**Лабораторная №4**

1. **Диаграмма пакетов**

Общую систему AirportSystem составляют:

1. Пакет Booking System, который работает с классами Пассажир, Полет и Посадочный билет: для процесса покупки и бронирования билетов.
2. Пакет Schedule System, который составляют классы Расписание, Авиакомпания, Самолеты и Рейсы: для управления расписанием полетов.
3. Пакет графического интерфейса UI, который состоит из пакетов для десктопной, мобильной и веб верстки.
4. Пакет Data Management, в котором хранятся все данные.

**2. Диаграмма развертывания**

Система состоит из мобильного приложения для пассажиров и десктопного приложения для сотрудников аэропорта. Мобильное приложения для пассажиров включает в себя среды исполнения для процессов регистрации пассажира и бронирования билетов. Для мобильного приложения предусмотрен артефакт app.apk. Десктопное приложение состоит из среды исполнения для управления расписанием полетов. Для него существует артефакт app.exe. Все приложенияпередают все запросы в AppServer, который эти запросы обрабатывает, и связан с DBServer, который работает с базой данных. Для DBServer предусмотрен артефакт data.db.

**3. Диаграмма компонентов**

Компонент BookingSystem – система бронирования билета

Компонент BuyTicketProcess (покупка билета) использует компонент UserRegistration (для получения информации о пассажире), FightSearchProcess (поиск определенного рейса) и реализует интерфейс IPayment, который ему предоставляет компонент PaymentProcess (для оплаты брониварония).

Компонент FlightSearchProcess использует компонент FlightCatalog (список рейсов), который реализует интерфейс IFlight (для работы с каталогом рейсов).

Компонент UserRegistration требует для работы интерфейс IUserRegistration (для регистрации пользователей, проверки паспортных данных и т.д.).

В целом компонент BookingSystem берет данные из компонента DataManagent, его также использует компонент AnalyticsSystem.

**Лабораторная №5**

1. **Диаграмма деятельности (покупка билета)**

Система предоставляет пассажиру список рейсов. Пассажир выбирает конкретный рейс из списка. Если на выбранный рейс не хватает мест, система предлагает пассажиру выбрать рейс на другие даты, а если и там пассажир не находит нужного рейса, алгоритм заканчивает работу. Если находится нужный рейс, системы переходит на страницу бронирования билета. Пассажир вводит свои паспортные данные, выбирает дополнительные услуги (страховка, возможность возврата, т.д.) и переходит к оплате. Система производит оплату, если она прошла успешно, выводит пассажиру купленный билет. Иначе сообщает об ошибке и ожидает действий от пользователя.

**2. Диаграмма деятельности (создание нового маршрута)**

Диаграмма описывает процесс составления нового рейса полета в аэропорту. Ответственный за расписание вводит данные о новом рейсе, система бронирования проверяет их на корректность и наличие конфликтов, после чего рейс либо добавляется в расписание.

Ответственный за расписание вводит данные о новом рейсе:

* Номер рейса
* Авиакомпания
* Пункты отправления и назначения
* Время вылета и прибытия
* Количество мест

Проверка корректности данных:

* Проверка на наличие всех обязательных полей
* Проверка формата данных (время, количество мест и т.д.)
* Проверка на пересечение времени вылета/прибытия с другими рейсами

Поиск самолета: если есть свободный самолет, формируется новый рейс и добавляется в расписание.

**3. Диаграмма состояний (регистрация пассажира на рейс)**

Состояния пассажира:

* Не зарегистрирован: Пассажир прибыл в аэропорт, но еще не зарегистрирован.
* В очереди на регистрацию: Пассажир стоит в очереди на регистрацию.
* Зарегистрирован: Пассажир зарегистрирован, билет проверен, багаж сдан.
* Прошел досмотр: Пассажир прошел досмотр безопасности.
* Прошел паспортный контроль: Пассажир прошел паспортный контроль.
* На посадке: Пассажир находится в зоне посадки.
* На борту: Пассажир находится на борту самолета.
* В ожидании: Рейс задерживается.

