Team Project

Easy Travel System

Liavontsi Brechka

300800345

Clinton Essien

300773051

Oleksandra Sypiachova

300855051

Object Oriented Software Engineering – 246 – 004

Professor Zoran Sarajlic

November 20, 2015

Content

Domain 3

Summary 3

Gantt chart (updated for B part) 4

Work Flows 4

1. Register customer 4

2. Plan Travel 5

3. Purchase Saved Package 7

4. Ask questions 9

5. Change Order 10

Use Case List 12

Use Case Diagrams 16

Register Customer Use Case Diagram 16

Plan Travel Use Case Diagram 17

Purchase Saved Package Use Case Diagram 18

Ask questions Use Case Diagram 19

Change Order Use Case Diagram 20

User Stories 21

1. As a customer on the website 21

2. As a traveller 21

Domain Class Diagram 22

List of Technology Tools for Software Development 23

Class Responsibility Collaboration Cards……………………………………………………………………………………………….24

Detailed System Sequence Diagrams 27

State Machine Diagrams 29

Design Class Diagram 31

# Domain

Travelling system (Easy travel) can be used by travelers for planning the trip thoroughly based on budget, destination, dates, and personal preferences with recommended staff to take in travelling. Selecting and purchasing travel packaged includes transport reservation (flights, train, car, bus), hotels and variety of accommodations, car rent and entertainment possibilities within budget range.

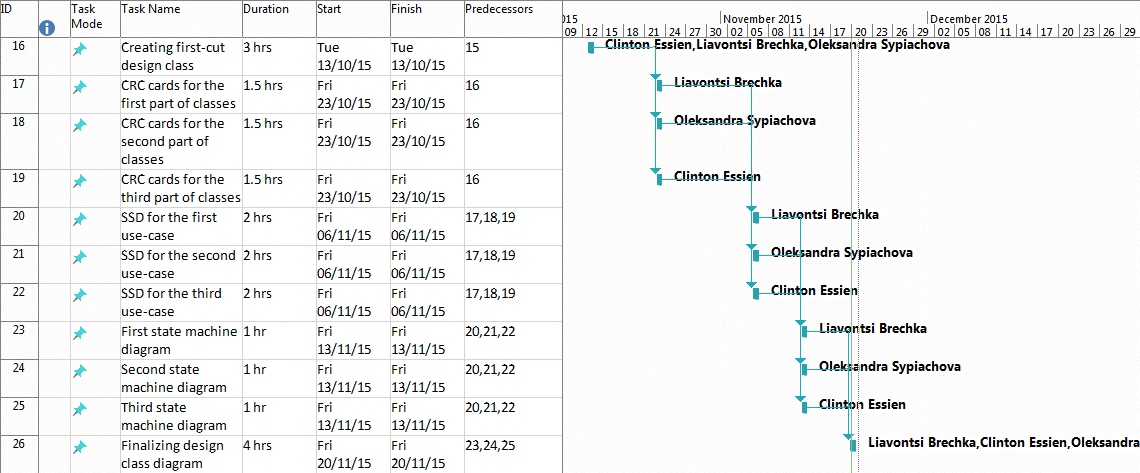
# Summary

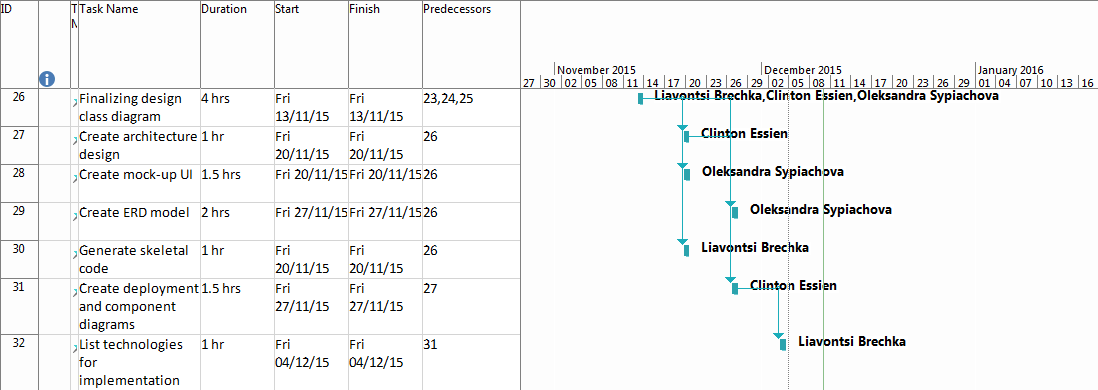
A customer (traveller) intends to plan his/her trip based on allocated budget and destination planned. She/he locates the system either through the website, mobile or computing application. If customer has been registered before, he/she would only need to authenticate their account. Else traveller will be requested to register as a new customer and at this point system records basic information input by user (first and last name, address, age, sex, preferences). The system receives registration form and then creates an account for new user after verification has been done. After account has been successfully created, customer then navigates to Travel Planning page, where he/she provides information about intended monetary budget, destination, accommodation preferences. System collaborates with partners to see if customer’s options are available, if yes, system validates those options, save them and then give feedback to customer. Customer could change options if it’s needed, then system would send new changes or updates to partners. If customer doesn’t change options he/she would proceed to checkout finalising the planning. Further traveller proceed with purchasing of saved package. Before payment, validation of saved options are made by system. After receiving of positive confirmation from partner’s system redirect customer to payment page with request to confirm payment amount, choose payment method and enter payment details. After validation of payment details system proceeds with payment transaction, receives payment confirmation from partner’s system. Then, system transfer saved package to order and sends order/payment details to customer. Customer can print details of both: saved package and bought order. Moreover customer can ask questions at any point of collaboration with system.

# Gantt chart

# 

# 



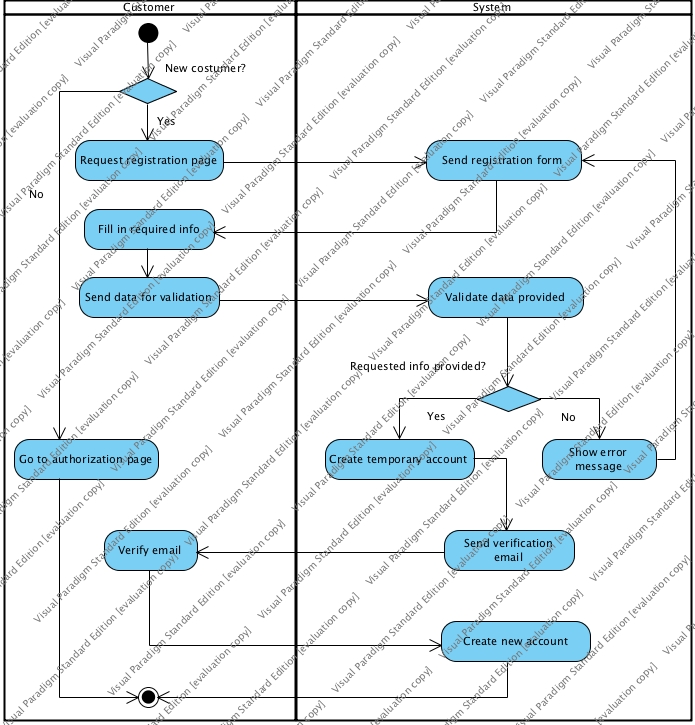


# Work Flows

## Register customer

* 1. If customer registered before, no registration needed and he can go to authorization page; if it is new, customer requires registration page.
  2. System send registration page and require information to create an account.
  3. Customer fills in required information.
  4. Customer sends information for validation.
  5. System validates provided information.
  6. If required information provided correctly, system creates temporary account for the customer; if information is not valid, system shows error message and sends registration form again.
  7. System sends email to verify account created.
  8. Customer verifies account via email.
  9. System deletes temporary account and creates new permanent one.

