Mini-project Instructions

Aim and motivation

In this course, we will cover technologies (e.g., EJB, SPRING) that can be used for building distributed component-based systems. Lectures will cover only the foundations of these technologies with some code examples, but that is not enough for complete comprehension. Therefore, in order to get hands-on experiences on how to use these technologies, you will design and implement two independent mini-projects that will connect together via a mashup service to form a complete distributed component system.

While doing these mini-projects, you will have to study a great deal about the aforementioned technologies by yourself. Therefore, you can expect a lot of reading, a large amount of programming, and some moments of frustration. However, if you work hard, you will overcome those moments of frustration and will graduate from this course as a champion of distributed component systems. The knowledge and techniques that you will learn will be useful in your future career, particularly in server-side Java development.

NOTE: It is expected that you already know Java language because we will use it a lot.

Work mode

Mini-projects are team-based projects with one student acting as the team leader. In the best case, all work is equally distributed among team members throughout all mini-projects. It is up to the team leader to conduct the project so that the best possible result will be achieved.

Here are the rules for team formation:

- Team size is 3 students.
- You can form teams by yourself; if you cannot find a team, we will assign you into one.
- A team leader is elected by the team members.
- Teams remain the same throughout all mini-projects.

It is recommended that each team member will take the responsibility for some component(s) in mini-projects. This way everybody can learn something and the work is balanced.

The big picture (with example)

During this course you will design and implement a distributed component system which comprises at least two subsystems and a mashup service. The subsystems are independent, and they are implemented with the covered technologies (EJB, Spring). The mashup service utilizes the two subsystems in order to provide a mashup service. In other words, a user can use the two subsystems independently, or via the mashup service (see Figure 1 below).

The components in your system communicate via interfaces. There should also be two connections to third-party APIs, one from each subsystem.

Here is **an example** with two independent subsystems and a mashup service that combines them all:

- **Subsystem 1**: Hotel reservation system (Spring) – enables users (or another subsystem) to manage hotel reservations. Connects to a TripAdvisor API for hotel information.
- **Subsystem 2**: Flight ticket reservation system (EJB) – enables users (or another subsystem) to manage

flight ticket reservations. Connects to a Flight Search API for flight information.

- **Mashup service**: Holiday reservation system – connects to the two subsystems to enable users to make holiday reservations (flights + hotel).

The diagram below illustrates this example architecture. Please note that this diagram is just a sketch – when you design and implement your system, it should contain different (and more!) components.

