Functional Programming: Doodle

Laurent Christophe, Wolfgang De Meuter

Programming Project: Assignment #2 (2015 - 2016)

1 Introduction

The final mark for the Functional Programming course is based 50% on a programming project (25% on the first programming assignment, 25% on a second programming assignment) and 50% on the oral exam. This document describes the first programming assignment. The second assignment will be handed out later in the semester.

Submission is done by sending your raw code files to the teaching assistant: Laurent Christophe (`lachrist@vub.ac.be`). The assignment is due January 11th (8AM). You will have to defend both parts of your project individually during the exam session on the same date as the oral exam. Notice that the execution of the project is strictly individual and no plagiarism shall be tolerated!

The project will be marked according to how well you fulfil the functional requirements and according to how well you apply the concepts explained during the lectures and the lab sessions. If you encounter any problem or if you have a precise question, feel free to contact the assistant at `lachrist@vub.ac.be`.

2 Overview

The goal of this programming assignment is to extend the doodle system of the first assignment to better handle the scheduling of exams "by appointment". At first, teachers are expected to create a doodle for each one of their exams. Then, students can subscribe to these exams and vote for their preferred exam time slots. The system can then output an *exam schedule* which is a mapping from exams to time slots such that: (i) time slots belong to the associated teacher doodle that was proposed at first. (ii) teachers do not have to take two exams at the same time. (iii) students do not pass two exams at the same time. Your system should be able to find the exam schedule (if any) that scores the highest amongst all the student's votes.

3 Implementation and Protocol

For this assignment, you will have to create a TCP server written in Haskell that implements the protocol detailed in Table 1. Your server should handle TCP requests concurrently using the module `Network.TCP` and the function `Control.Concurrent.forkIO`. Each TCP connection is expected to carry exactly one client request and one server response, after that it should be closed. Data race conditions should be handled using software transactional memory from the module `Control.Concurrent.STM` – as seen in the lectures. To manage rights, requests should always embeds a user login. If the user identifier does not exists for the requested role or if the password does not match, the server should return the message `wrong-login`. Next, we enumerate the requests that your server should handle.

`add-student` and `add-teacher` : enables the administrator to add a student or a teacher to the system. If the requested user identifier is already taken, the server should return the message `id-taken`. If the requested user identifier is available, the server should return a randomly generated password of at least 4 characters. Note that the administrator login should be passed as command-line parameters during the server startup.

```
=> add-teacher admin@1234 walter
<= ok f1Qs
=> add-student admin@1234 jesse
<= ok 2bYo
```

`change-password` : enables users to change their password.

```
=> change-password walter@f1Qs foobar
<= ok
```

`set-doodle` : enables teachers to create new exam doodles. Unlike in the previous assignment, an exam doodle may contain overlapping time slots. If the teacher already defined a doodle for the exam, the new doodle should overwrite the old one. If an other teacher already claimed the exam identifier, the server should return the message `id-taken`.

```
=> set-doodle walter@foobar Cooking [
  2016-01-04T14:00+01:00 / 2016-01-04T16:00+01:00,
  2016-01-04T13:00+01:00 / 2016-01-04T15:00+01:00
]
<= ok
```

`get-doodle` : enables users to get the doodle associated to an exam. If the exam does not exist, the server should return the message `no-such-id`. The time slots should be sorted by starting time.

```
=> get-doodle Cooking
<= ok [
  2016-01-04T13:00+01:00 / 2016-01-04T15:00+01:00,
  2016-01-04T14:00+01:00 / 2016-01-04T16:00+01:00
]
```

`subscribe` : enables students to subscribe to an exam.

```
=> subscribe jesse@2bYo Cooking
<= ok
```

`prefer` : enables subscribed student to vote for their preferred exam time slot. If the student is not subscribed to the exam, the server should return the message `not-subscribed`. A student may only vote for one preferred time slot for each one of its subscribed exam. A second vote for the same exam simply overwrite the first vote.

```
=> prefer jesse@2bYo Cooking 2016-01-04T13:00+01:00/2016-01-04T15:00+01:00
<= ok
```

`exam-schedule` : enable a user to get the current best exam schedule. If there is no exam schedule possible, the server should return the message `no-possible-exam-schedule`.

```
=> exam-schedule
<= {Cooking : 2016-01-04T13:00+01:00/2016-01-04T15:00+01:00}
```

Grammar Rule			Comments
`REQUEST`	$\rightarrow$	`add-teacher LOGIN TOKEN`	The administrator `LOGIN` add the teacher `TOKEN` to the system.
		`add-student LOGIN TOKEN`	The administrator `LOGIN` add the student `TOKEN` to the system.
		`change-password LOGIN TOKEN`	Change the password of `LOGIN` to `TOKEN`.
		`get-doodle LOGIN TOKEN`	User `LOGIN` gets the doodle of course `TOKEN`.
		`set-doodle LOGIN TOKEN DOODLE`	Teacher `LOGIN` sets the doodle of the course `TOKEN` to `DOODLE`.
		`subscribe LOGIN TOKEN`	Student `LOGIN` subscribes to course `TOKEN`.
		`prefer LOGIN TOKEN SLOT`	Student `LOGIN` prefers the pass the exam of course `TOKEN` at the time slot `SLOT`
		`exam-schedule LOGIN`	User `LOGIN` gets the current best exam schedule.
`RESPONSE`	$\rightarrow$	`wrong-login`	Applicable for all requests.
		`ok TOKEN`	Applicable for: `add-student` and `add-teacher`.
		`ok DOODLE`	Applicable for: `get-doodle`.
		`ok SCHEDULE`	Applicable for: `best-schedule`.
		`ok`	Applicable for: `change-password`, `subscribe` and `prefer`.
		`id-taken`	Applicable for: `add-teacher`, `add-student` and `set-doodle`.
		`no-such-id`	Applicable for: `get-doodle`, `subscribe` and `prefer`.
		`no-such-slot`	Applicable for: `prefer`.
		`not-subscribed`	Applicable for: `prefer`.
		`no-possible-exam-schedule`	Applicable for: `exam-schedule`.
`LOGIN`	$\rightarrow$	`TOKEN @ TOKEN`	A login is an identifier and a password.
`DOODLE`	$\rightarrow$	`[ SLOTS ]`	A doodle is a list of time slot.
`SLOTS`	$\rightarrow$	`, SLOT SLOTS`	
		ϵ	
`SLOT`	$\rightarrow$	`TIME / TIME`	A time slot is defined by a start time and an end time.
`SCHEDULE`	$\rightarrow$	`{ EXAMS }`	
`EXAMS`	$\rightarrow$	`TOKEN : SLOT , EXAMS`	A schedule is a mapping from exam identifier to time slot.
		ϵ	

Table 1: Client - server exchange protocol. Any white space in the grammar rules can be replaced by a combination of white spaces, tab and new line characters. `TOKEN` is any group of letters uninterrupted by white spaces, tab and new line characters. `TIME` follows to ISO 8601 format – e.g: `2015-11-30T10:28+01:00`.