Group Project Report: Robin

Robert Bastian
Paavan Buddhdev
Aiken Cairnscross
James Dai
Alistair Gavin

Supervisor: Ivo Sluganovic
Corporate Sponsor: *Palantir*

University of Oxford

May 2015

Contents

Chapter 1

Specification and planning

In this section we will outline the brief we were given, our initial ideas for what project to do, a flow-diagram for our chosen idea, and our plan of action. At the end, a full project specification is included. Full details about the implementation will be covered in the second half of this report.

1.1 The brief

The following was the brief that we selected from the list of possible projects:

> Project K: Gamification
> There have been many attempts to improve the way that key skills are developed throughout a programmer's career, and one of the ways that has proven to be highly effective is gamification. When done correctly, gamification can provide motivation to learn new skills, or hone existing ones. The challenge is to create a group game that develops one or more core skills that you consider to be important to programmers. The game can be collaborative or competitive, but it must be easy to set up and most importantly fun. If you have an awesome idea that requires a little hardware, Palantir will happily cover any reasonable costs.
> Example project: "Security Capture the Flag". Write a web server with typical security vulnerabilities and install it on a Raspberry Pi. Players compete against each other to gain the most complete control of the Pi in the fastest time. In this example, real time competition provides an extra incentive for players to do well (a common example of gamification). Global leaderboards and bonuses for completing optional challenges can also motivate players, and reward players for focusing efforts towards a particular skill (for example, providing a bonus for responding to a new defence within a time limit, to encourage players to write maintainable, extendable code).

1.2 Our initial ideas

From this brief, we had a lot of ideas coming out of our initial discussions

'Kevin' / Minigames
: The player controls a character travelling through a 2D world, whose progress is determined by the player solving simple programming challenges that are encountered by Kevin within different minigames. Optional paths are also available, so that players of different skill levels are also challenged.

Digital systems game
: The player builds logic circuits to solve progressively more difficult challenges. At each level the player has access to components that had previously been built, which helps to teach different layers of abstractions.

'Extreme startup' style game
: The players would write software to interpret and reply to requests from a central server. The system would award points to each player for every correct response, and deduct for incorrect responses or errors.

Classroom Pi
: The users learn basic programming concepts (for loops, conditional statements, etc.) by programming on a Raspberry Pi. This idea was intended to be used in a classroom environment, where the students would learn the different concepts while working in teams / small groups, whilst within a supervised environment.

'The Elevator Game'
: The player programs one or more elevators. The elevators are required to transport a certain number of people to the floor they each want to go to in the quickest time possible. Each successive level increases in complexity, e.g. more people, fewer elevators, stricter weight restrictions, etc.

The round-robin challenge
: Teams of players create challenges for each other to solve within a limited time-frame. After the expiry date, the challenge setter rates each of the solutions, and the other players rate the challenge. A new challenge is then chosen, and the game moves on in a round-robin format. The challenges are language-independent.

1.3 Our chosen idea (and the reasons why we chose it)

Our chosen idea was the round-robin challenge system. We had three main reasons for choosing it, as opposed to the other ideas:

1. The other ideas, while all involving the user playing a game, were not as directly focused on **competitiveness** - any competition was implemented by using high scores and best times. The round-robin idea involved directly challenging other players, which we think is the best way to keep people interested and motivated and thus the best way for users to learn more about programming.

2. By leaving the details of challenges up to the players, there was a lot of **scope** for players to go where they wanted. Teams of experienced developers have the ability set more difficult challenges for each other.

3. Moreover, implementing this idea of a **'metagame'** allowed us to focus on building a framework for setting challenges for different users using a server instead of getting bogged down in specific details, which was something we were all keen to try out.

While the idea may initially seem quite simple, we were wary of being too ambitious, as few of us had worked on a project like this before.

1.4 Proposed lifecycle of a challenge

The following diagram was our conceptual run-through of how one round of our round-robin tournament would go. In this diagram the actors are underlined, and any potential ideas, advanced features or undecided design decisions are italicized.

