IT2901 - Informatics Project II

IDI Open
Programming Contest System

Haakon Konrad William Aasebø
Håkon Gimnes Kaurel
Tino Hakim Lazreg
Filip Fjuk Egge
Anders Sildnes
Eirik Fosse

May 2014

Norwegian University of Science and Technology

Supervisor: Hong Guo

Foreword

Originally inspired by the Nordic Collegiate Programming Contest (NCPC), it has been held at NTNU every spring since 2007. The format is a five-hour contest with competing teams consisting of one, two or three contestants. A team of volunteer judges write the problems and answer clarification requests during the contest, while another team hands out balloons for each solved problem. Usually a rather hectic affair, it is extremely important that everything is well prepared. The number of teams is often more than 100, with the record being 162 teams in 2011

The contest system that verifies solutions is at the heart of the contest when it is in progress, and needs to be working perfectly at all times. The system must handle several submissions per second, while verifying that each one is correct and runs within the set resource limits. Submissions must show up on the high score list, and when problems are solved the team handing out balloons must be notified. In addition to this there were a lot of other functional requirements having to do with the bureaucracy of organizing the contest

A requirement was that new features could be easily added in the future, and the code was written with this in mind. The project will now become open source, and all programming contest enthusiasts will soon be able to request and implement their desired features

All aspects of this project have been pleasing and delightful for us. The team has exceeded all our expectations and their system will be used for years to come.

Preface

Before there were computers, there were algorithms. But now that there are computers, there are even more algorithms, and algorithms lie at the heart of computing. Designing a system for eager students to hone their skill in the heart of computing has been a true joy

Our group never wanted to settle for adequacy and mere requisiteness. For the past few months, weve taught ourselves a new programming language and framework and used advanced development frameworks - while tackling many social and technical conflicts.

We have ve proven how Ambition is a dream with a V8 engine, as Elvis Presley once said.

The group would like to thank our eager customers, Finn Inderhaug Holme, Christian Chavez and Christian Neverdal Jonassen for their time to meet us and provide constructive feedback. We also owe a big thanks to our supervisor, Hong Guo, for constructive criticism and reflections; without which, we would not ascertain the peak of our own potential

Contents

Chapter 1

Architecture

This chapter contains an overview over the architecture for the system. The first part will describe different views of the system and the second part will show the quality attributes and patterns used when developing the system.

The main parts of the system is shown in Figure 1.1. Clients sends requests to the web server and receives the processed results. The execution nodes process user submissions, and updates the results to the database.

1.1 Views

We have chosen to depict the architecture using Philippe Kruchten's 4+1 view model. [1] This is a method of describing the architecture for software-intensive systems from the viewpoint of different stakeholder by using multiple, concurrent views. We chose this model because it gives a good overview and is widely accepted by the software industry. Below are the 4 main views in the model; Logic, Process, Development, and Physical. The "+1" view is Use Cases which is addressed in [Chapter 2 Task Description and Overview]

1.1.1 Logic View

The logical view describes the functionality of the system by breaking down requirements into classes and representing them, and their relations, through class and sequence diagrams.

Figure 6.1 shows the main classes involved in GentleIDI. Each team participates in a single contest, and consists of a predefined number of contestants. Each team also has a team leader that handles most of the administrative tasks. The team can also try to solve problems by uploading submissions.

1.1.2 Process View

The process view explains the communication between different processes in the system, as well as how the system behaves in runtime.

As this system is a web application the first thing to note is that there will be concurrent users in runtime. Each user generates HTTP requests to the server, which in turn may execute database