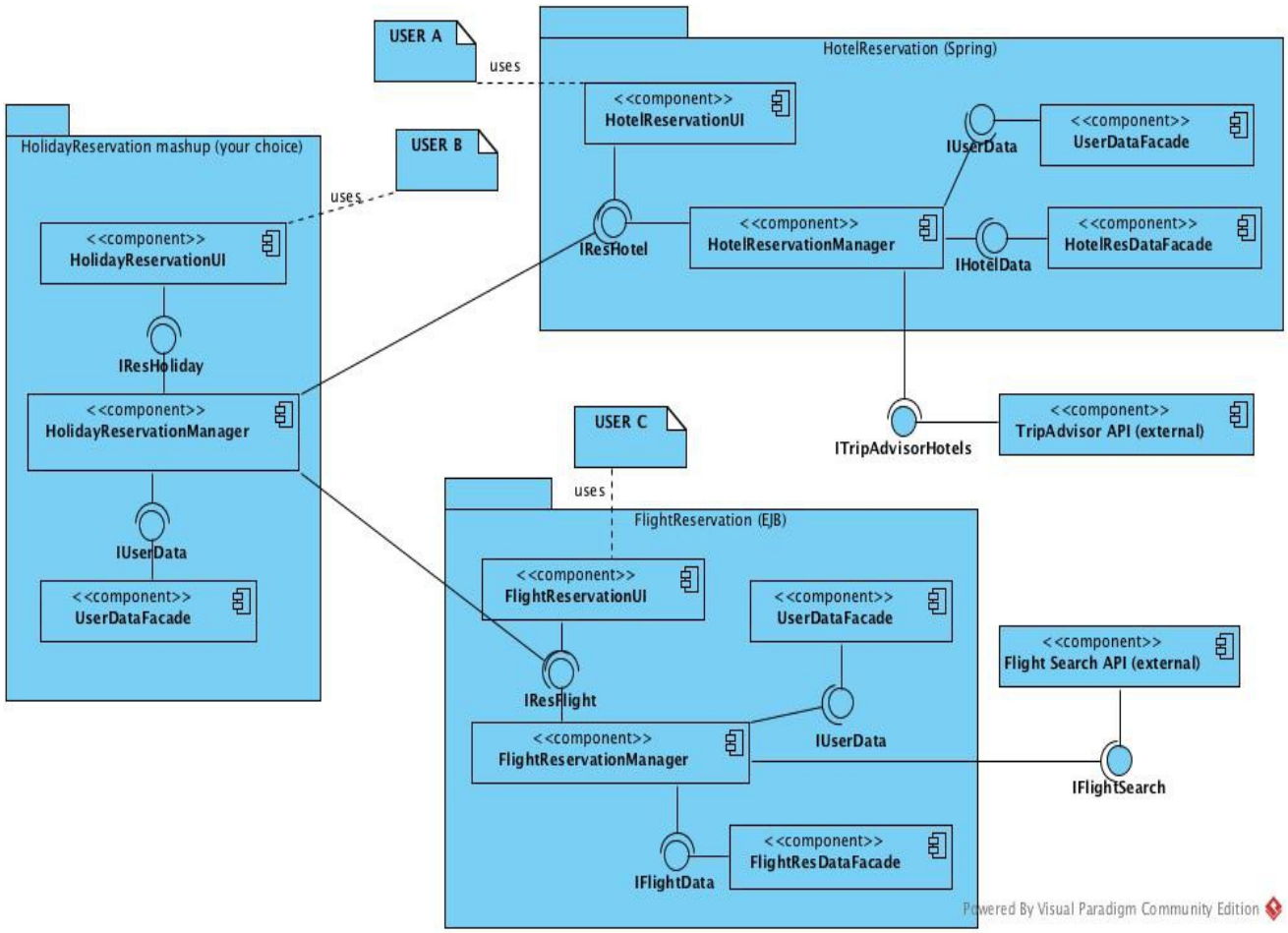

Figure 1. An example system with two independent subsystems (Hotel Reservation, Flight Reservation), and a mashup service combining the two (Hotel Reservation).

Here the "TripAdvisor API" and "Flight Search API" are third party APIs which are used to get information about hotels and flights.

IMPORTANT: The above scenario is an example – you should design your scenario including what the subsystems & the mashup service are and how they work together. Just make sure that the requirements (see below) are met.

Division into mini-projects

Your work is divided into three mini-projects (2 independent, 1 mashup) for two reasons. Firstly, we cannot cover all technologies at once during the lectures. Secondly, we don't want you to do all the work during the last couple of weeks of the course. Here is the chronological of your project work:

- **Project proposal:**
  - High-level design of the entire system with all moni-projects (see instructions below)
- **Mini-project 1:**
  - Design and implementation of Subsystem 1 using **EJB** (+ related technologies).
  - Presentation of the overall architecture and Subsystem 1.
- **Mini-project 2:**
  - Design and implementation of Subsystem 2 using **Spring Framework**.
  - Presentation of Subsystem 2.
- **Mini-project 3:**
  - Design and implementation of a mashup service using **any technology of your choice (e.g., JavaScript, Android, etc)**.
  - Combining the subsystems together.
  - Presentation of the complete system.

Requirements

Size of the project

As mentioned above, your system should have at least two subsystems that are implemented with Spring Framework and EJB. The technology used for implementing the mashup service is up to you.

The minimum number of components in each mini-project is as follows:

- Subsystem 1 (Spring): 6
- Subsystem 2 (EJB): 6
- Mashup service: 4

Of course, you are allowed to add more components than the minimum!

Communication

Internal communication of subsystems is based on method calls via interfaces. Each subsystem has a different way of connecting components together depending on the technology:

- **Spring Framework**: Dependency injection (manual or automatic wiring)
- **EJB**: Dependency injection or JNDI

External communication between subsystems also happens v i a interfaces; however, the method of communication is up to you. For example, you could use:

  - JSON/XML over HTTP (e.g., web services)
  - Java Remote Method Invocation (Java RMI)
  - SOAP (Simple Object Access Protocol)
  - A custom protocol over socket communication (not recommended because it can get complex)

Data storage

Subsystems should use a persistent data storage (e.g., database) to store the data. You can use for example JPA (Java Persistance API) to map Java objects to database tables without SQL.

User interface

Each subsystem should have a user interface component. This is important so that you can test and demonstrate each subsystem before you combine them all together. The UI components of subsystems can be implemented as standalone client applications, mobile apps, or websites. But keep the UIs simple because this is not a GUI class.