1. While no challenge is active, the players can submit a challenge
    - Each challenge comprises of
        - Title, description, sample dataset / sample input
        - A grading method
            - By default the Question Master grades each solution
            - *Or could be automatically graded? Or the solvers grade (peer reviewed)*
        - Expiry date
    - *Can challenge be changed if a problem is found with it? How is this dealt with?*
    - *Should we allow a challenge be added to the queue whenever or only when none active?*
2. A Question Master is chosen
    - The Question Master has to be someone who has submitted a challenge.
    - *Chosen in a round-robin order? Can users choose to pass?*

- Chosen based on points from last round / winner of last round?
- *Chosen by random allocation?*

3. Everyone else becomes a solver. Each solver:
   - Gets a challenge and solve the challenge
   - *Submits their solution as code? Or as text? Sent by email or through a web app or directly to a server?*
   - *A more advanced: allow solvers to modify or resubmit solutions*
4. Solvers grade the challenge, and the question-master grades the solutions
   - Grading of challenge by submitters
   - *A more advanced way would grade by combining ratings in the categories*
     - *Complexity*
     - *Ingenuity/technical*
     - *Fun factor*
   - Grading of solutions
   - *If consensus is that challenge is bad (e.g. 40% down-votes) then scrap challenge?*
   - *Questions / challenges posed to challenger? Is this done irl or in the system?*
5. Points allocated to
   - Solvers and challengers
     - *Include a bias to incentivise people to choose to set challenges – setting challenges gets more points / placed at top of board?*
6. Leaderboard updated with points of each participant
7. Challenge archived

1.5 Use cases

The main use cases are outlined in Figures 1.1-1.4

Primary actor Users

Goal To log in

Main Success Scenario

1. The user enters their username and password
2. The server checks if the username is a correct username and that the password entered matches the user's password
3. The user is logged in

Extensions If the username or password is not correct, the server informs the user to retry

Figure 1.1: Logging in

Primary actor Question Master

Goal To set a challenge for others to solve if none already is

Stakeholders

1. Question Master: wants to set the challenge
2. Solvers: wants to have a challenge to solve

Trigger a user gains the right to set a challenge

Main Success Scenario

1. The QM decides on the challenge
2. The QM inputs the challenge into the web app and selects an end time

Figure 1.2: Setting a challenge

Primary actor Solvers

Goal To submit a solution to the current challenge

Stakeholders

1. Question Master: wants to have their challenge solved
2. Solvers: wants to solve the challenge

Success guarantees The solution is submitted to the server

Trigger The QM sets a question

Main Success Scenario

1. The solver reads the question and decides upon a solution
2. The solver types their solution and submits it

Figure 1.3: Submitting a solution

Primary actor Users

Goal To score the challenges and solutions

Stakeholders

1. Question Master: wants to be able to rate solutions and have their challenge rated
2. Solvers: wants to rate the challenge and have their solutions rated

Success guarantees Scores are submitted successfully

Trigger The current question expires

Main Success Scenario

1. Each solver has to rate the challenge
2. The QM rates each individual solution

Figure 1.4: Scoring

1.6 Collaboration methods and group roles

We decided to use *Slack* as our main communication method. *Slack* is a multi-platform program which allowed us to converse, and organise our conversations in different channels, each with a specific purpose. This allowed us to discuss the tasks that needed to be done separately, which different was really useful when working on separate modules of the application.

We used the *Git* version control system to collaborate on the code. It was essential as it allowed us to keep working on the same project when not in the same place, which was especially important during the break, when we were all very far away from each other and had very different schedules.

After initially considering using *Asana* to organise tasks, we decided that it would overcomplicate things, and that *Slack* and *Git* were sufficient for our needs.

Chapter 2

Design and Implementation

After we had our specification and our plan, our next step was to implement the various components (e.g. the web interface, the Rails server, the structure for setting challenges, the structure for rating challenges and solutions, etc.). This second half of this report shall explain how we planned these components, and then integrated them together. It will also go over how we tested the system and the challenges we encountered.

2.1 Design/layout mockups

Figures 2.1-2.3 are some of the initial drawings from one of our first sessions for what the site structure will look like.

