Software Design Document

Adriaan Leijnse

April 21, 2014

Contents

1 Change history

Version	Comment
1	Iteration 1's basic design
2	Iteration 3's design

2 Introduction

This Software Design Document describes the software architectural design decisions for the Xiast program. This document is geared towards anyone who might have to interact with the Xiast code base, by providing a high-level description of its implementation. It aims to follow the follow the IEEE Std 830-1998 "Standard for information Technology – Systems Design – Software Design Descriptions" standard.

Xiast is developed in an iterative fashion, starting with a minimal list of features to allow developers to familiarise themselves with the technologies used. Over four iterations the feature list will be filled out step by step, fully satisfying the Software Requirements Specifications. Please see the appropriate section in the "Software Project Management Plan" document for a description of each planned iteration.

3 Context

The Xiast program is an HTTP server running a web application accessible via the user's browser. Its goal is to facilitate schedule management for universities.

4 Project structure

Xiast's code is roughly subdivided in two parts: a back end and a front end. The back end includes the scheduler, persistent data management and features for managing communication between users. The front end interfaces with the back end and provides a web-based user interface. It does not contain any significant program logic, but caches schedules to allow off-line use of the application.

Xiast's back end is written in the Clojure programming language [fn::http://clojure.org], which runs on the JVM. Front end code running in the browser will be written in JavaScript. The HTML for the Xiast website is partly statically generated server-side, and partly dynamically generated

using JavaScript. Communication between the two occurs via Xiast's JSON API.

All production Clojure code can be found in the "/src/xiast" directory, and each Clojure namespace has a corresponding file, e.g. `xiast.core` can be found in `core.clj`. Tests for Clojure code are in the "test/xiast" directory. If a namespace has tests written for its functionality the corresponding test namespace has a `-test` suffix. E.g. `xiast.scheduling` has a corresponding `xiast.scheduling-test`, which can be found in "/test/xiast/scheduling_test.clj".

JavaScript, CSS and image files which are directly linked to by the front-end are in "/resources/public". HTML templates can be found in "/templates".

4.1 Front-end

The HTML pages for the front-end are generated server-side in Clojure using the Enlive library, and internationalised using the Tower library. Client-side, the pages – if not complete – are filled in using JavaScript and AJAX calls to the back-end JSON API.

The front-end JavaScript code uses the Twitter Bootstrap [fn::http://getbootstrap.com] and jQuery libraries.

4.2 Back-end

The back-end is written in Clojure and uses a MySQL database for persistent storage. The Compojure and Ring libraries are used for routing and session management. User authentication is done through the VUB's authentication API; sessions are stored in encrypted cookies.

5 Common abstractions in the Xiast code base

Xiast contains a number of abstractions and data types which allow code reuse throughout the program. Some of these data types will be used to transmit information from the back end to the front end and back. The data types are described in the `xiast.schema` namespace (schema.clj).

The full definitions of the following data types can be found in the aforementioned source file:

Courses This data type describes a course, with its name and unambiguous identification string.

Schedule blocks Schedule blocks describe a time span on a specific day and location during which a class is taught, using the VUB's academic hours and calendar conventions. E.g.: Scheme is taught in week 3, on day 1, from 9am till 11am in room E1.03.

Timespans Describe a span of time over multiple days or weeks.

6 Design components

7 Data description, validation and transformation

The `xiast.schema` namespace uses Prismatic's Schema library to describe the various data types used in Xiast. Schema is data description language which also validates data in a contracts-like manner. This validation is turned on during testing, as a first layer of defence against bugs.

Another Schema feature used by Xiast is *data coercion* between formats. This is used in `xiast.api` to convert Clojure (edn[1]) data to and from JSON.

8 The Xiast web server and JSON API

HTTP requests are processed by the Ring[2] and Compojure[3] libraries. Ring handles requests by sending them through a number of handlers (e.g. `wrap-with-session`, see below), while Compojure routes requests to specific parts of the code based on request URLs.