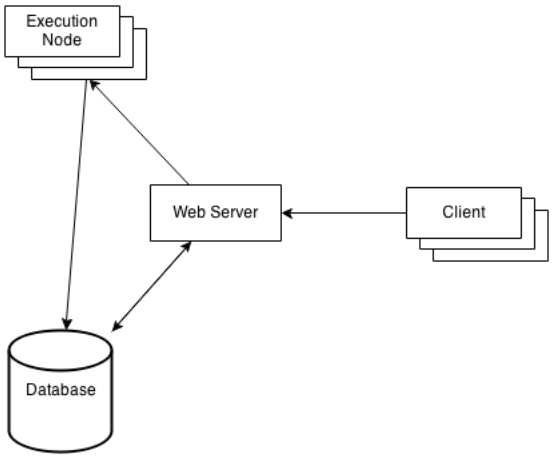


Figure 1.1: System overview

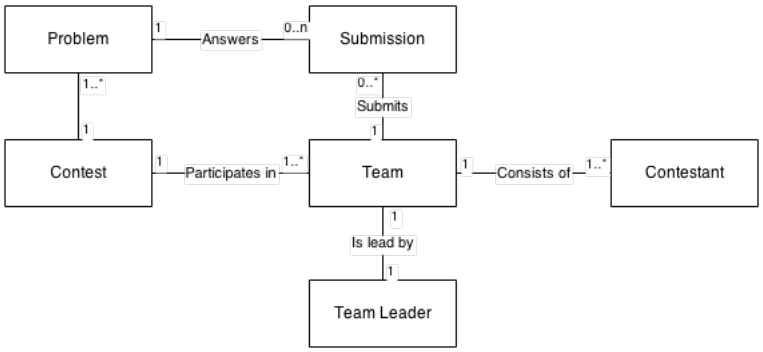


Figure 1.2: Top level class diagram

lookups for information like score tables or problem sets. When a user submits a solution the system will place it in a queue, which decides which node the solution will execute on according to availability and load.

We will now show examples for two important parts of the application. First is the action of successfully registering a user and creating a team. See figure 1.4.

Second is submitting a solution to a programming problem. See figure 1.5.

1.1.3 Development View

Purpose

The developer view is intended for the developers. It should ease development, and focus on software module organization by packaging the software in small chunks.

We wanted a modular and maintainable system where it is easy to maintain and change specific parts of the system without changing everything. The structure of the system can therefore be

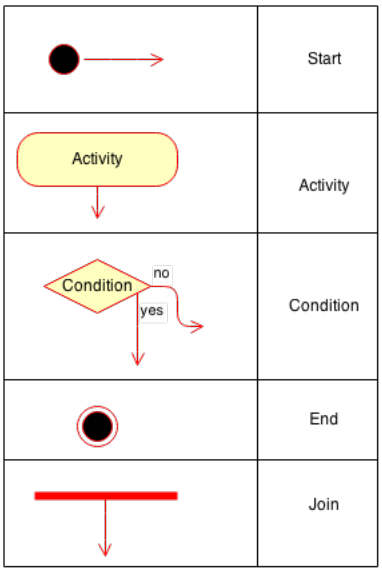


Figure 1.3: Symbology

divided into the following main packages: Contest, Registration, Submission, Execution, Balloon, Clarification, Admin, and Article. These packages are described in detail in chapter 8 Implementation.

1.1.4 Physical View

Purpose

The physical view shows the interaction between the physical components of the system.

Physically the system is structured as a multitiered architecture. It consists of three tires, presentation tier, application tier, and data tier, see figure 1.6. The tiers represents a physical structuring mechanism for the system infrastructure. The user is physically separate from the application and database.

Presentation tier

This tier presents information to the user through the public website and admin interface. It translates the web server response into web pages generated using HTML5, CSS, Ajax, and JavaScript. It sends requests to the underlying web server and renders the response.

Application tier

The application tier contains the logical layer, it controls an application's functionality by performing detailed processing. Primarily this is done through python code, although when running solutions the file is run on an execution node through the use of built in unix commands.