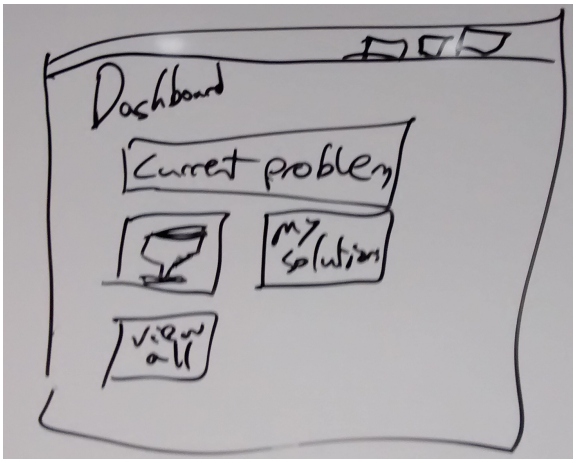

Figure 2.1: Dashboard

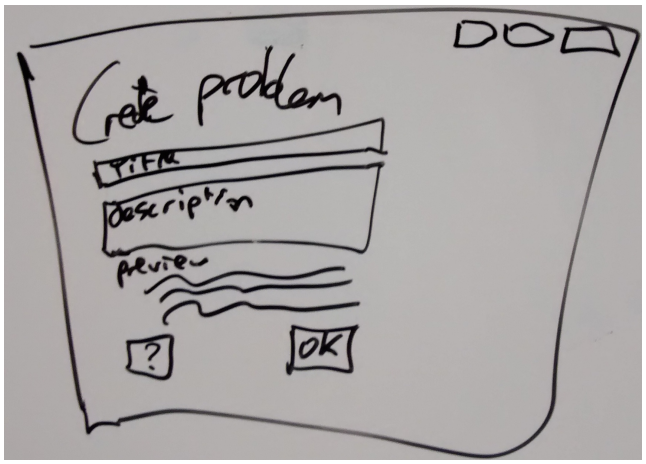

Figure 2.2: Challenge submission

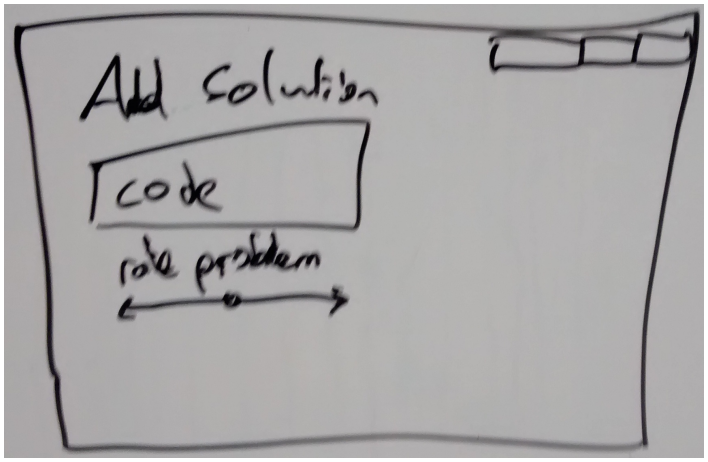

Figure 2.3: Solution submission

2.2 Site structure and state diagram

Figures 2.4-2.6 are the initial outlines for the app's database (model) and its website structure (views).

