout.SignInPage;
import org.apache.wicket.markup.html.WebPage;
import org.apache.wicket.markup.html.form.Button;
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.panel.FeedbackPanel;
import org.apache.wicket.model.PropertyModel;

public class RegistrationPage extends WebPage {

    private Account acc = new Account();

    public RegistrationPage() {

        final TextField loginEnter = new TextField("loginEnter", new PropertyModel<String>(acc,
"login"));
        final PasswordTextField passwordEnter = new PasswordTextField("password", new
PropertyModel<String>(acc, "password"));
        final TextField nameEnter = new TextField("nameEnter", new PropertyModel<String>(acc,
"name"));
        final TextField surnameEnter = new TextField("surnameEnter", new PropertyModel<String>(acc,
"surname"));
        final TextField ageEnter = new TextField("ageEnter", new PropertyModel<String>(acc, "age"));
        final TextField perCodeEnter = new TextField("perCodeEnter", new PropertyModel<String>(acc,
"persCode"));

        FeedbackPanel error = new FeedbackPanel("error");
        Button createButton = new Button("Create") {
            @Override
            public void onSubmit() {
```

```
            AccountDAO accountDAO = new AccountDAO();

            String username = loginEnter.getValue();
            String password = passwordEnter.getValue();
            String name = nameEnter.getValue();
            String surname = surnameEnter.getValue();
            String age = ageEnter.getValue();
            String persCode = perCodeEnter.getValue();

            if (AccountDAO.checkLogin(username) == true) {
               error("This login alredy taken");
            } else {
               if ((username.equals("")) || (password.equals("")) || (name.equals("")) || (surname.equals("")) ||
(age.equals("")) || (persCode.equals(""))) {
                  error("Fill all required fields");
               } else {
                  AccountDAO.addNewAccount(username, password, name, surname, age, persCode);
                  setResponsePage(SignInPage.class);
               }
            }
         }
      };
      Form registerForm = new Form("registerForm");
      registerForm.add(error, loginEnter, passwordEnter, nameEnter, surnameEnter, ageEnter,
perCodeEnter, createButton);
      add(registerForm);
   }
}
```

*Класс SignInSession*

```
package tour.session;

import java.util.Iterator;
import tour.account.Account;
import tour.account.AccountDAO;
import org.apache.wicket.authroles.authentication.AuthenticatedWebSession;
import org.apache.wicket.authroles.authorization.strategies.role.Roles;
import org.apache.wicket.request.Request;

public class SignInSession extends AuthenticatedWebSession {

   private String userName;
   private AccountDAO accDAO = new AccountDAO();

   protected SignInSession(Request request) {
```

```
        super(request);
    }

    @Override
    public final boolean authenticate(final String username, final String password) {

        Iterator itr = accDAO.getAccountList().iterator();

            if (userName == null) {

                while(itr.hasNext()){
                    Account user = (Account) itr.next();
                if (user.getLogin().equalsIgnoreCase(username) &&
user.getPassword().equalsIgnoreCase(password)) {
                    userName = username;
                }}
            }

            return userName != null;

    }

    public String getUser() {
        return userName;
    }

    public void setUser(final String user) {
        this.userName = user;
    }

    @Override
    public Roles getRoles() {
        return null;
    }
}
```

*Класс TourApplication*

```
package tour.session;

import tour.main.MainPage;
import tour.signinout.SignInPage;
import org.apache.wicket.Component;
import org.apache.wicket.Page;
import org.apache.wicket.RestartResponseAtInterceptPageException;
import org.apache.wicket.Session;
```

```
import org.apache.wicket.authorization.Action;
import org.apache.wicket.authorization.IAuthorizationStrategy;
import org.apache.wicket.protocol.http.WebApplication;
import org.apache.wicket.request.Request;
import org.apache.wicket.request.Response;
import org.apache.wicket.request.component.IRequestableComponent;


public class TourApplication extends WebApplication {

   @Override
   public Class<? extends Page> getHomePage() {
      return MainPage.class;
   }

   @Override
   public Session newSession(Request request, Response response)
   {
      return new SignInSession(request);
   }

   @Override
   protected void init()
   {
      super.init();

      getSecuritySettings().setAuthorizationStrategy(new IAuthorizationStrategy()
      {
         @Override
         public boolean isActionAuthorized(Component component, Action action)
         {
            return true;
         }

         @Override
         public <T extends IRequestableComponent> boolean isInstantiationAuthorized(
            Class<T> componentClass)
         {
            if (AuthenticatedWebPage.class.isAssignableFrom(componentClass))
            {
               if (((SignInSession)Session.get()).isSignedIn())
               {
                  return true;
               }
```

```
                throw new RestartResponseAtInterceptPageException(SignInPage.class);
            }
            return true;
        }
    });
  }
}
```

*Класс SignInPage*

```
package tour.signinout;

import tour.main.MainPage;
import tour.registration.RegistrationPage;
import tour.session.SignInSession;
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;
import tour.header.HeaderPanel;

public final class SignInPage extends WebPage {

  public SignInPage() {
    add(new HeaderPanel("headerPanel"));
    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 TextField<String>(USERNAME, new PropertyModel<String>(properties,
USERNAME)));
      add(new PasswordTextField(PASSWORD, new PropertyModel<String>(properties,
PASSWORD)));
```

```
        add(new Link("regPage") {
            @Override
            public void onClick() {
                setResponsePage(RegistrationPage.class);
            }
        });
    }

    @Override
    public final void onSubmit() {
        SignInSession session = getMySession();
        if (session.signIn(getUsername(), getPassword())) {
            setResponsePage(MainPage.class);
        } else {
            String errmsg = getString("loginError", null, "Unable to sign you in");
            error(errmsg);
        }
    }

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

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

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

*Класс SignOutPage*

```
package tour.signinout;

import org.apache.wicket.markup.html.WebPage;
import tour.header.HeaderPanel;

public final class SignOutPage extends WebPage {

  public SignOutPage() {
    add(new HeaderPanel("headerPanel"));
    getSession().invalidate();
  }
```

}