Third party APIs

Both of your subsystems should communicate with a third party API to request a service or data. You can choose a target API based on your needs. Here are some examples

- User authentication API (e.g., Facebook, OpenID)
- Public transport API (e.g., Seoul public transport API, Helsinki public transport API, etc).
- Social networking API (e.g., Facebook, Twitter)
- Other content API (e.g., hotel information, book information).

The following links list some APIs (mostly HTTP-based) to help you get started (search online for more):

http://en.wikipedia.org/wiki/List_of_open_APIs

http://www.programmableweb.com/apis

http://www.publicapis.com/

https://compassioninpolitics.wordpress.com/2009/06/03/best-free-apis-for-web-developers/

Security

This is not a data security class, so you don't need to worry about making communication and data storage secure by encryption. However, your system should include user authentication.

Documentation

All source code must be well documented. Additionally, you should submit a technical documentation at the end of the course as follows:

- component diagram of the entire system including subsystems and mashup
- explanations of the component diagram
- purpose of each subsystem and mashup
- features of each subsystem and mashup
- communication between and within subsystems/mashup (interfaces, protocols, message formats)
- test report (for each test case: what feature is tested, what is input, what is expected ouput, and what is actual output)
- instructions for setting up and running the system (including separate instructions for all subsystems)
- roles of team members (incl. component responsibilities)

It is recommended that you write documentation for each subsystem after you finish implementing them. This way it will be easier for you to create a technical documentation at the end of the course.

Project proposal

First you should write and submit a project proposal (PDF) which must be accepted by professor/TA before you can start working. The proposal should contain the following information:

- The purpose and features of your system
- High level architecture design (component diagram)
  - what subsystems / mashup will you have?
  - what kind of components will be in the subsystems / mashup?
  - What technology each subsystem / mashup will use?
  - Which external API(s) will you use?

Your architecture design cannot be very precise at this moment, but it should give us a rough idea of the scope of the project.

Presentations

You will give four presentations as follows:

- Project proposal presentation
  - Purpose and motivation
  - Overall design of the system, including subsystems and mashup service (component diagram).
  - Planned roles
  - Schedule
- Mini-project 1 presentation
  - Features and components of your Spring subsystem (implemented component diagram)
  - Communication between the components, external API
  - Demonstration
- Mini-project 2 presentation
  - Features and components of your EJB subsystem. (implemented component diagram)
  - Communication between the components, external API
  - Demonstration
- Mini-project 3 + complete system presentation
  - Overall system architecture, including also interface between the subsystems (implemented component diagram)
  - Features and components of the mashup service
  - Demonstration
  - Roles of team members for each mini-project.
  - Lessons learned beyond lecture materials.

Schedule (tentative)

What	When	How
Team formation	March 13	Team leader notifies TA about the team.
Project proposal presentation	March 31	Present in class (10min)
Project proposal (incl. high-level design)	April 1	Submit project proposal (PDF) to e-class
Mini-project 1 (Spring) presentation	April 26	Give presentation; submit presentation (PDF); submit peer evaluation.
Mini-project 2 (EJB) presentation	May 24	Give presentation; submit presentation (PDF); submit peer evaluation.
Mini-project 3 (mashup) + complete system presentation	June 9, June 14	Give presentation; submit presentation (PDF); submit peer evaluation.
System demonstration	June 16 - June 17	Demonstrate the system to TA (make an appointment).
Final results (all code, technical documentation)	June 17	Submit to e-class (ZIP for code and PDF for documentation)

Grading

Mini-projects will be graded individually, and you will also receive a grade for the entire system (i.e. how well your subsystems and mashup service work together). We will consider the following aspects in grading (weights will be decided later):

- For each mini-project
  - Features
  - Architecture design
  - Interface design and communication between components
  - Code correctness (bugs)
  - Code efficiency (smart, efficient coding style)
  - Mini-project presentations (incl. final presentation)

- For the entire system
  - Overall architecture
  - Interface design and communication between subsystems
  - Opponent duties in final presentation
  - Technical documentation
  - System demonstration to TA after final presentation

Peer evalution

To prevent freeloaders, each team member will receive a grade according to work input in the project. For each mini-project you will submit a peer evaluation document which is used to adjust grades between team members.

Getting help

Renny Lindberg (rennylindberg@ajou.ac.kr) is the TA in this course. Both Renny and professor have only a limited amount of time to help you. Essentially, we will NOT be your personal debuggers. IDE's debugger tools, screencast videos, and other online resources are your first choices for help. If you really cannot find a solution to the problem, then you may first contact Renny, then professor.

Good luck!