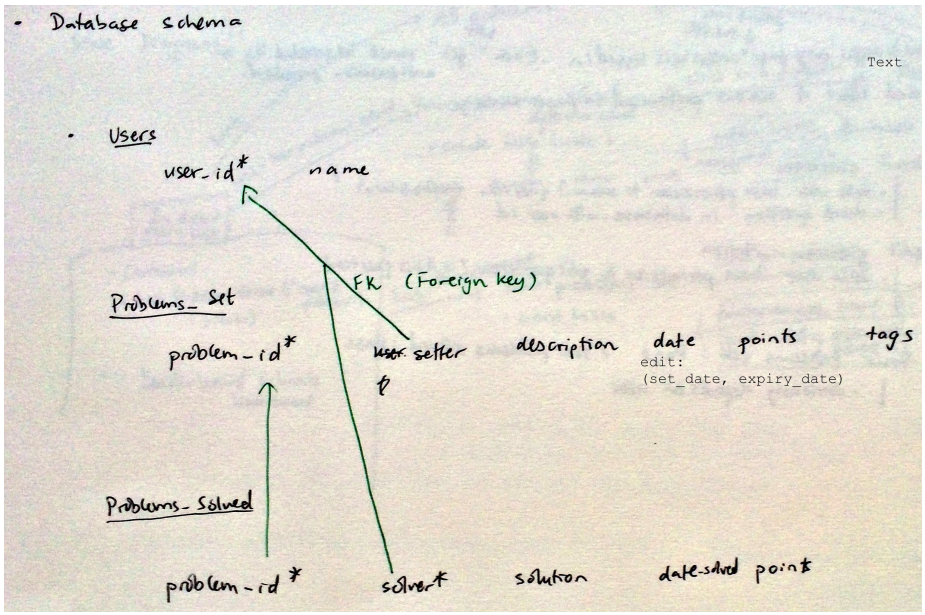

Figure 2.4: Database schema

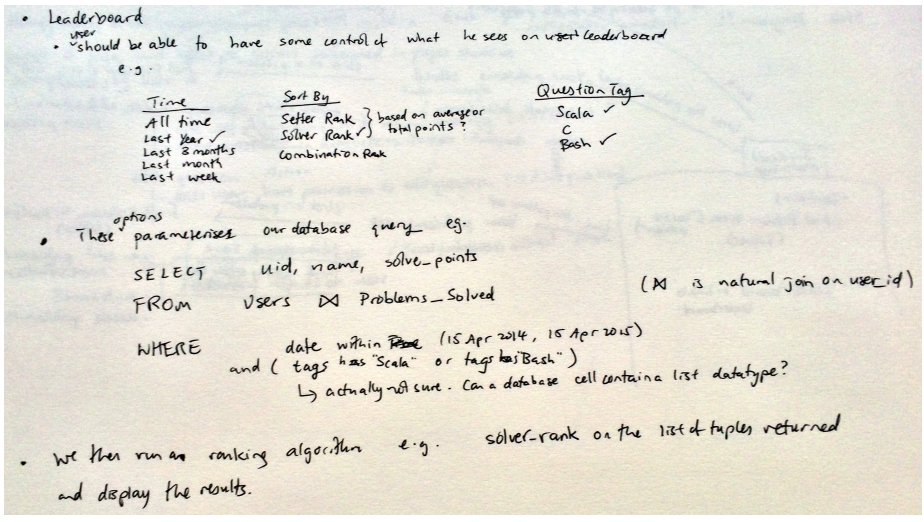

Figure 2.5: Leaderboard functionality

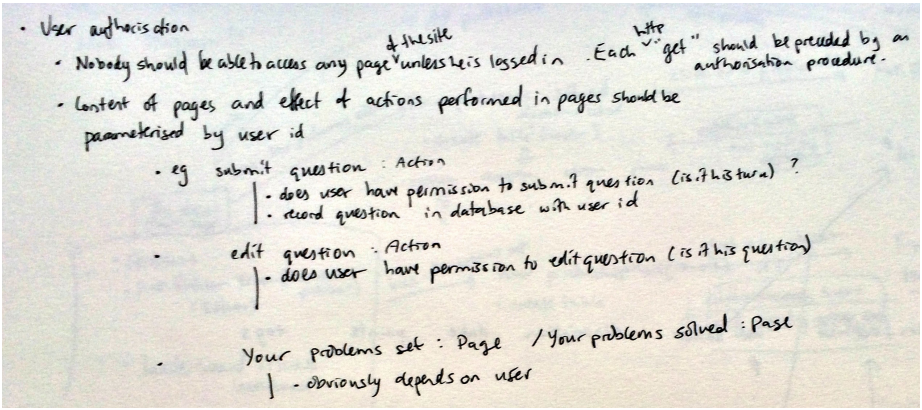

Figure 2.6: User authorization

2.3 States

There are 4 main states. The application uses the methods outline in Figure 2.7 to decide which state it is in.

Anyone is able to submit a challenge
: If there is no previous challenge then anyone can submit a challenge. This is the current state if `active_problem?` is false and `most_recent_problem` is `nil`

The QM can set a challenge
: If the previous challenge has ended then the winner of that challenge can set the next challenge. This is the current state if `most_recent_problem.winner` is not `nil` and `active_problem?` is false

The solvers can submit solutions
: If the QM has submitted a challenge and it has not ended yet, the solvers can all submit solutions. One solver can submit one solution. This is the state if `active_problem?` is true.