**Лабораторная №3**

**Сценарий «Купить билет»**

**Описание**: Пользователь выбрал нужный рейс и хочет оплатить билет.

**Цель**: Обеспечить пассажиру покупку билета.

**Предусловие**: Пользователь находится на странице покупки билета. До этого применил поиск рейсов.

**Основной сценарий:**

1. Пассажир выбирает команду «Купить билет».

2. Система открывает страницу с конкретным рейсом для бронирования.

***Примечание***: на странице бронирования есть ссылка на страницу с доп.услугами

3. Пассажир вводит свои паспортные данные. Нажимает «оплатить».

4. Система перенаправляет запрос на страницу для ввода реквизитов карты.

5.1. Оплата прошла успешно, система открывает страницу билета.

5.2. Оплата не прошла. Переход на шаг 4.

***Альтернативный сценарий 1:***

**Заменяет шаг 3.** Начинается, когда пользователь открывает страницу бронирования билета:

1. Пользователь вводит паспортные данные. Переходит по ссылке для выбора дополнительных услуг.
2. Система выводит пользователю список дополнительных услуг (страхование, возвратный билет, т.д.)
3. Пользователь выбирает из перечня нужные ему услуги и нажимает «оплатить».

***Альтернативный сценарий 2:***

**Заменяет шаг 5.1.** Оплата прошла успешно.

1. Система открывает страницу билета.
2. Пользователь выбирает «Распечатать билет».
3. Система открывает пользователю ссылку на pdf-файл с билетом.

**Комментарии к ДВИ:**

1. Будущий пассажир: может искать рейсы
2. Пассажир:
   1. Может купить билет, что обязательно включает в себя оплату билета и оформление (ввод паспортных данных и выбор дополнительных услуг – страхование, покупка возвратного билета и т.д.).
   2. При желании билет можно распечатать
   3. Зарегистрироваться на рейс, что включает в себя выбор места
3. Модератор: может создавать, изменять, просматривать и удалять Рейсы и Расписание.

**Лабораторная №2**

1. **Диаграмма классов**
2. **Авиакомпания** – имеет множество самолетов и рейсов (маршрутов).
3. **Самолет** – назначается на рейс. Один самолет может летать по нескольким рейсам.
4. **Рейс –** маршруты, которые предоставляет авиакомпания (место отправления, место прибытия). На один рейс может быть назначены разные самолеты
5. **Полет –** самолет + рейс: кол-во мест (Пассажиров).
6. **Табло -** дата/время, выход на посадку, терминал. Имеет множество полетов.
7. **Посадочный Билет** – полет + пассажир
8. **Первый класс** – наследник от Посадочного
9. **Бизнес класс** – наследник от Посадочного
10. **Пассажир –** может иметь несколько билетов (Полет)
11. **Диаграммы последовательностей**

1) Покупка билета (создание объекта) **Passenger: BuyTicket**

Пассажир у Табло (Schedule) получает информацию, и передает ее для создания объекта Посадочный билет (Ticket).

2) Поиск полета (вложенные активности) **Passenger: GetFlight**

Пассажир вводит место отбытия, место прибытия и дату/время, находить на основе этой информации объект класса Табло. Табло вызывает метод getFlights для получения списка всех полетов. В цикле у каждого полета проверяем количество свободных мест. Если количество мест положительно, мы возвращаем текущий полет и прерываем цикл.

3) Создание маршрута (РЕЙС) (циклы/условие) **Airline: createRoute**

Авиакомпания имеет у себя коллекцию самолетов. В цикле у каждого самолета проверяем его доступность. Если обнаруживается свободный самолет, создаем новый маршрут, передаем туда поля Отбытие и Прибытие. Возращаем новый маршрут.

1. **Диаграмма объектов**

Снимок системы в конкретный момент времени: Кайрат Нуртас собирается на вылет из Новосибирска в Астану 31 октября 2024 года в 17:35 из международного терминала М1, через выход В1, на самолете Боинг-737-800 авиакомпании S7 Airlines, в котором 189 мест. Кайрат имеет место 3А в бизнес классе.