From the `xiast.core` namespace:

```
(def app
  (-> main-routes
      wrap-with-session
      wrap-keyword-params
      wrap-nested-params
      wrap-params
      wrap-multipart-params
      wrap-flash
      (tower.ring/wrap-tower-middleware :fallback-locale :en
                                        :tconfig t/tower-config)
```

[1] https://github.com/edn-format/edn
[2] https://github.com/ring-clojure/ring
[3] https://github.com/weavejester/compojure

```
(wrap-session {:store (cookie-store {:key "Kn4pHR5jxnuo3Bmc"})})
(wrap-resource "public")
(wrap-file-info)))
```

This code is read from top to bottom, as each wrapper is wrapped by the subsequent wrapper.

`xiast.core` contains the routes for the regular HTML pages accessible to users, while `xiast.api` has those which serve the JSON API.

9 Session storage

Session data for logged in users will be stored in the database, indexed by a unique session identifier stored in a cookie. However, the entire session is currently stored in an encrypted cookie. `wrap-with-session` from session.clj makes the session dictionary `*session*` available for the rest of the program.

10 Internationalisation

Internationalisation is accomplished using the Tower library. The `xiast.translate` namespace provides the `translate` and `translate-nodes` functions, which lookup translations in the "/resources/dictionaries/all.clj" dictionary. `translate-nodes` can translate Enlive-nodes (i.e. HTML markup in Clojure form) which have been tagged with a `msg` attribute. E.g.

```
<h1 msg="about/about/title">About Xiast</h1>
```

is transformed into

```
<h1>Over Xiast</h1>
```

when the locale is nl-BE.

11 User authentication

Authentication of users is done via the VUB's authentication API. `xiast.authentication` has two functions: `login` verifies the NetID/password combination, and adds the relevant user rights to the users session. `logout` clears the session.

12 Querying and updating curricular and personal facts

The (internal) public interface for querying and updating data can be found in `xiast.query`. These functions query and update curriculum and personal data, as well as schedules for rooms, courses, students, programs and instructors.

13 The scheduling process

Back-end scheduling code resides in the `xiast.scheduling` namespace, while the front-end JavaScript code dealing with displaying and modifying schedules can be found in "generic_calendar.js".

A schedule is a list of schedule blocks:

```
(def ScheduleBlock
  {(s/optional-key :id) ScheduleBlockID
   :week AcademicWeek
   :day DayNumber
   :first-slot ScheduleSlot
   :last-slot ScheduleSlot
   :item ScheduledCourseActivity
   :room RoomID})

(def ScheduledCourseActivity
  {:type CourseActivityType
   (s/optional-key :title) (s/named s/Str "Course title")
   :course-activity s/Int
   :course-id CourseCode})
```

Xiast is implemented to make working with the existing schedule as easy as possible. To achieve this it has the concept of "schedule proposals", which denote changes against the current schedule.

From `xiast.schema`:

```
(def ScheduleProposal
  {(s/optional-key :new) #{ScheduleBlock}
   (s/optional-key :moved) #{ScheduleBlock}
   (s/optional-key :deleted) #{ScheduleBlockID}})
```

New schedule blocks are those which have been newly created by the front-end. Moved schedule blocks are existing schedule blocks of which the time or room has been changed. Communicating the deletion of schedule blocks is done by listing their schedule block ids. These ids are assigned to all schedule blocks in the currently accepted schedule.

When the back-end receives a schedule proposal, it runs a number of schedule checks against the current schedule in the database (see `check-proposal`):

```
(def ScheduleCheckResult {:type (s/enum :mandatory-course-overlap
                                        :elective-course-overlap
                                        :room-overlap
                                        :instructor-unavailable
                                        :activity-more-than-once-weekly
                                        :room-capacity-unsatisfied
                                        :room-facility-unsatisfied)
                          :concerning #{ScheduleBlock}
                          s/Any s/Any})
```

These check results describe errors and warnings which the user must solve before the schedule proposal can be applied.