The QM can grade the solutions
: When the challenge has ended the QM can grade the solutions. The challenge ends when either it expires (using the date given by the QM) or when all the solvers have submitted solutions. This is the state if `active_problem?` is false and `most_recent_problem.winner` is `nil`

```
module ProblemHelper
    def active_problem?
        ps = Problem.order(:created_at => :desc)
        if ps.size == 0 then false else ps.first.expiry > DateTime.now end
    end

    def most_recent_problem
        Problem.order(:created_at => :desc).first
    end
end
```

Figure 2.7: The `ProblemHelper` class used to determine the app's state

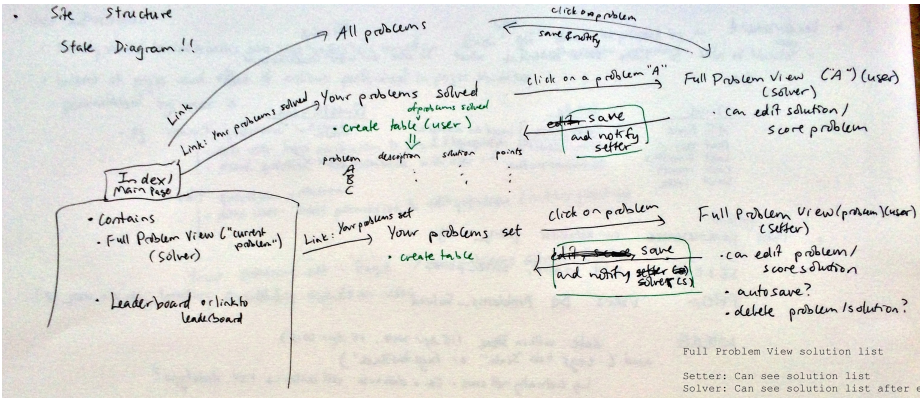

Figure 2.8: Proposed state diagram

2.4 Back end design

Initially we looked into using a *Scala*-run server, but quickly we realised that the database library (*Slick*) wasn't well documented and the server itself would be very slow. Thus we changed to using *Ruby on Rails* and *SQLite*.

Rails is incredible powerful and versatile, but most importantly, it has been used for thousands and thousands of applications already. This means that there is a lot of documentation and support for common issues, as well as quick-start guides to help us get up and running in the short amount of time we had. Moreover, all of us were new to the *Ruby* programming language and so were excited to try it out.

2.5 Front end design

Initially we tried building the website using a basic responsive framework *Skeleton* (`http://getskeleton.com`) but soon realised that it would take too long to write the CSS for everything, so we moved to Twitter's emphBootstrap (`http://getbootstrap.com`), which is more complete. We also are using *FontAwesome* icons to improve usability.

As the website is built on a responsive framework, it is fully compatible to view on a mobile website, although the expectation is that users would likely use it from a computer as they can type out challenges and solutions more easily.

2.6 Tests

Throughout building the application we have been testing with sample problems and solutions to get everything working. When it was complete, we allowed seven real-world (non-CS!) testers onto our server to try it out, and were given feedback praising the design and its ease-of-use. A few bugs (regarding mobile website sizing and a link on the FAQ page) were also squashed.

2.7 Extensions

One extension that would help would be to create individual domains for different companies or groups to use, in the same way that *Slack* does. This would allow more specialised questions for the companies using them.

We have a template in place on the dashboard where awards would go, but have not implemented them. Awards for a coding challenge would for example be 'Best Comments', 'Most Concise Code', etc. Implementing an awards system would help by increasing competition between users.

