An Internship Management Web site

Ioana Manolescu

November 29, 2020

We discuss the design and implementation of a Web application that would enable the management of 3A internships at Ecole Polytechnique.

Below, each concept appears in **boldface** upon its first mention.

1 Outline

Internship proponents (hereafter called just proponents, for brevity) propose internship topics by uploading them in the Web site, with the goal of making them available to the students.

Before the topics are visible to students, **administrative assistants** need to perform an **administrative validation** (yes/no). An administratively validated topic needs then to be **scientifically validated** by a professor. Professors can **assign** a topic to a student when there is enough evidence that the proponent of the topic and the student agree to work together on that topic.

It is of course essential that the Web application users can interact with it from machines distinct from the one where the Web application is running (the only thing the client can assume is an HTTP connection to the machine hosting the Web application).

2 Data model details

2.1 Internship programs and categories

The application supports a set of **internship programs** (program, in short). Each program has a **name** (a short string, e.g., 'Inf592') and a **numeric ID** (automatically assigned by the system). *Throughout the application, all IDs must be numerical and assigned by the system upon insertion.* Each internship proposal also has a year. The pairs (name, year) are all distinct, e.g., there exists only one internship program 'Inf592' for '2020'. The year and name cannot be null.

There exists a set of **categories**, e.g., 'Databases', 'Machine Learning' etc., each of which has an ID and a description (a string), which cannot be null. All the description strings must be distinct. A category can be associated to one or several internship programs. An internship program can be associated to one or more categories. There may be programs not associated to any category and categories not associated to any program.

2.2 Internship topics

Each internship topic:

- Has an ID, a title, a creation date, and a content, which is a PDF file.
- Can be associated to at least one category and possibly more.
- Is associated to exactly one proponent.

2.3 Users and roles

Each user has a name, an ID (assigned by the database), an **email**, and a **password**, which must have at least 8 characters. All the emails must be distinct. Each user also has a **account creation date**.

The database contains the following **roles**:

- **Student**
- **Administrative assistant** (or **assistant**, in short).
- **Professor**
- **Proponent**
- **Web site administrator** (or **admin**, in short).

Each user has at least one role and may have several. Below, we call "a student" (resp., assistant, professor, proponent, admin) any user that has the respective role (regardless of how many other roles the user has).

Each user has a boolean attribute called **valid**. This field will be used to authorize access to the site's operations (see below). *No user attribute can be null.*

Each user may **have access to** to one or more internship programs.

The Web application behavior for users of each role is described below.

A student may be **assigned** to an internship topic. No student can be assigned to more than one internship topic. No topic can assigned more than one student.

2.4 Errors

A table should store every error occurring while the Web application ran: the date and time when the error occurred, the method which caused the error, the user which triggered the method to run, and a description of the error (see also Section 7).

3 Functioning

A general principle is: *whenever the database is modified as a consequence of a user action, the user must be shown the modified state of the database afterward.*

3.1 Creation of an Admin account

One Admin user must be created before launching the Web app, with its valid attribute set to true. *All other necessary changes to the database must be supported by the Web application.*

3.2 Topic upload

A proponent is shown a page that does not require login, and which requires: the proponent's email, a topic (PDF file to be chosen with the help of a navigator in the proponent's local file system), a title, and gives the choice between the existing categories. The proponent must chose one category. When all the input is available:

- If the proponent is not already known, create a new proponent in the database. Proponents are never supposed to log in; thus, fill in their password and valid attribute somehow (it does not matter).
- Save the topic and its association to the category and the proponent.

3.3 Creation and validation of other accounts

Create-account On a page that is accessible without logging in, users can solicit the creation of an account by supplying their names, emails and passwords to the Web site as well as the desired user type (Student, Professor or Assistant - the user must be allowed to chose from a menu). This leads to adding a new user in the database, with valid set to false.
Validate-account The admin has access to a "User management" page showing users (valid or not) ordered by their ID. On this page, an admin can validate a user, at which point the admin must associate a user to at least one program (possibly more). The admin can also invalidate a user, and can modify its set of roles.
Login A user can log in by supplying an ID and a password, which must match those stored in the database, and must correspond to an account whose valid attribute is true.