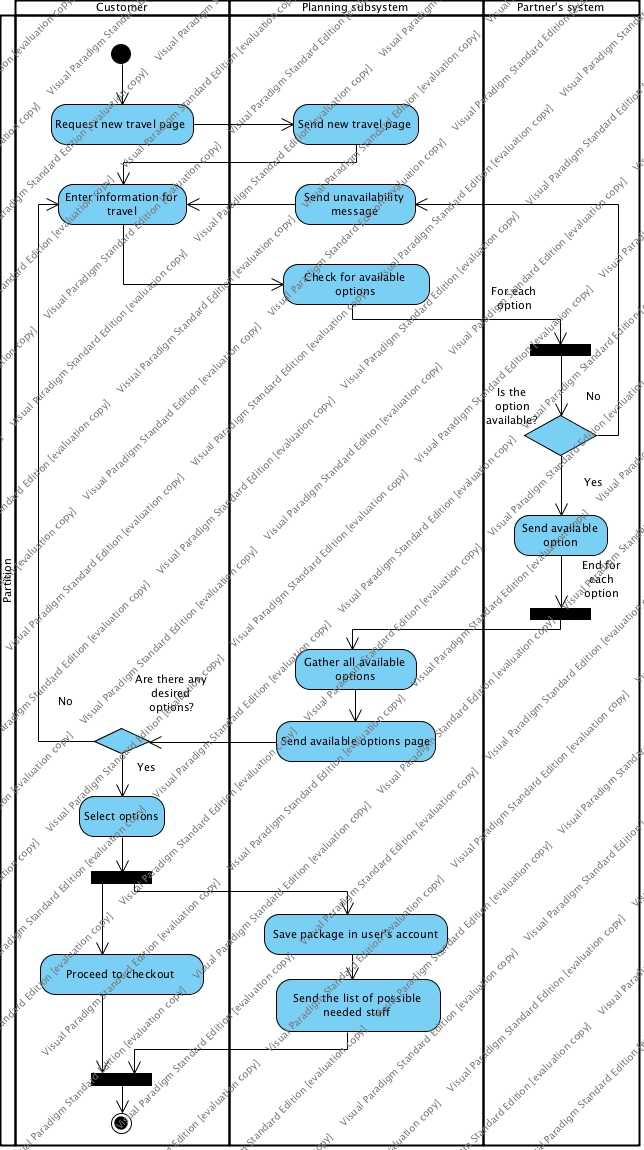
**Register Costumer Activity Diagram**



## Plan Travel

* 1. Customer requires page for new travel planning.
  2. System sends the page for new travel planning.
  3. Customer enters information about travel (e.g. destination, budget, date, etc.).
  4. System checks availability for each option in partners` systems for each option.
  5. If there is available option, partner`s system sends information about this option; if there is no desired option, system sends unavailable message for this option.
  6. System gathers all available options.
  7. System sends all available options to customer.
  8. If there are desired options, customer selects them to create a package; if any option is inappropriate, customer enters another information for travel.
     1. Customer proceeds to checkout.
     2. System saves generated package in customer`s account.
     3. System sends the list of possible items needed during a travel.

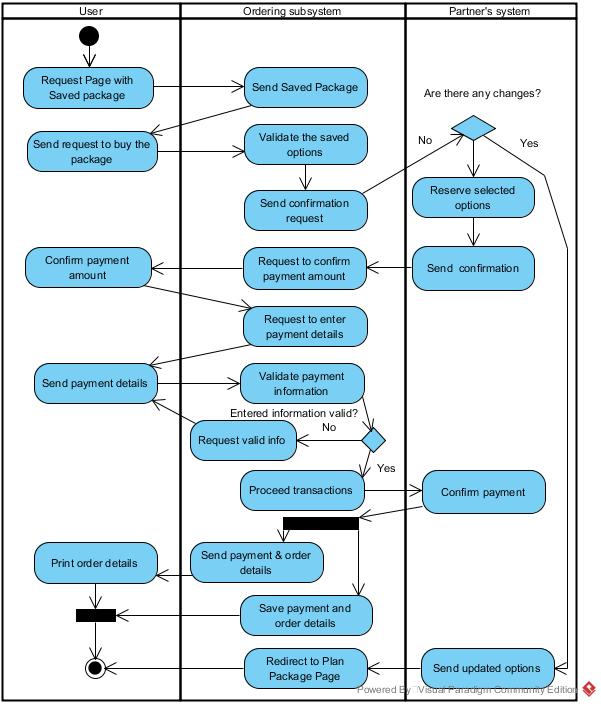
**Plan Travel Activity Diagram**



**3.Purchased Saved Package**

* 1. User requests page with saved package and after receiving it from system requests to buy this package
  2. Ordering subsystem (further here “system” ) validates saved options – (whether they are still up-to-date and reservations deadlines isn’t passed etc.) and send confirmation request to partners’ system
  3. If there are no changes partners’ system reserves the options and sends confirmation. If there are changes partners system send updated information to ordering subsystem and this subsystem redirects customer to the planning step by sending adjustment page.
  4. System propose to make actual payment:
     1. System send request to confirm payment amount
     2. User confirms payment amount
     3. System sends request to enter payment method and details
     4. User enters payment method and details
     5. System validates payment details and in case the details are valid, system proceed the transaction
  5. After confirmation of payment from partners system, System change the saved package – to order and sends the order and payment details to the user with parallel process of saving it

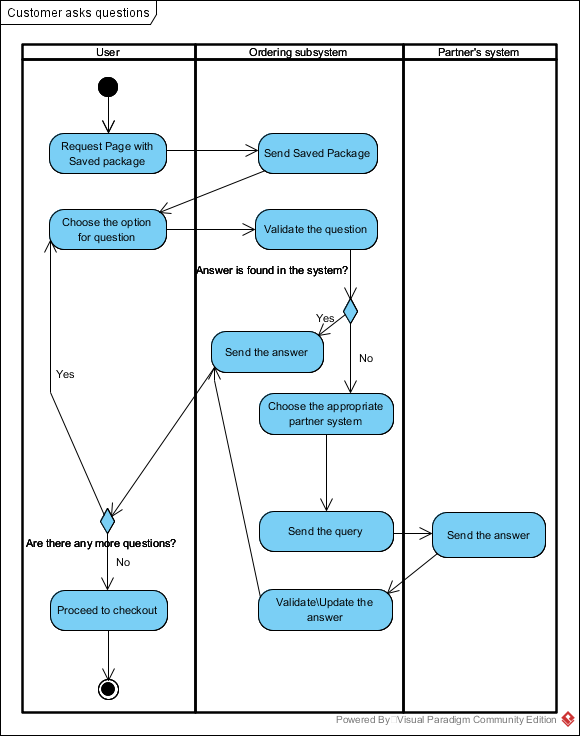
**Purchase Saved Package Activity Diagram**



**4.Ask questions**

* 1. User requests page with saved package
  2. System send page with saved package
  3. User choose the option about which he would like to ask the question (accommodation, flight/train/car etc.) and asks questions
  4. System validates the question.
  5. In case the answer is found in the system – it’s sent directly to user.
  6. In case there is no answer – system sends the query to the appropriate partner’s system
  7. Partner system sends answer to the system
  8. System validate/update answer and send it to the user