Appendix A

Desktop screenshots

Chrome 42.0.2311
Microsoft Windows 8
`http://robin.robertbastian.de`

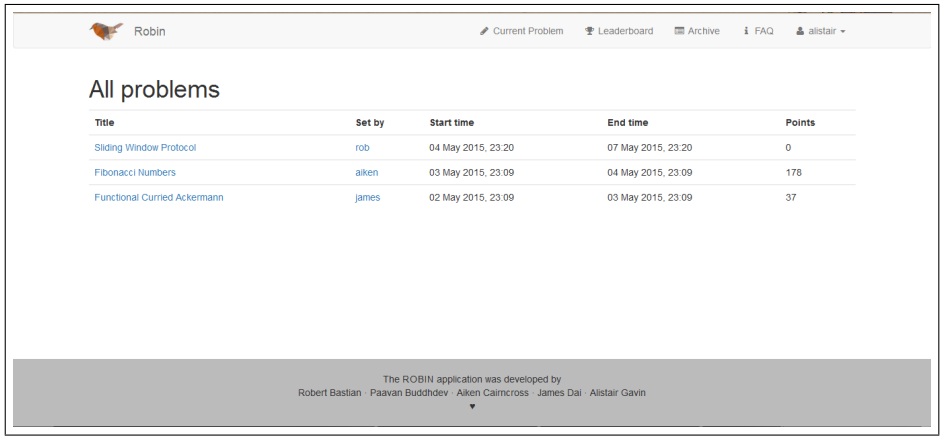

Figure A.1: Archive

Figure A.2: Solution creation page

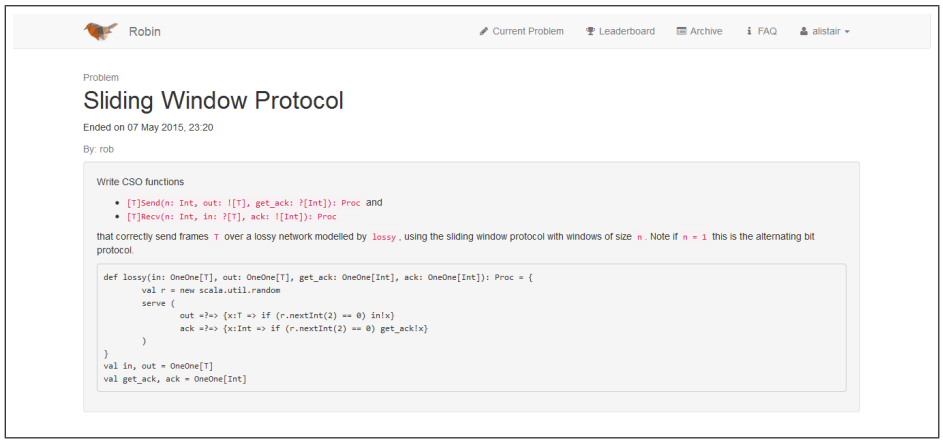

Figure A.3: Current problem

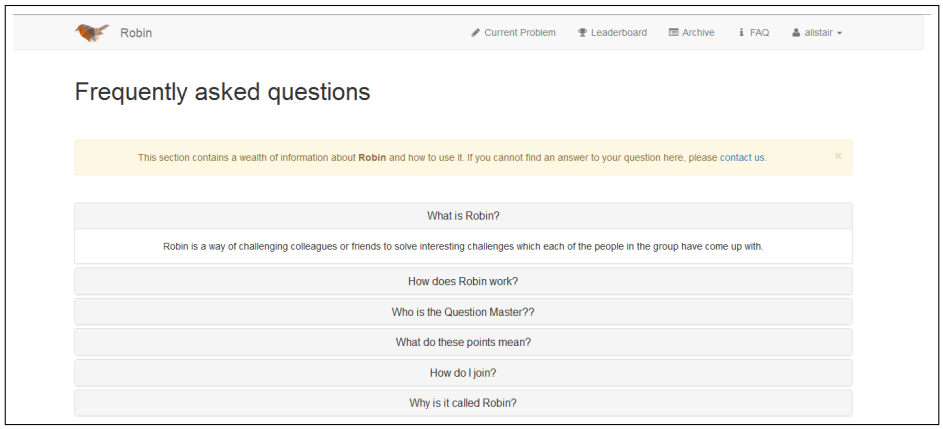