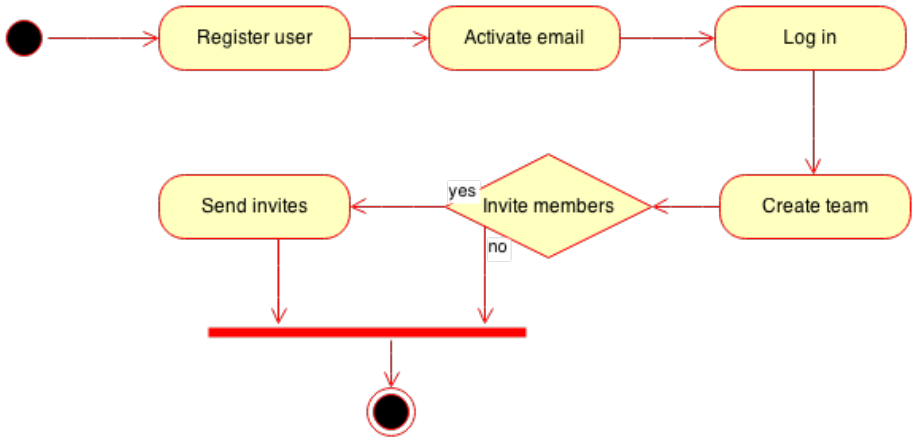


Figure 1.4: Activity Diagram for registering a user and a team

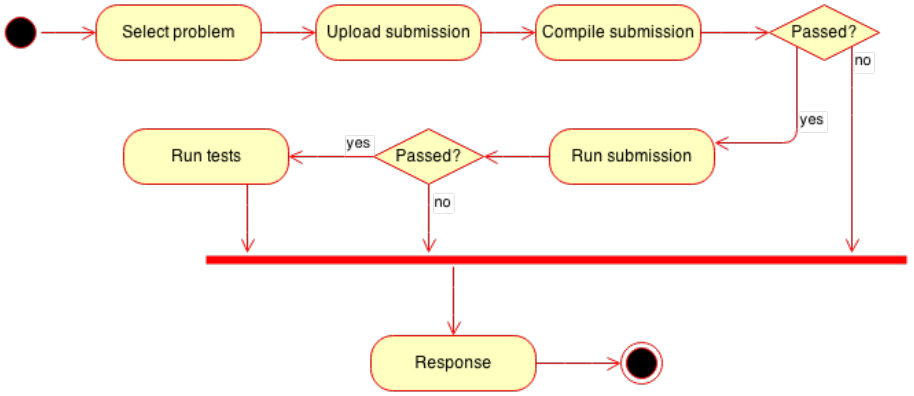


Figure 1.5: Activity Diagram for submittion a solution

This splits the application tier in two parts, the web server that serves static and dynamic content, and the execution nodes that process uploaded submissions. This division can be seen in "Application Tier" in Figure 1.6.

For the web server we use Nginx for serving static files, and as a reverse proxy for Gunicorn, the server providing dynamic HTTP content to the user. Gunicorn is the server that processes requests and returns HTTP pages. The execution nodes process submissions through a FIFO queue implemented with Celery and RabbitMQ. This provides load balancing across CPU cores and multiple nodes in the cluster. The execution nodes also share parts of the filesystem, this is implemented with SSHFS (SSH Filesystem), and is a secure way of sharing the uploaded files across the execution nodes.

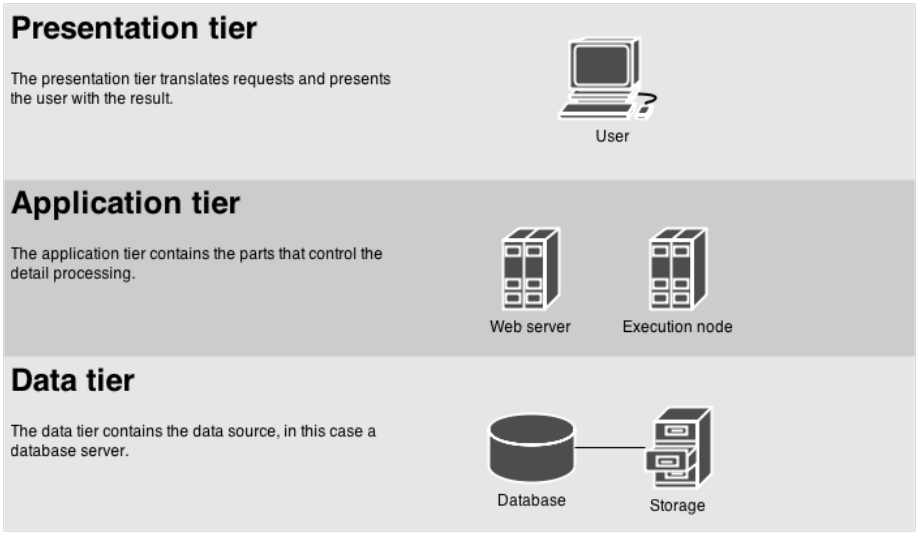


Figure 1.6: Multitier architecture

Data tier

This tier includes the data control functionality. The system utilises a shared SQL database for the execution nodes and the web server. See Figure 1.1. This database links to the file storage on the main web server. However, the execution nodes requires some files to be shared across multiple nodes. Like explained earlier in section 1.1.4, this is implemented with SSHFS. For more specific details see **??** and 2.

1.2 Quality attributes

1.2.1 Availability

Since this software is to be used in a programming contest, it is crucial that the system has high uptime and availability. And since the contest only lasts for about 4 hours, our margin for failure is minimal. We have made an effort to account for all possible outcomes, and to safeguard the application for any errors that might occur.