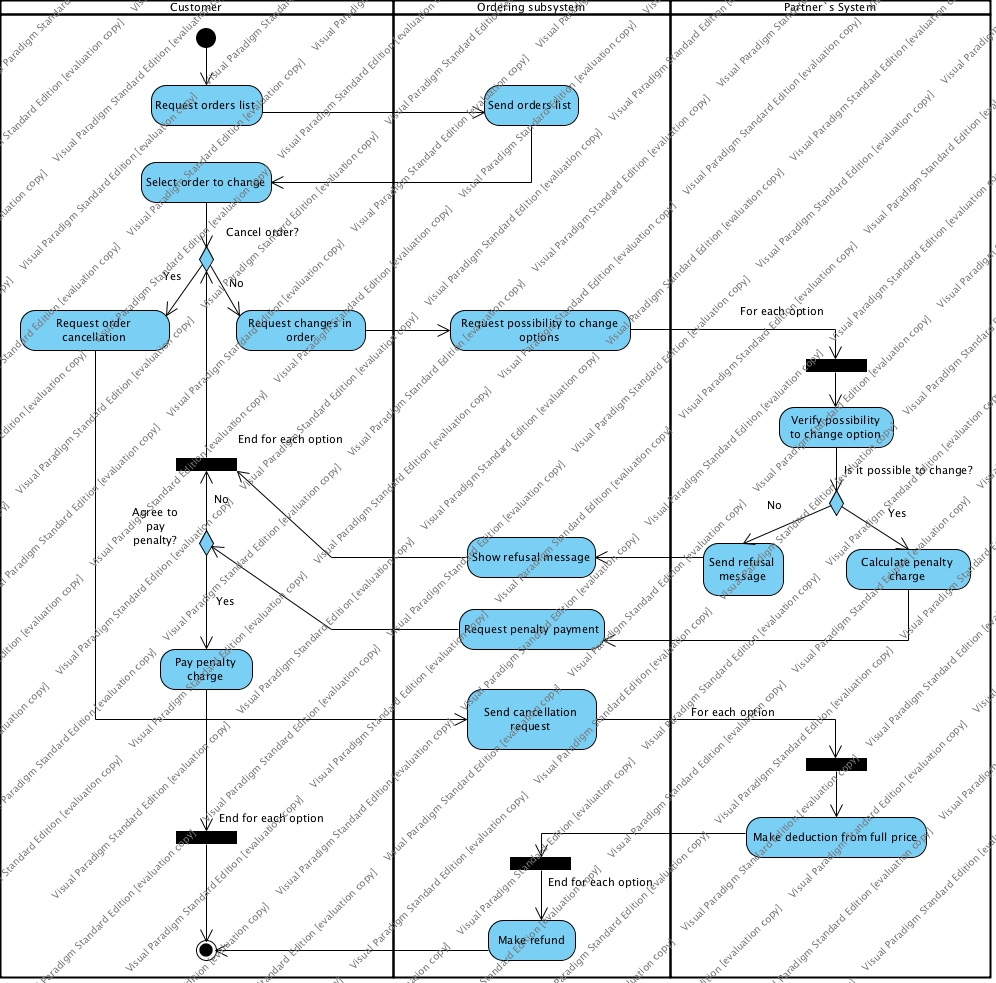
**Ask Questions Activity Diagram**



## 5. Change Order

1. Customer requires an orders list.
2. System sends the orders list.
3. Customer selects the order to change.
4. If customer wants to make some changes in order, he requests particular changes in order.
   1. System requests possibility to change options from partners.
   2. Partner’s system verifies possibility to change options for each of them.
   3. If it is possible to change, partner`s system calculate penalty charge.
      1. System requests a penalty payment.
      2. If customer agrees to pay penalty, he pays it; if he doesn`t want to pay penalties, customer may decide to cancel the order.
   4. If it is not possible to change option, partner`s system sends refusal message.
      1. System shows this message to customer.
      2. Customer decides to change other options or cancel the order.
5. If customer wants to cancel the order, he sends an order cancelation request
   1. System sends cancelation request to partners.
   2. Partner’s system makes a deduction from full price to refund for each option.
   3. System makes a refund to the customer.

**Change Order Activity Diagram**

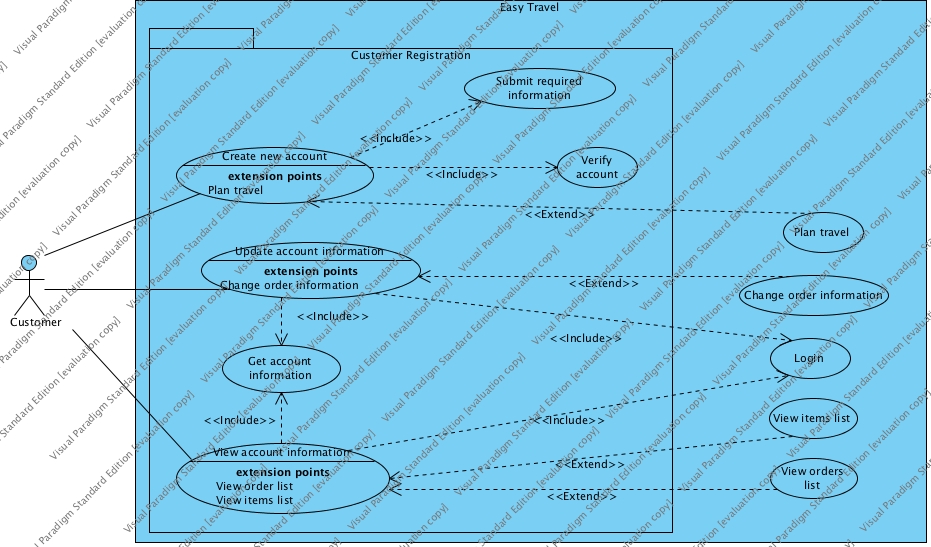


# Use Case List

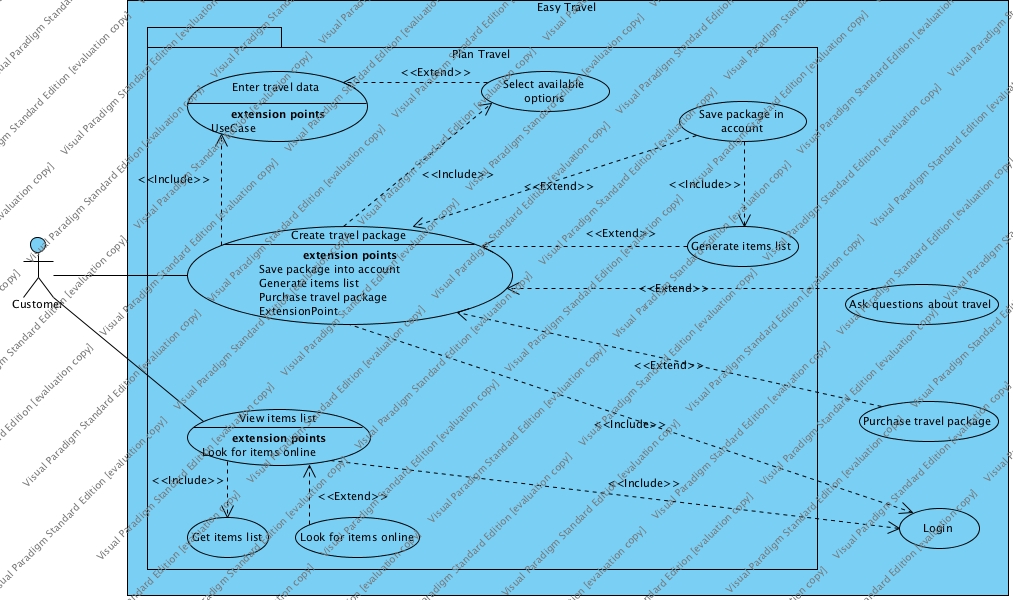
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **No.** | **Actor** | **Goal Use Case** | **<Include>** | **<Extend>** | **Generalization** | **Priority** | **Brief Description** |
| 1.1 | Customer | Create new account | * Submit required information * Verify account | * Plan travel |  | H | Customer creates new account for future system use |
| 1.2 | Customer | Update account information | * Login * Get account information | * Change order information |  | M | Customer updates his account info, which may be extended to change order info |
| 1.3 | Customer | Views account information | * Login * Gets account information | * View items list * View orders list |  | M | Customer views his account information and may proceed to items and orders lists. |
| 2.1 | Customer | Create travel package | * Login * Enter travel data * Select available options | * Save package in account * Generate items list * Ask questions about travel * Purchase travel package |  | H | Customer creates travel package that consists of desirable and available options. |
| 2.2 | Customer | View items list | * Login * Get items list |  |  | H | Customer view list of items that may be needed during the travel. |
| 3.1 | Customer | Purchase saved package |    Validate saved package   * Payment transaction |         Asks questions       Change/cancel order |  | H | User review saved package, check and confirm payment amount, enter payment details, proceed payment. |
| 3.2 | Customer | Review order (purchased package) |     Get order details |        Print order details          Receive notification of travelling arrangements    Ask questions |  | H | User requests for order (including payment) details if needed |
| 4.1 | Customer | Ask questions |         Review saved package |       Print answers |  | H | User asks questions |
| 4.2 | Customer | Save/Review answers |  |       Print answers |  | M | Customer saves answer received. If needed, customer reviews answers |
| 5.1 | Customer | Change order information | * Pay penalty charge * View orders list * Login |  |  | H | Customer makes changes in order by specifying other options that are available by paying particular penalty amount |
| 5.2 | Customer | Cancel order | * View orders list * Get refund |  |  | H | Customer cancels the order and gets refund for it. |

# Use Case Diagrams

## Register Customer Use Case Diagram



## Plan Travel Use Case Diagram



## Purchase Saved Package Use Case Diagram

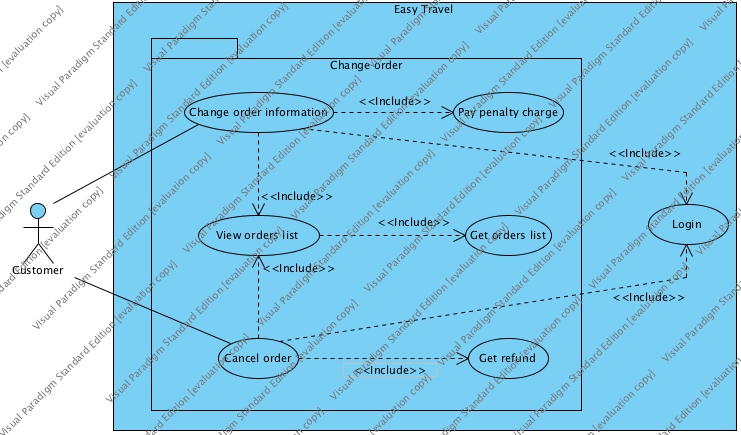
## C:\Users\Oleksandra\Documents\VPProjects\use case diagram purchase2.jpg