1. **Код**

public class Airline {

public string name;

public Plane []airline\_planes;

public Route []airline\_routes;

public Route createRoute(string departure, string arrival) {

Route newRoute;

foreach (Plane plane in airline\_planes) {

bool freePlane = plane.isFree();

if (freePlane) {

newRoute = new Route(departure, arrival);

break;

}

}

return newRoute;

}

}

public class Plane {

public string model;

public bool isFree() {}

}

public class Route {

public string departure;

public string arrival;

public Route(string departure, string arrival) {}

}

public class Flight {

public int seat\_count;

public Plane plane;

public Route route;

public void getInfo() {}

public int checkFreeSeats() {}

}

public class Schedule {

public string dateTime;

public string gate;

public string terminal;

public Flight []flights;

public string display() {}

private void changeGate(string newGate) {}

public Flight[] getFlights() {

return flights;

}

}

public class Passenger {

public string name;

public string surname;

public string passport;

public Ticket BuyTicket(Schedule s) {

string data = s.display();

Ticket ticket = new Ticket(data);

return ticket;

}

public Flight GetFlight(string date, string departure, string arrival) {

Schedule schedule = compute(date, departure, arrival);

Flight []flights = schedule.getFlights();

foreach (Flight flight in flights) {

int hasPlace = flight.checkFreeSeats();

if (hasPlace > 0) {

return flight;

}

}

return null;

}

public Schedule compute(string date, string departure, string arrival) {}

}

public class Ticket {

public Flight flight;

public Passenger passenger;

public Ticket(string data) {}

}

public class FirstClass : Ticket {

public string first\_seat;

}

public class BusinessClass : Ticket {

public string business\_seat;

}

**Лабораторная №1**

[https://drive.google.com/file/d/1WtEkevqHSmtCLXu89py4w9LKymf2CZlZ/view?usp=sharing](https://vk.com/away.php?to=https%3A%2F%2Fdrive.google.com%2Ffile%2Fd%2F1WtEkevqHSmtCLXu89py4w9LKymf2CZlZ%2Fview%3Fusp%3Dsharing&utf=1)

1. **МПО**

Сущности:

1. **Авиакомпания** – имеет множество самолетов и рейсов (маршрутов).
2. **Самолет** – назначается на рейс. Один самолет может летать по нескольким рейсам.
3. **Рейс –** маршруты, которые предоставляет авиакомпания (место отправления, место прибытия). На один рейс может быть назначены разные самолеты
4. **Полет (Билет) –** самолет + рейс: дата/время, кол-во мест (Пассажиров).
5. **Посадочный Билет** – полет + пассажир: место в самолете.
6. **Пассажир –** может иметь несколько билетов (Полет)
7. **Диаграмма коммуникаций**

Процесс покупки билета пассажиром.

1: findTheRoute – пассажир ищет рейс

1.1: data = compute(info) – введенные пассажиром данные преобразуются в нужный формат

1.2: getFlight(data) – получаем список доступных полетов

1.2.1 \* flight = chooseFlight(data) – в цикле ищем полет, который удовлетворяет введенным данным

[i[flight= not null] 1.2.2: hasPlace = checkPlaces() – проверяем наличие мест в полете

[hasPlace = true] 1.2.3: registerTicket() – регистрируем посадочный билет

1. **Диаграмма анализа**

Процесс покупки билета пассажиром.

1: FindRoute – пользователь на странице рейсов (RoutesPage) ищет нужный маршрут

2: ChooseRoute – пользователь на странице рейсов выбирает нужный маршрут. Подразумевается, что там же пользователь выбирает дату и время.

3: ChooseFlight - контроллер рейсов (RoutesController) берет информацию из Полет (Flight). Подразумевается, что контроллер в процессе проверяет кол-во мест, выбирает самолет.

4: CreateNewTicket – контроллер рейсов вызывает контроллер билетов (TicketContoller) для создания нового посадочного билета

5: CreateTicket – контроллер билетов создает Посадочный Билет (Ticket). Контроллер выдает место пассажиру

6: ShowTicket – контроллер билетов выводит на страницу TicketPage созданный посадочный билет

7: Show – пользователь видит у себя на устройстве новый билет.