Project Development and Design Tips

Lesson 1

Luca Arrotta and Riccardo Presotto

EWLab – Università degli studi di Milano

Professor: Claudio Bettini

These slides are based on previous versions created by Letizia Berolaja, Sergio Mascetti, Dario Freni, Claudio Bettini and Gabriele Civitarese

Copyright

Some slides for this course are partly adapted from the ones distributed by the publisher of the reference book for this course (Distributed Systems: Principles and Paradigms, A. S. Tanenbaum, M. Van Steen, Prentice Hall, 2007).

All the other slides are from the teacher of this course. All the material is subject to copyright and cannot be redistributed without consent of the copyright holder. The same holds for audio and video-recordings of the classes of this course.

The next steps

- Four lessons that aim at assisting the development of the project:
  - You can take advantage of these lessons to develop your project
  - You can clarify doubts or ask for advice

- The lessons have the usual timetable and the usual duration
- The first part consists of a short lecture
- Then, it is possible to work on your project

- It is mandatory that you work <u>individually</u>
  - This does not mean that you can't talk with your colleagues…
  - Two <u>copied</u> projects are EVIDENT

Disclaimer

- The content of these slides has to be considered indicative
  - Development and design suggestions will be provided.
  - Each student can make different choices

- The directives on how to carry out the project are reported in the text of the project on the course website.

Applications to develop

- The system requires to develop:
  - *SETA*
  - *Taxi*
  - *Administrator Server*
  - *Administrator Client*

Recommended Development flow I

- First step (today's lesson): **REST server and SETA development**
  - Design of the *Administrator Server* (resources and methods)
  - Synchronization problems analysis
  - Testing of the REST server with dedicated tools
  - *Administrator Client* Development
  - Implementation of SETA and the MQTT protocol to publish and receive orders

- Second step (second lesson): Development of the taxis’ network
  - Architecture and protocols design of the peer-to-peer network of taxis
  - Insertion of a taxi in the peer-to-peer network
  - Rides management via a distributed and decentralized algorithm
  - Removal of a drone from the peer-to-peer network

Recommended Development flow II

- Third Step (Third lesson): Sensor data collection and local statistics
  - Implementation of the sensors data collection.
  - Computation and communication of the local statistics

- It is **crucial** to carefully consider both the internal synchronization and distributed synchronization problems

Administrator Server

- The *Administrator Server* is a single application that is in charge of:
  - Manage the insertion and removal of taxis
  - Receive the local statistics about the taxis’ state and the pollution level
  - Enable the *Administrator Client* to query statistics

- These services must be delivered via a REST architecture

- Synchronization mechanisms are required to manage access to the shared resources

Resources and Task

- Which resources:
  - The first step involves the identification of:
    - Which resources have to be modeled
    - The CRUD operations to perform on resources and the mapping with HTTP verbs
  - Can the set of taxis be considered a resource?
  - And the statistics?

- Mapping example:
  - GET → obtain information and statistics about the taxis of the smart-city
  - POST → insert a new taxi into the smart-city
  - PUT → modify the information of a taxi (is it useful?)
  - DELETE → remove a taxi from the smart-city

Handle taxis

- Insert/remove a taxi
    - When a taxi requests to join the network, the *Administrator Server*:
        - Tries to add the taxi to the smart-city
        - If a taxi with the same identifier already exists, an error message is returned
        - Otherwise, the taxi is added to the list of taxis and the *Administrator Server* returns to that taxi:
            - The position of the recharge station of a randomly chosen district
            - The list of taxis already registered in the smart-city

- The removal of a taxi consists of simply removing it from the list of taxis

- What kind of synchronization is required?

Handle local statistics

- Taxi side:
  - It periodically forwards to the *Administrator Server* its local statistics
  - On the *Administrator Server* side, these statistics will be saved in a data structure that enables subsequent analysis

- *Administrator Client* side:
  - Interfaces are needed to analyze the data as the project requires

- Try to make the synchronization as fine-grained as possible:
  - Is it useful to block any server-side operation (e.g., insert/removing taxis) while statistics are being calculated?
  - Is it useful to block the whole data structure while computing statistics?

Jersey: reminder

- Remember:
  - Each class that manages a resource (annotated with @Path) is instantiated (approximately) every time a single HTTP request is executed
  - It is therefore necessary to properly manage the shared memory access (i.e., using singletons or, alternatively, static fields)
  - Synchronizing the methods of a class annotated with @Path is useless!

- Multi-threading is handled automatically: concurrent calls concurrently execute the code of different instances of the class that manages the resource

- Concurrency issues need to be appropriately handled

Synchronization

- Possible synchronization issues:
  - During the server development, many synchronization problems may arise:
    - The list of taxis is modified and read concurrently
    - Statistics are added and read concurrently

- Carefully, select whether and where to use a synchronization statement and overall, manage the synchronization as fine-grained as possible

- Using *synchronized* randomly and everywhere is not a best practice

How to test the *Administrator Server*

- The first step is to test every single REST method with specific tools (e.g., Advanced REST Client)

- Test concurrency problems with *sleep()*

- To automate the trickiest tests, you may write some Java code and then implement the *Administrator Client*

SETA and MQTT

- SETA is a process that implements an MQTT publisher
  - Start an MQTT broker and connect SETA to it
  - SETA periodically generates rides and publishes them to the topic of the corresponding district

- To test SETA, create another process that simulates a subscriber over the topics *seta/smartcity/rides/#*

- Note that this simulated subscriber will be the basis on which you can build the logic of the taxi

Good Job!

Every
Ware
Lab