## 

## Ask questions Use Case Diagram

## C:\Users\Oleksandra\Documents\VPProjects\Use case diagram User asks questions about Package.jpg

## Change Order Use Case Diagram



# User Stories

## As a customer on the website

**I want to create account**

**So that I can buy travel packages, save them in account and see information about my orders**

**Acceptance criteria:**

1. Validate full name
2. Validate billing information
3. Validate email
4. Verify account via email

## As a traveller

**I want easily plan my new journey**

**So that I can spend my free time in exciting and organized way**

**Acceptance criteria:**

1. **Validate account existence**
2. **Accept user`s desirable options (destination, car lease, accommodation, budget, etc.)**
3. **Check available travel options in partners` systems**
4. **Save generated package in account**
5. **Generate a list of possible needed items**
6. **As a traveller**

**I want to purchase my travel package**

**in order to have final reservations.**  
  
**Acceptance criteria:**

1. **User can purchase previously saved travel package (order).**

**2. User can review, update and cancel order**

**3. User can chose different payment methods**

**4. User can’t pay for options that aren’t confirmed by partner’s system**

**5. User can cancel purchase before payment confirmation**

**6. User can’t make purchase if payment details are invalid**

1. **As a user**

**I want to have opportunity to ask questions at any step of collaboration with system**

**Acceptance criteria:**

1. **User can ask questions during:**

**1.1. Planning of travelling**

**1.2. Purchasing Package including payment**

**1.3. Changing/canceling the order  
2. User can review, update and delete saved answers**

**3. User should receive response from system within 60 seconds**

**4. If no answer in the system user should receive message with expected time of answer or options to ask help desk (online, phone)**

**5. As a user I want to receive notifications/alerts about due dates  
  
Acceptance criteria:**

1**. User receives notifications in case:**

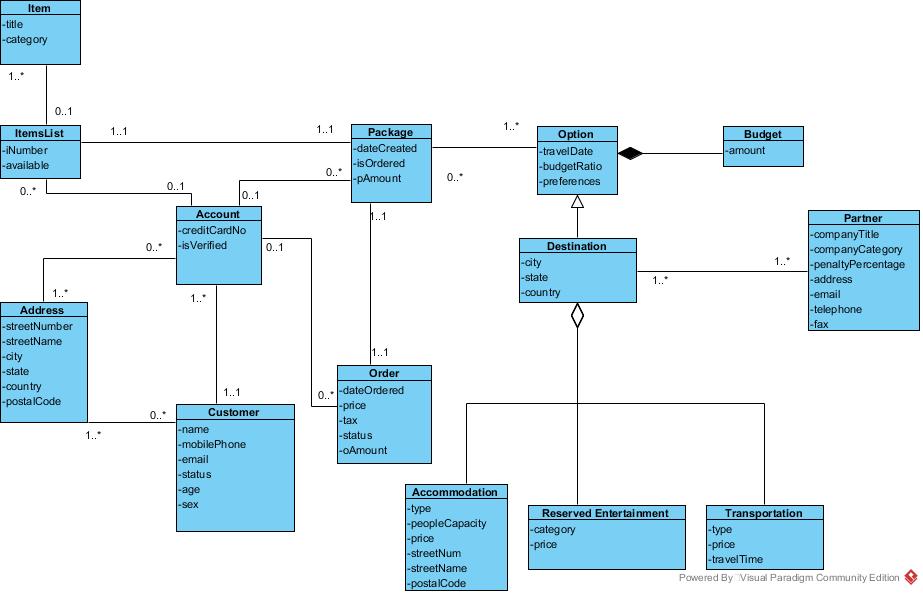
**1.1. 5, 3, 1 day before deadline for making purchase and final reservation**