Figure A.4: Frequently asked questions

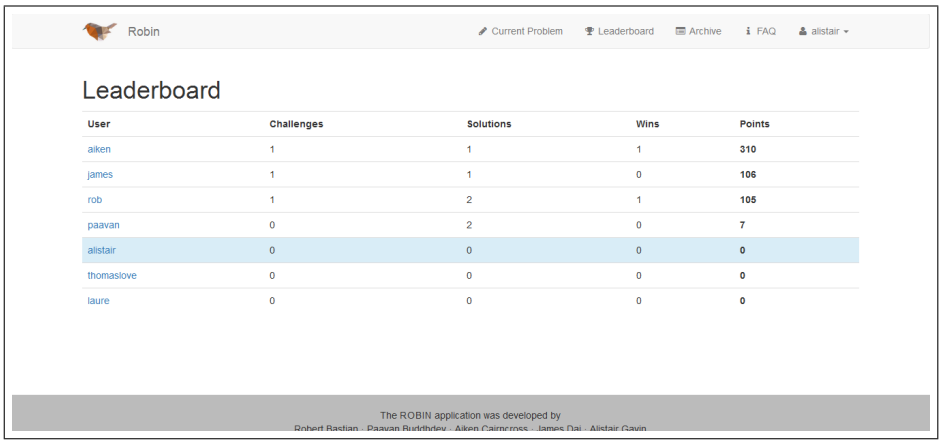

Figure A.5: Leaderboard

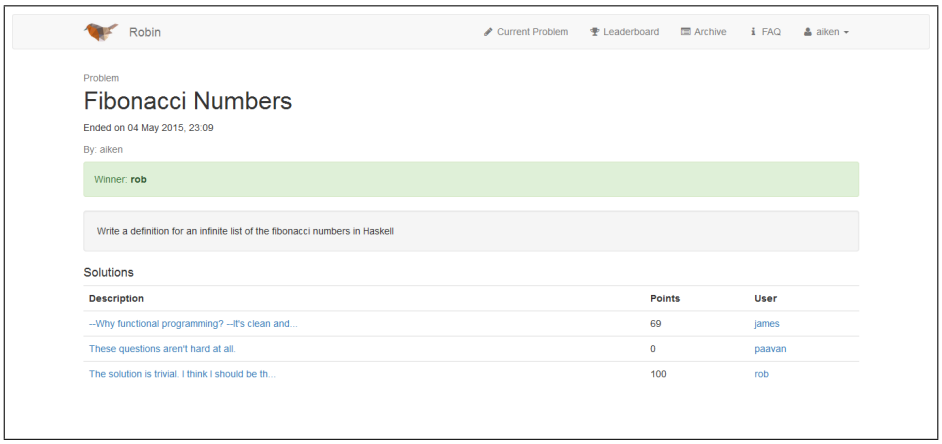

Figure A.6: Finished challenge

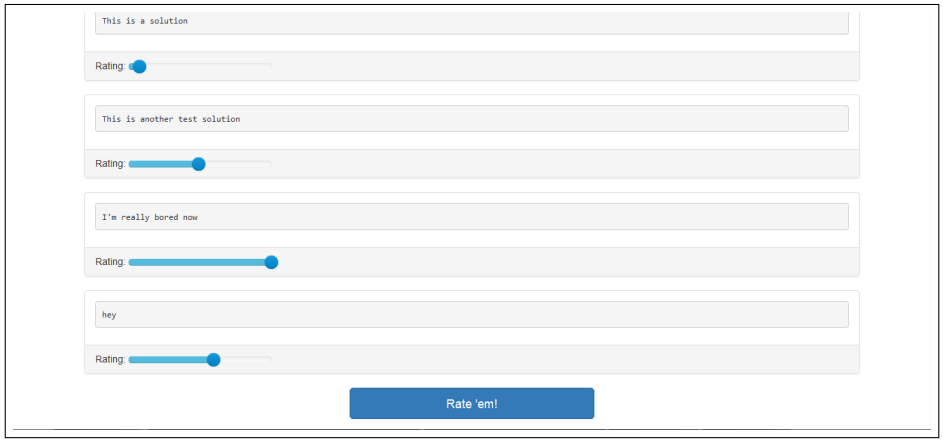

Figure A.7: Challenge marking

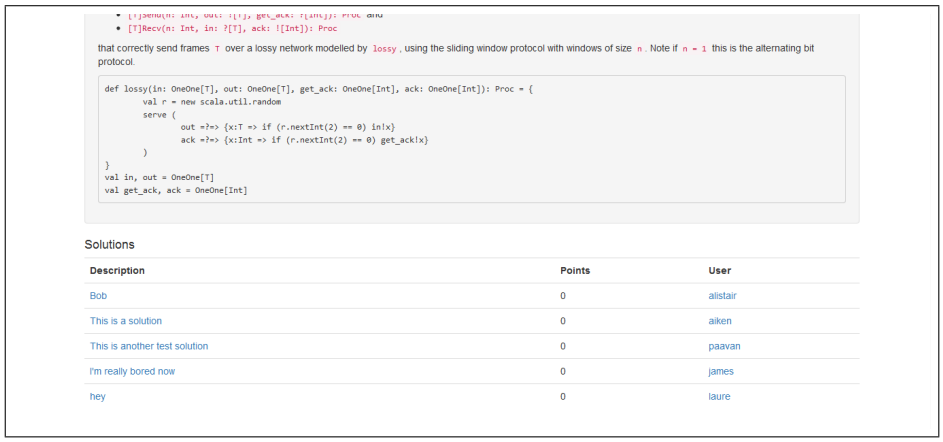