1.2.2 Modifiability

This is a system that we hope will be used for many years to come. With the ever changing nature of the web, the ability to adapt and improve is imperative. To accommodate this, we chose to implement our solution in Python, a language taught to most of new students of computer courses at NTNU. These are the same students that hopefully will use and continue to work on this software. To our best ability we have also tried to write and document the code in a way such that it is easy to understand and improve.

1.2.3 Performance

Performance is an important aspect of every application, especially web applications. Users expect that sites loads fast. Failing to accomplish this is a sign of a bad application, at least from the user's perspective. For this reason we have focused on making our pages load as fast as possible. And since this application will be used by over 100 users simultaneously, it is also important that the servers will handle the load.

1.2.4 Security

Since our application contains user data and data that should be hidden from unauthenticated users, security is another important aspect. Django provides many security features by default, and others that can be implemented with very little effort. We also chose to enable SSL on the web server to increase security on web requests.

1.2.5 Testability

When we first started out, we wanted to utilize testing during development. Testing is a way to find problems early, and before they begin to encompass larger parts of the application. But testing is also one of the most time consuming parts of the development process. In the end we did not have as much test coverage as we would like, but we feel that we covered the most important parts.

1.2.6 Usability

As with any web application, we want the users of the system to accomplish their desired task, and learn the functions of the system with ease. The user should receive feedback if something went wrong or if the outcome is not clear. We also want the web pages to provide information how to use the system.

1.3 Patterns

1.3.1 Client-Server

Since we are making a web application we will use the Client-Server pattern. The clients connect to the server through a web interface, either the website or the admin interface.

1.3.2 MVC(model-view-controller)

The front end is implemented using the Django framework and follows a rather strict implementation of MVC. Every HTTP request sent to the site is handled by a controller function, which in turn fetches the appropriate models from a database, creates a view based on the models and returns the view as an HTTP response.

1.3.3 Shared-Data

The system utilises multiple execution nodes as well as a web server, through which users access data. We wanted to have a central shared database server that scales with the number of execution nodes and the amount of data.

1.3.4 Multi-tier

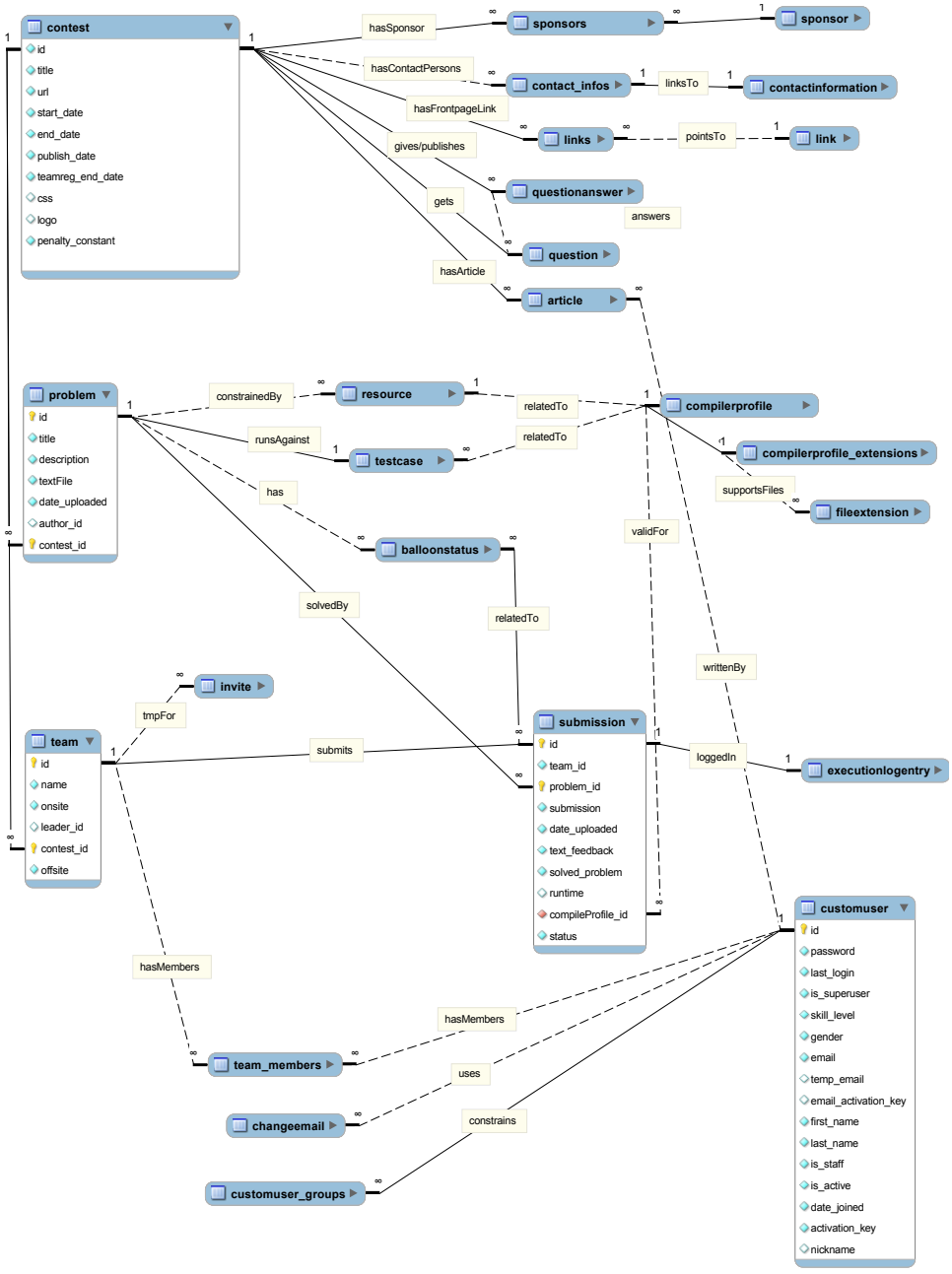
See: 1.1.4 Physical View. References:

[1] Architectural Views -

Chapter 2

ER-Diagram

Our ER-diagrams follows a convention ER-convention. Each relation has a name, and is intended to be read either from left to right or top towards bottom.


contest
id
title
url
start_date
end_date
publish_date
teamreg_end_date
css
logo
penalty_constant
hasSponsor
sponsors
sponsor
hasContactPersons
contact_infos
linksTo
contactinformation
hasFrontpageLink
links
pointsTo
link
gives/publishes
questionanswer
answers
gets
question
hasArticle
article
problem
id
title
description
textFile
date_uploaded
author_id
contest_id
constrainedBy
resource
relatedTo
compilerprofile
runsAgainst
testcase
relatedTo
compilerprofile_extensions
supportsFiles
fileextension
has
balloonstatus
validFor
solvedBy
relatedTo
writtenBy
invite
tmpFor
team
id
name
onsite
leader_id
contest_id
offsite
submits
submission
id
team_id
problem_id
submission
date_uploaded
text_feedback
solved_problem
runtime
compileProfile_id
status
loggedIn
executionlogentry
customuser
id
password
last_login
is_superuser
skill_level
gender
email
temp_email
email_activation_key
first_name
last_name
is_staff
is_active
date_joined
activation_key
nickname
hasMembers
hasMembers
team_members
uses
changeemail
constrains
customuser_groups

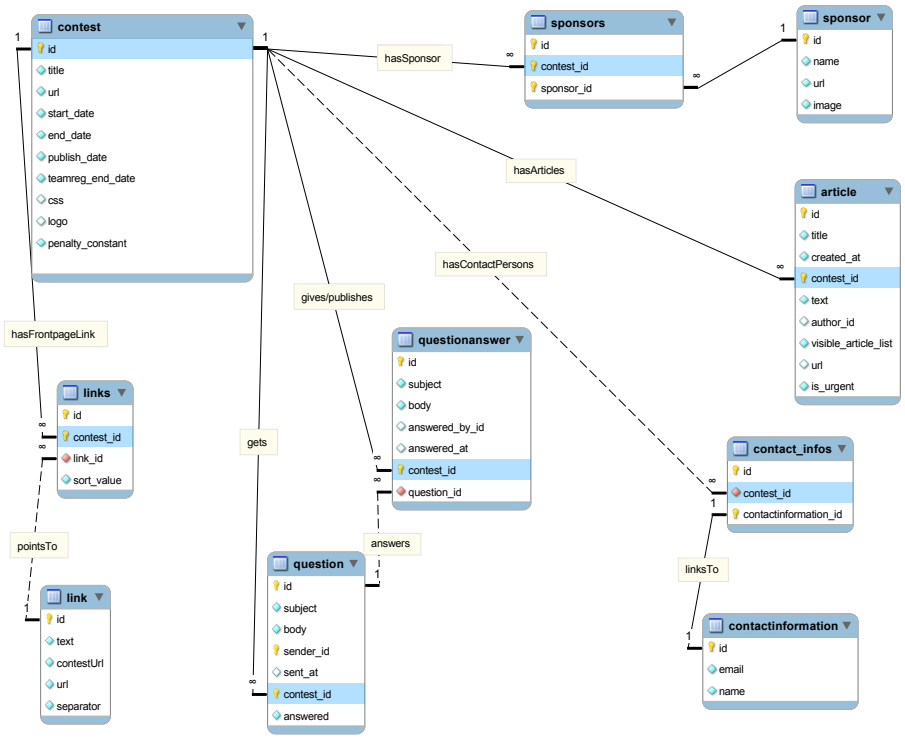


Figure 2.2: ER-diagram for the models used for the contest.