**1.2. 5, 3, 1 day before actual flight (train, bus)/check in (hotel, accommodation)/rental car**

**1.3. 2, 1 day before the last possible day for making changes**

**2. User has possibility to set up notifications and to sign out from notifications**

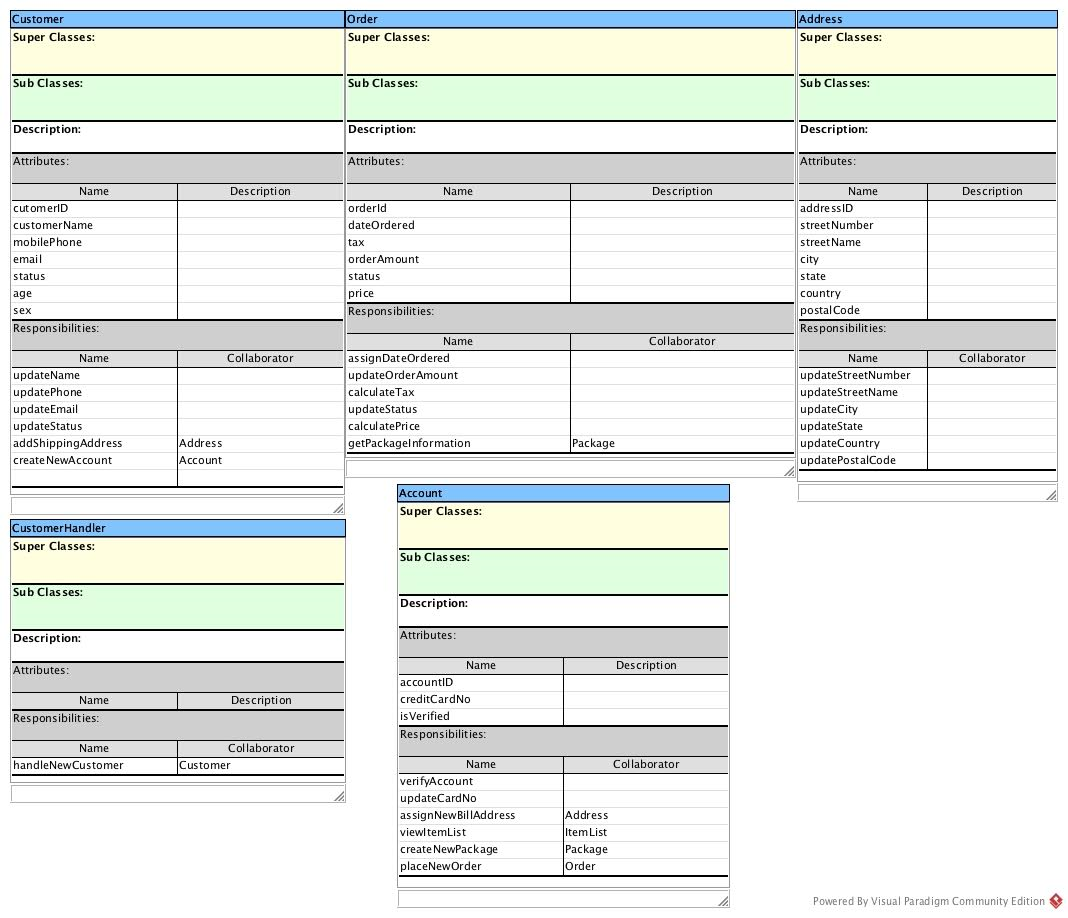
# Domain Class Diagram

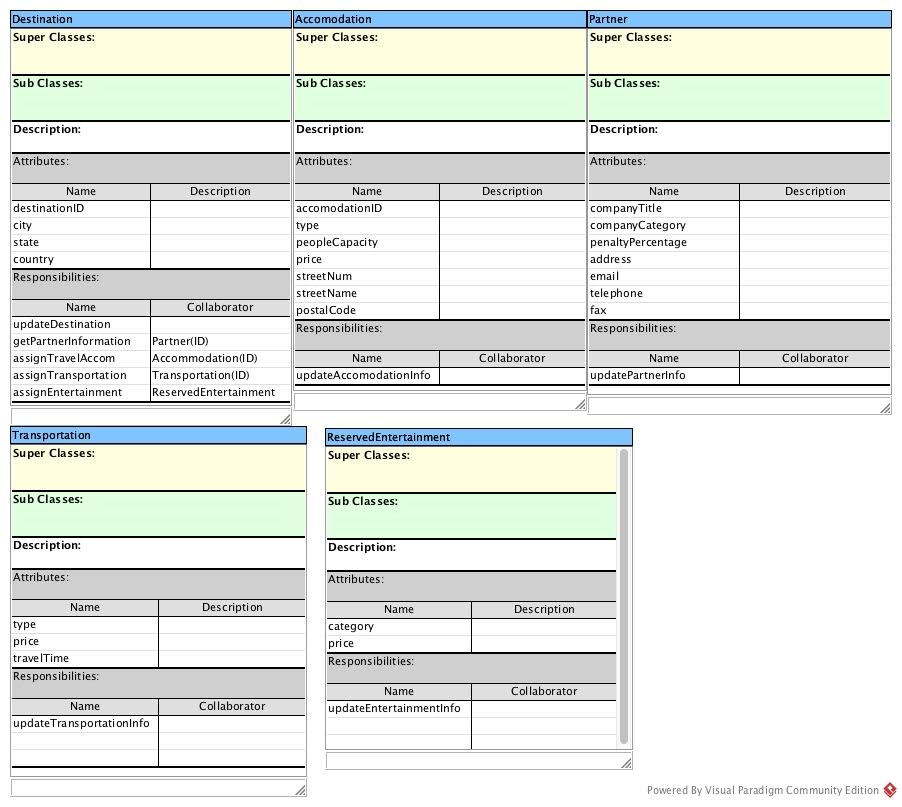
****

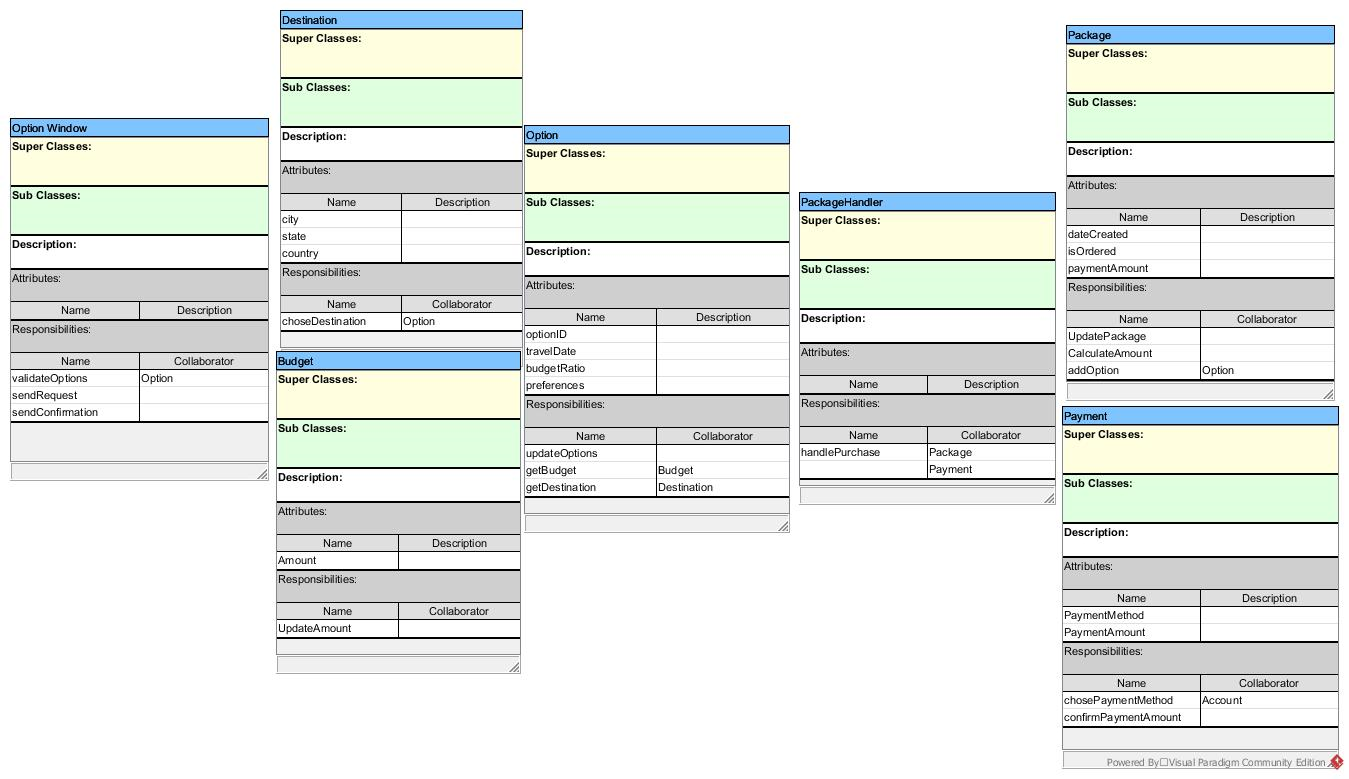
# ****List of Technology Tools for Software Development****

1. Visual Paradigm (Activity Diagrams, Use Case diagrams, Domain Class Diagram)
2. Microsoft Project (Gantt Chart Diagram)
3. Microsoft Word (lists and final document)

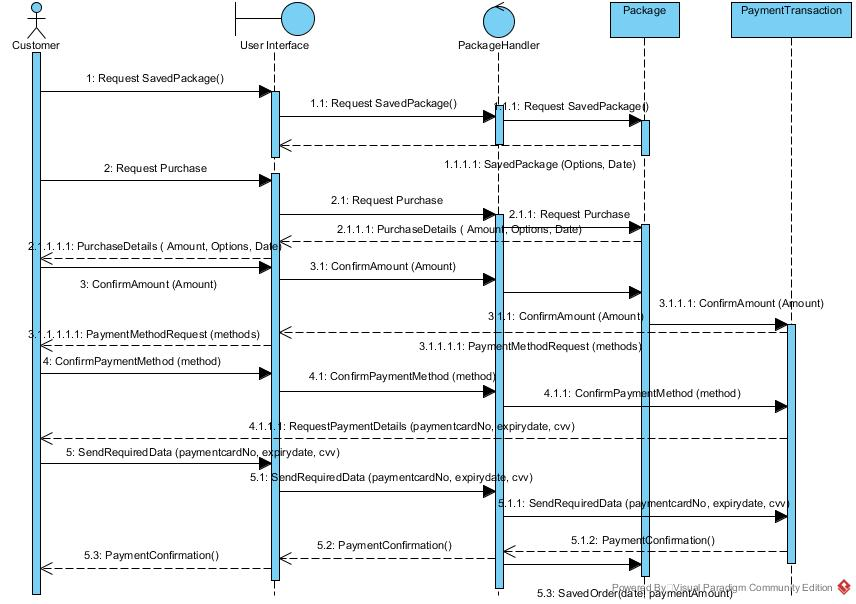
# ****Class Responsibility Collaboration cards****