3.4 Administrative validation of topics

After logging in, administrative assistants must be able to:

Admin-valid See all the topics submitted but not administratively validated. An assistant can select a topic (or several) and validate them.

Admin-invalid See all the topics submitted and administratively validated, but not scientifically validated. Here, an assistant can revert an administrative validation.

Admin-inspect-delete See all the topics, select some topic(s) and ask for their deletion.

3.5 Scientific validation of topics

After logging in, professors must be able to:

Prof-category-program A professor can see all the programs and all categories for each program. A professor can add a program, can add a category, and can associate an existing category with an existing program (the possible categories and programs must be provided in a menu to chose from).

Prof-admin-valid See all the topics submitted but not administratively validated yet. A professor can select a topic (or several) and administratively validate it. A professor can also associate such a topic with one or more categories. A professor can also associate a topic with a program, only if the topic is already associated to a category of that program; otherwise, an error message should be displayed (see Section 7).

Prof-sci-valid See all the topics submitted and administratively validated, but not scientifically validated. A professor can select a topic (or several) and scientifically validate them. A professor can also associate such a topic with a category, and/or with a program.

Prof-sci-invalid See all the topics submitted, administratively and scientifically validated. A professor can revert the scientific and/or the administrative validation.

Prof-assign-topic A professor can assign a topic to a student (the Web site must help the professor locate the topic and locate the student, for example through menus, or approximate keyword search).

4 Students

Aftere logging in, a student should be able to see all the topics which are administratively validated, scientifically validated, and have not already been asigned to another student. The topics could be separated e.g., in one page for each program, and should be shown sorted by ID, possibly also grouped by their first category (but all categories of each topic should be visible).

When the student has an assigned topic, this should be visible to the student in the Web application.

5 Transaction management

Each update to the database must be a transaction for which the ISO **serializable** isolation level should be requested.

6 Session management and log out

While logged in, a user should be shown their own name at the top of each page.

Users should also be able to **explicitly log out**.

7 Error management

In all cases when an error is experienced (e.g., wrong password, invalid account, error for any reason while performing an update etc.):

1. An informative error message must be displayed to the user experiencing the error. This should capture as much as possible of the internal cause of the error.
2. A tuple must be recorded in the errors table, storing the user, date, time, method where the error occurred, and error details.

An admin user also has read-only access to the content of the error table. Errors older than 1 month are automatically deleted.

8 Deliverables and requirements

8.1 Commands needed to create the database

Supply an SQL file (possibly including comments) with all the table creation directives. Keep in mind all constraints listed above. The tables are assumed created, and an admin user inserted, before the application starts running.

8.2 Web application code

The code should be *structured* in a set of Java classes/files, with the *name(s) of the contributor(s) indicated at the top of each file*; a contributor should be able to explain the functioning of their respective part of the code.

Use *human-understandable names* for all the classes, methods, and functions, and if possible variables. Include a *short comment* for each of the main methods.

Use *prepared statements* whenever possible.

A nice (but sober) *look & feel* (obtained for instance through CSS or JavaScript) would be appreciated.

8.3 General remarks

Simple is good. There exist ellaborate frameworks for building Web applications. A framework provides a general mindset in which one just needs to "insert" their application logic. Given the time constraints and for clarity of the code, it is preferrable *not to* use a framework for this project, but instead handle the (relatively simple) application logic in your own classes.

Standard libraries are good. Whenever you have the choice, rely on standard Java libraries, or some from a well-established software provider such as the Apache Foundation.

Use **top-down design**: each user has a set of actions to complete. The choice between the actions should be provided when the user logs in; the user may also change from one action to another. Each action (or a small set of actions) should be handled on one page.