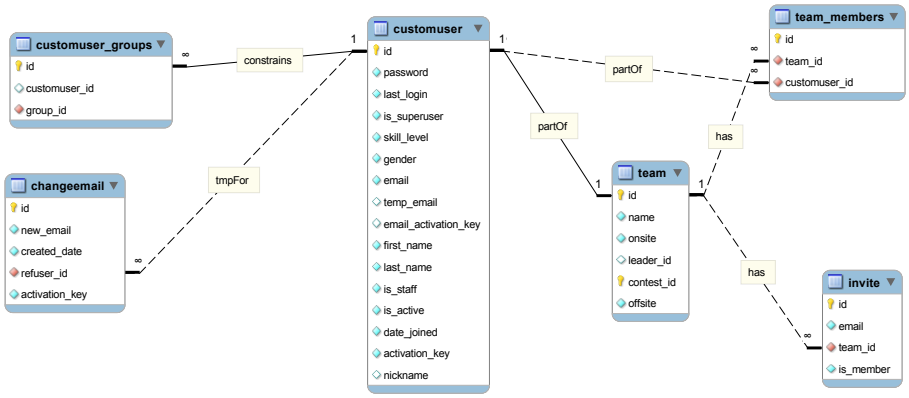


Figure 2.3: ER-diagram for the models used for the user registration.

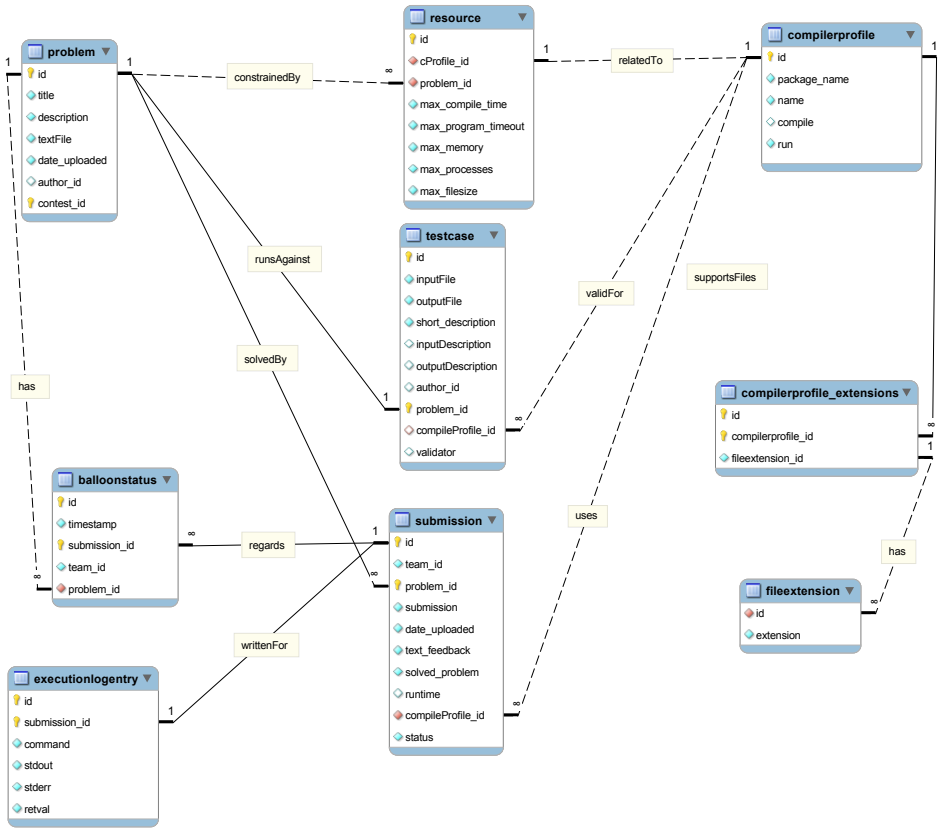


Figure 2.4: ER-diagram for the models used for the submission.