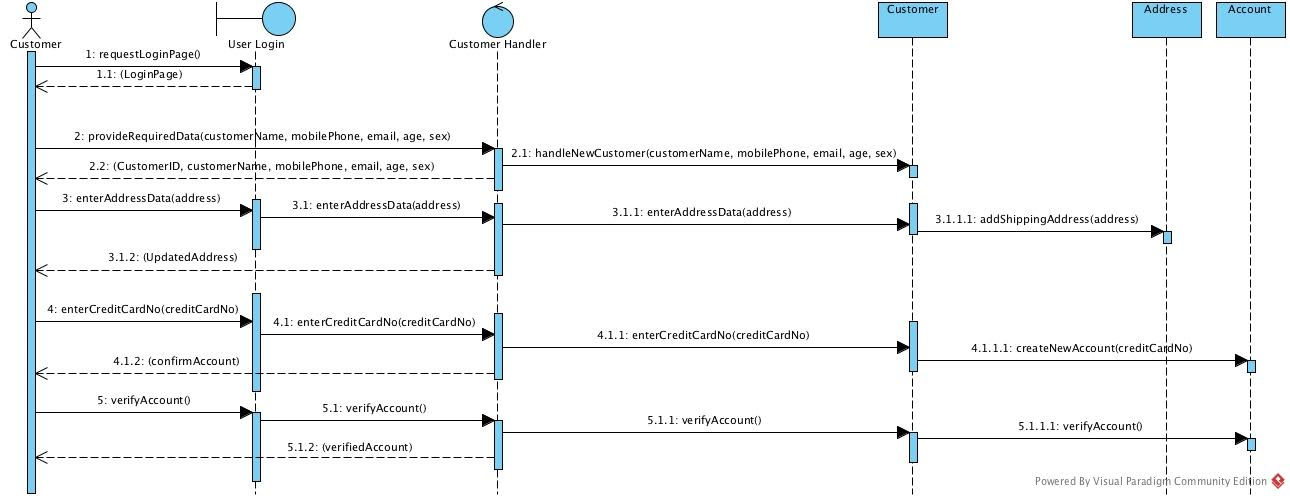


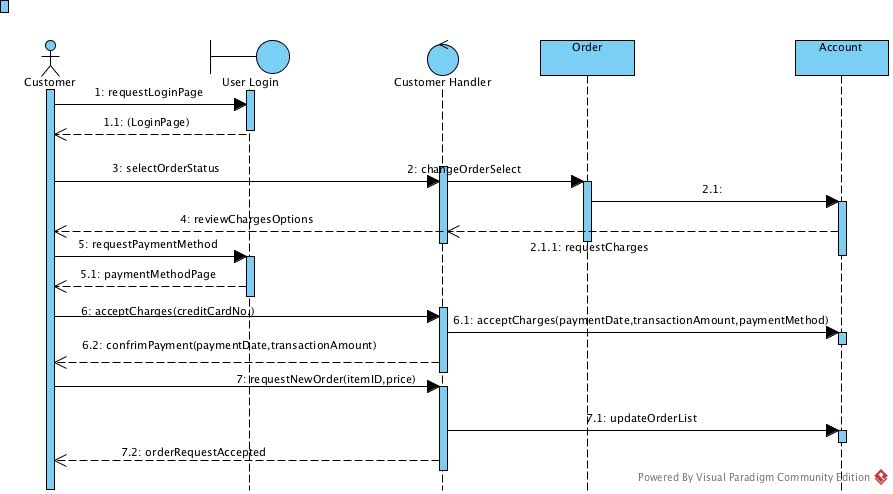




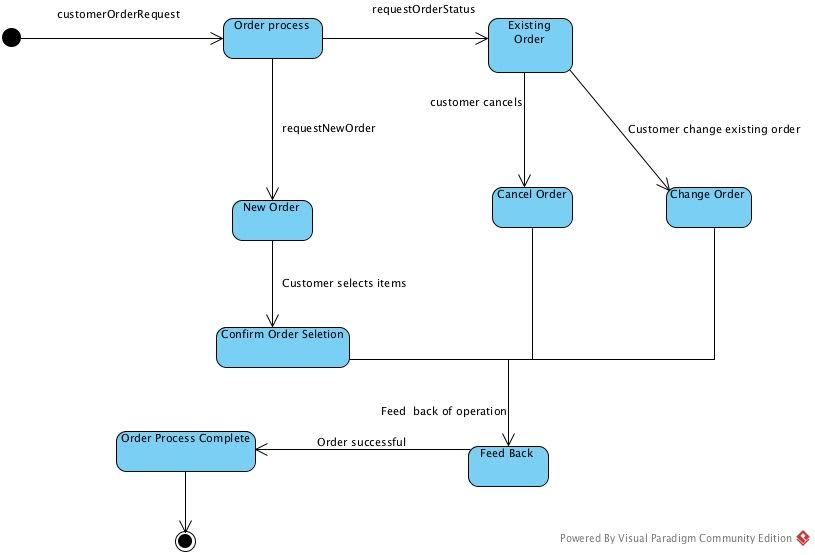
# **Detailed Sequence System Diagrams**

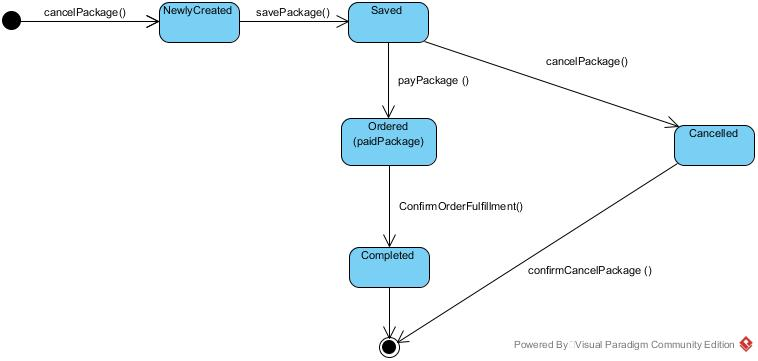






# **State Machine Diagrams**



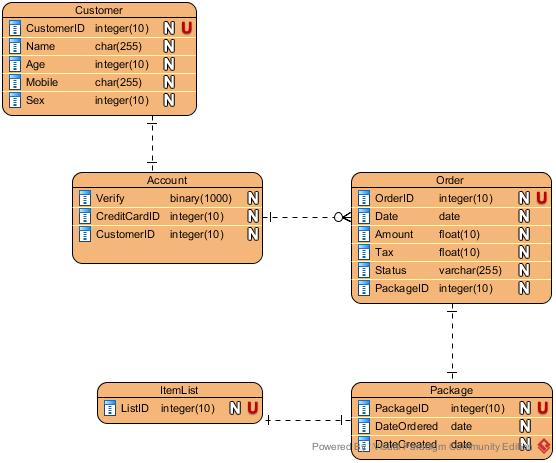




# ****Design Class Diagram****



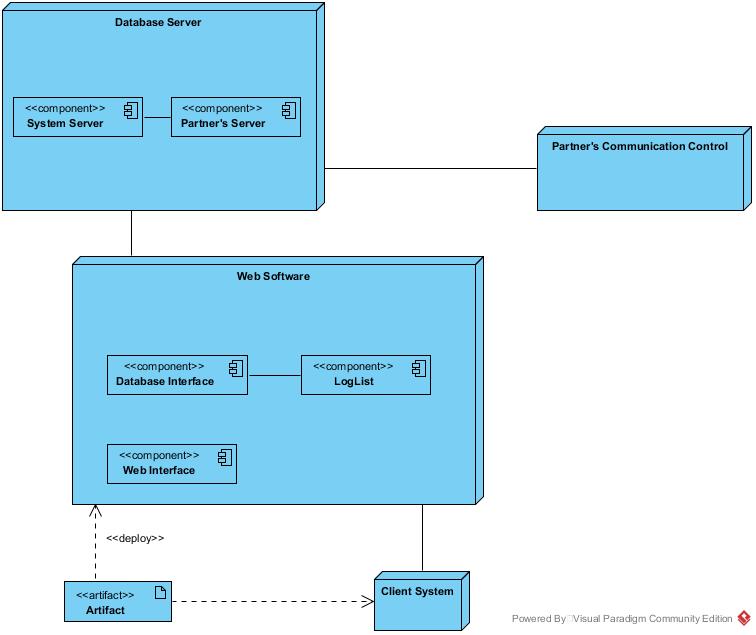
# ****ERD****



# **High level architecture design (component diagram)**

# **C:\Users\Oleksandra\Desktop\Object-oriented engineering\Component Diagram1.jpg**

# **Deployment diagram**



# **Generated Java code**

public class Accommodation {

private int accommodationID;

private string type;

private int peopleCapacity;

private float price;

private string streetNum;

private string streetName;

private string postalCode;

/\*\*

\*

\* @param accomInfo

\*/

public void updateAccommodation(int accomInfo) {

// TODO - implement Accommodation.updateAccommodation

throw new UnsupportedOperationException();

}

}

public class Account {

private int accountID;

private int creditCardNo;

private boolean isVerified;

public void verifyAccount() {

// TODO - implement Account.verifyAccount

throw new UnsupportedOperationException();

}

/\*\*

\*

\* @param cardNo

\*/

public void updateCardNo(int cardNo) {

// TODO - implement Account.updateCardNo

throw new UnsupportedOperationException();

}

/\*\*

\*

\* @param address

\*/

public void assignNewBillAddress(int address) {

// TODO - implement Account.assignNewBillAddress

throw new UnsupportedOperationException();

}

/\*\*

\*

\* @param listID

\*/

public void viewItemList(int listID) {

// TODO - implement Account.viewItemList

throw new UnsupportedOperationException();

}

public void createNewPackage() {

// TODO - implement Account.createNewPackage

throw new UnsupportedOperationException();

}

/\*\*

\*

\* @param orderInfo

\*/

public void placeNewOrder(int orderInfo) {

// TODO - implement Account.placeNewOrder

throw new UnsupportedOperationException();

}

}

public class Address {

private int addressID;

private int streetNumber;

private string streetName;

private string city;

private string state;

private string country;

private string postalCode;

/\*\*

\*

\* @param addressInfo

\*/

public void updateAddressInfo(int addressInfo) {

// TODO - implement Address.updateAddressInfo

throw new UnsupportedOperationException();

}

}

public class Budget {

private float amount;

/\*\*

\*

\* @param amount

\*/

public void updateAmount(int amount) {

// TODO - implement Budget.updateAmount

throw new UnsupportedOperationException();

}

}

public class Customer {

private int customerID;

private string name;

private float mobilePhone;

private string email;

private string status;

private int age;

private string sex;

/\*\*

\*

\* @param customerInfo

\*/

public void updateCustomerInfo(int customerInfo) {

// TODO - implement Customer.updateCustomerInfo

throw new UnsupportedOperationException();

}

/\*\*

\*

\* @param address

\*/

public void addShippingAddress(int address) {

// TODO - implement Customer.addShippingAddress

throw new UnsupportedOperationException();

}

/\*\*

\*

\* @param accountInfo

\*/

public void creatNewAccount(int accountInfo) {

// TODO - implement Customer.creatNewAccount

throw new UnsupportedOperationException();

}

}

public class CustomerHandler {

/\*\*

\*

\* @param customerInfo

\*/

public void handleNewCustomer(int customerInfo) {

// TODO - implement CustomerHandler.handleNewCustomer

throw new UnsupportedOperationException();

}

}

public class Destination {

private int destindationID;

private string city;

private string state;

private string country;

/\*\*

\*

\* @param city

\* @param state

\* @param coutnry

\*/

public void updateDestination(int city, int state, int coutnry) {

// TODO - implement Destination.updateDestination

throw new UnsupportedOperationException();

}

/\*\*

\*

\* @param partnerID

\*/

public void getPartnerInformation(int partnerID) {

// TODO - implement Destination.getPartnerInformation

throw new UnsupportedOperationException();

}

public void assignTravelAccom() {

// TODO - implement Destination.assignTravelAccom

throw new UnsupportedOperationException();

}

public void assignTransportation() {

// TODO - implement Destination.assignTransportation

throw new UnsupportedOperationException();

}

public void assignEntertainment() {

// TODO - implement Destination.assignEntertainment

throw new UnsupportedOperationException();

}

}

public class Item {

private int itemID;

private string title;

private string category;

}

public class ItemsList {

private int listID;

private int itemNumber;

private string itemDescription;

private boolean available;

}

public class Option {

private date travelDate;

private float budgetRatio;

private int preferences;

/\*\*

\*

\* @param optionInfo

\*/

public void updateOption(int optionInfo) {

// TODO - implement Option.updateOption

throw new UnsupportedOperationException();

}

public void getBudget() {

// TODO - implement Option.getBudget

throw new UnsupportedOperationException();

}

public void getDestination() {

// TODO - implement Option.getDestination

throw new UnsupportedOperationException();

}

}

public class Order {

private int orderID;

private date dateOrdered;

private float price;

private float tax;

private string status;

private float orderAmount;

/\*\*

\*

\* @param date

\*/

public void assignDateOrdered(int date) {

// TODO - implement Order.assignDateOrdered

throw new UnsupportedOperationException();

}

/\*\*

\*

\* @param amount

\*/

public void updateOrderAmount(int amount) {

// TODO - implement Order.updateOrderAmount

throw new UnsupportedOperationException();

}

public void calculateTax() {

// TODO - implement Order.calculateTax

throw new UnsupportedOperationException();

}

/\*\*

\*

\* @param status

\*/

public void updateStatus(int status) {

// TODO - implement Order.updateStatus

throw new UnsupportedOperationException();

}

public void calculatePrice() {

// TODO - implement Order.calculatePrice

throw new UnsupportedOperationException();

}

public void getPackageInformation() {

// TODO - implement Order.getPackageInformation

throw new UnsupportedOperationException();

}

}

public class Package {

private date dateCreated;

private boolean isOrdered;

private int priceAmount;

/\*\*

\*

\* @param packageInfo

\*/

public void updatePackage(int packageInfo) {

// TODO - implement Package.updatePackage

throw new UnsupportedOperationException();

}

public void calculateAmount() {

// TODO - implement Package.calculateAmount

throw new UnsupportedOperationException();

}

public void addOption() {

// TODO - implement Package.addOption

throw new UnsupportedOperationException();

}

}

public class PackageHandler {

public void handlePurchase() {

// TODO - implement PackageHandler.handlePurchase

throw new UnsupportedOperationException();

}

}

public class Partner {

private int partnerID;

private string companyTitle;

private string companyCategory;

private float penaltyPercentage;

private string address;

private string email;

private string telephone;

private string fax;

/\*\*

\*

\* @param partnerInfo

\*/

public void updatePartnerInf(int partnerInfo) {

// TODO - implement Partner.updatePartnerInf

throw new UnsupportedOperationException();

}

}

public class PaymentTransaction {

private int paymentID;

private date paymentDate;

private float transactionAmount;

private string paymentMethod;

/\*\*

\*

\* @param type

\*/

public void choosePaymentMethod(int type) {

// TODO - implement PaymentTransaction.choosePaymentMethod

throw new UnsupportedOperationException();

}

public void confirmPaymentMethod() {

// TODO - implement PaymentTransaction.confirmPaymentMethod

throw new UnsupportedOperationException();

}

}

public class ReservedEntertainment {

private int entertainmentID;

private string category;

private float price;

/\*\*

\*

\* @param entInfo

\*/

public void updateEntertainmentInfo(int entInfo) {

// TODO - implement ReservedEntertainment.updateEntertainmentInfo

throw new UnsupportedOperationException();

}

}

public class Transportation {

private int transportationID;

private string type;

private float price;

private date travelTime;

/\*\*

\*

\* @param transInfo

\*/

public void updateTransportationInfo(int transInfo) {

// TODO - implement Transportation.updateTransportationInfo

throw new UnsupportedOperationException();

}

}

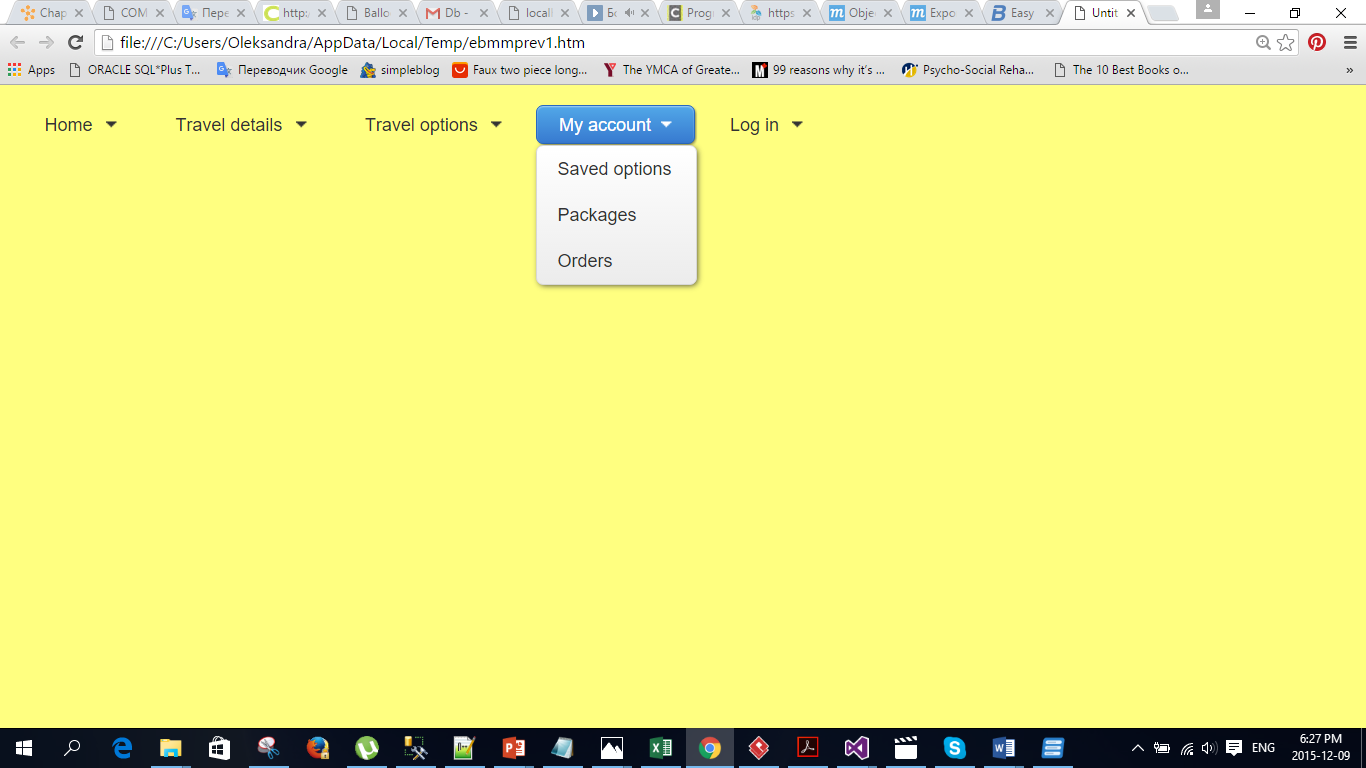
public interface UserLogin {

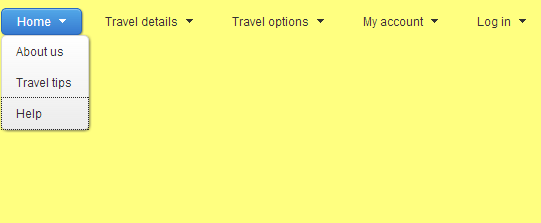
}

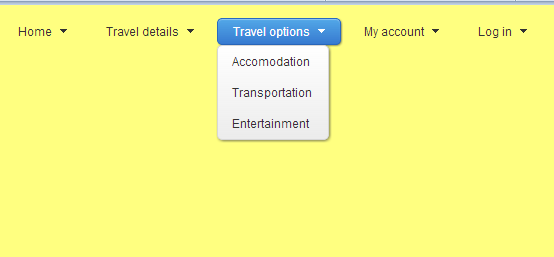
# **User interface**

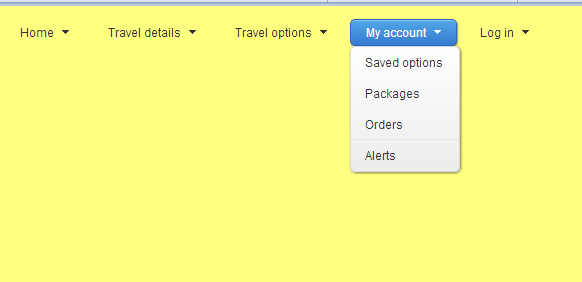
Easy Travel is web based application.

User can navigate via navigation drop down menus:







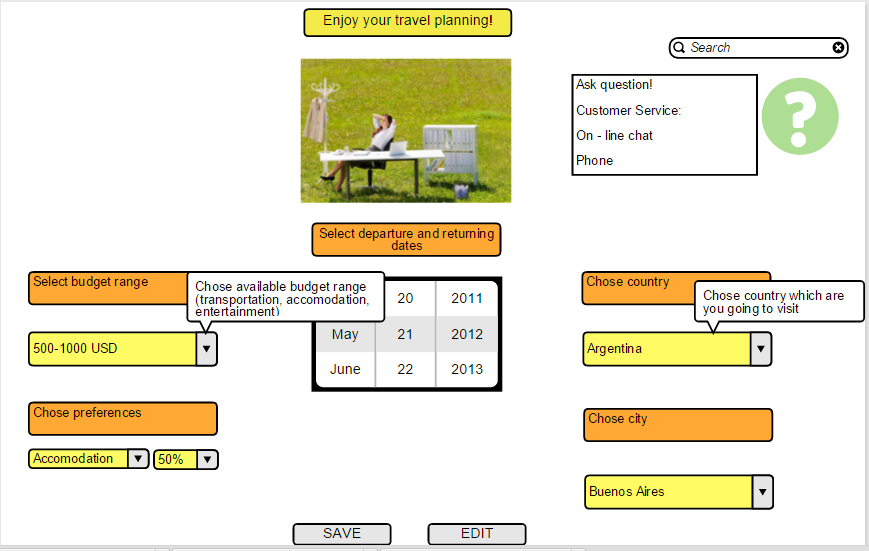


For improving usability of the application:

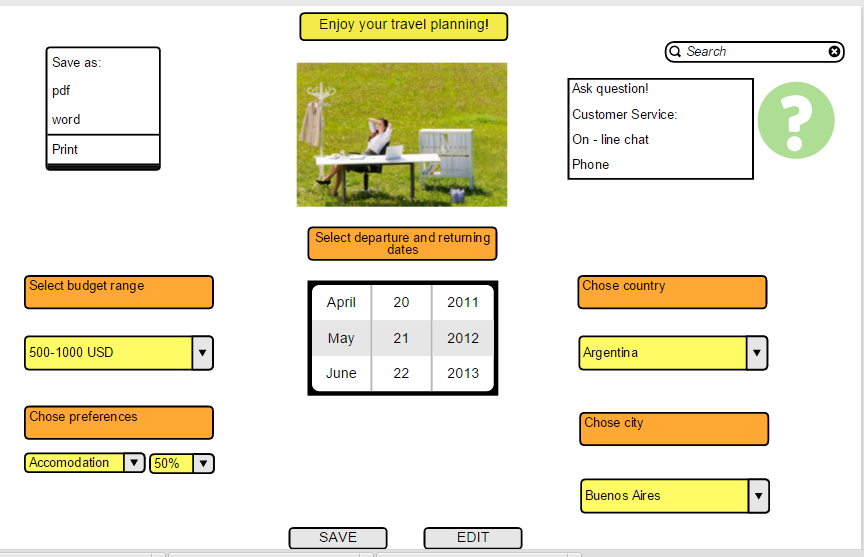
* Every page has Search option
* Every page has link to “Ask questions” (use case “Ask questions”)

**Use case Plan travel**

* Every selection box have tool tips with details
* Drop down list eliminates the input mistakes
* Automatically changeable content accordingly user’s choices



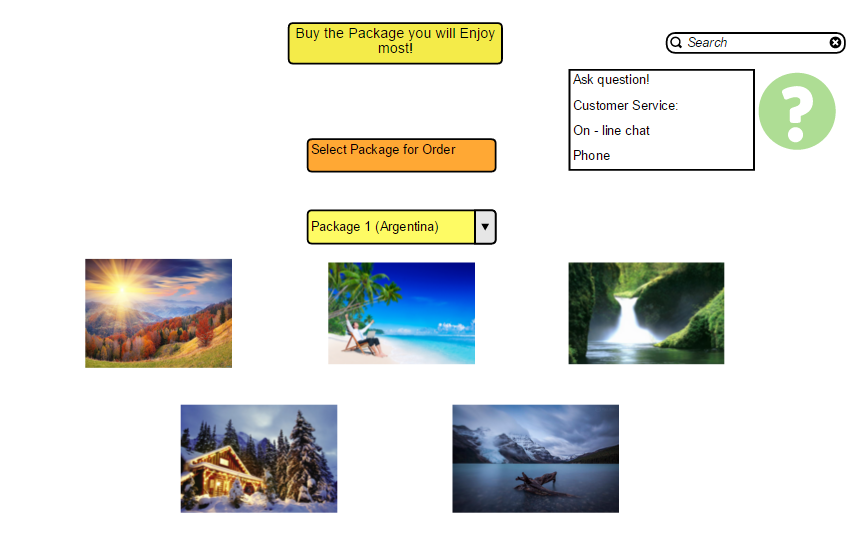
All chosen options are saved automatically. User can save to his PC (tablet, phone etc.) or print whole saved package (“**Print/Save Package**” extended use cases):



**Use case Purchase Saved Package**

1. User can chose Saved Package via two options:

* Visual: via pictures that were saved together with Package
* Drop down list



* 1. If saved package is still up – to –date, no changes made – user will be redirected to payment page
  2. If saved package should be modified, user will be redirected to Travel Planning Page – with highlighting what option should be changed

All other pages will have the same structure:

1. drop down lists for chosen the options: payment type, order number etc.;
2. save, edit buttons;
3. cancel and ok buttons – for making the user actions reversible
4. text areas for individual personal data with validation of entering data types (last, first name, credit card etc)

**Technologies for implementation:**

View layer:

- HTML/CSS

- JavaScript/jQuery

- Java applets/[ASP.NET](http://asp.net/)

Business layer:

- Java servlets/[ASP.NET](http://asp.net/)

- XML

Database layer:

- Oracle Database 12c