Figure A.8: Question Master's view of a challenge

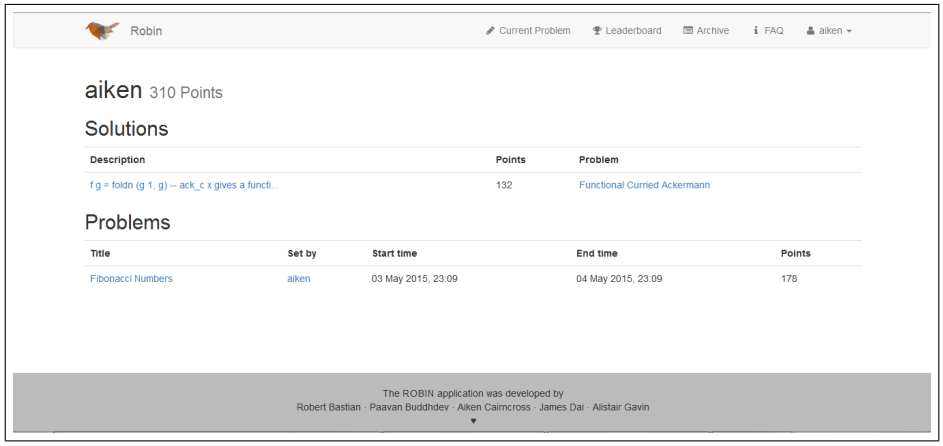

Figure A.9: Profile

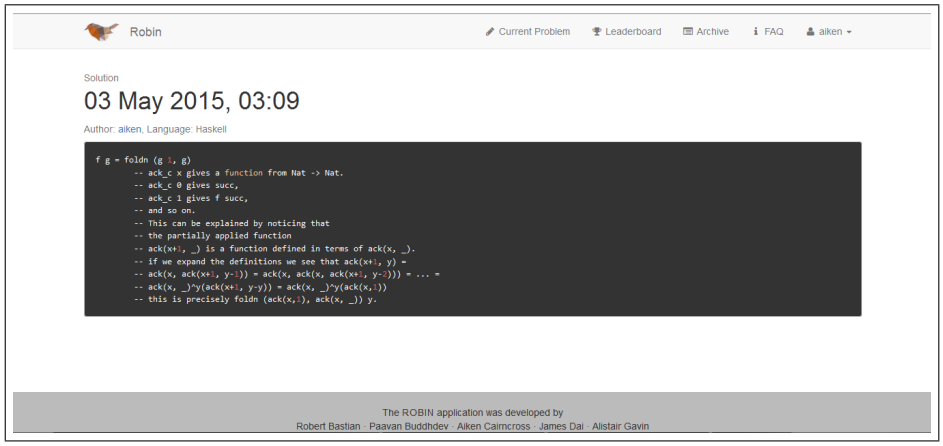

Figure A.10: Solution

Appendix B

Mobile screenshots

Chrome 42.0.2311.47
iOS 8/iPhone 5
`http://robin.robertbastian.de`

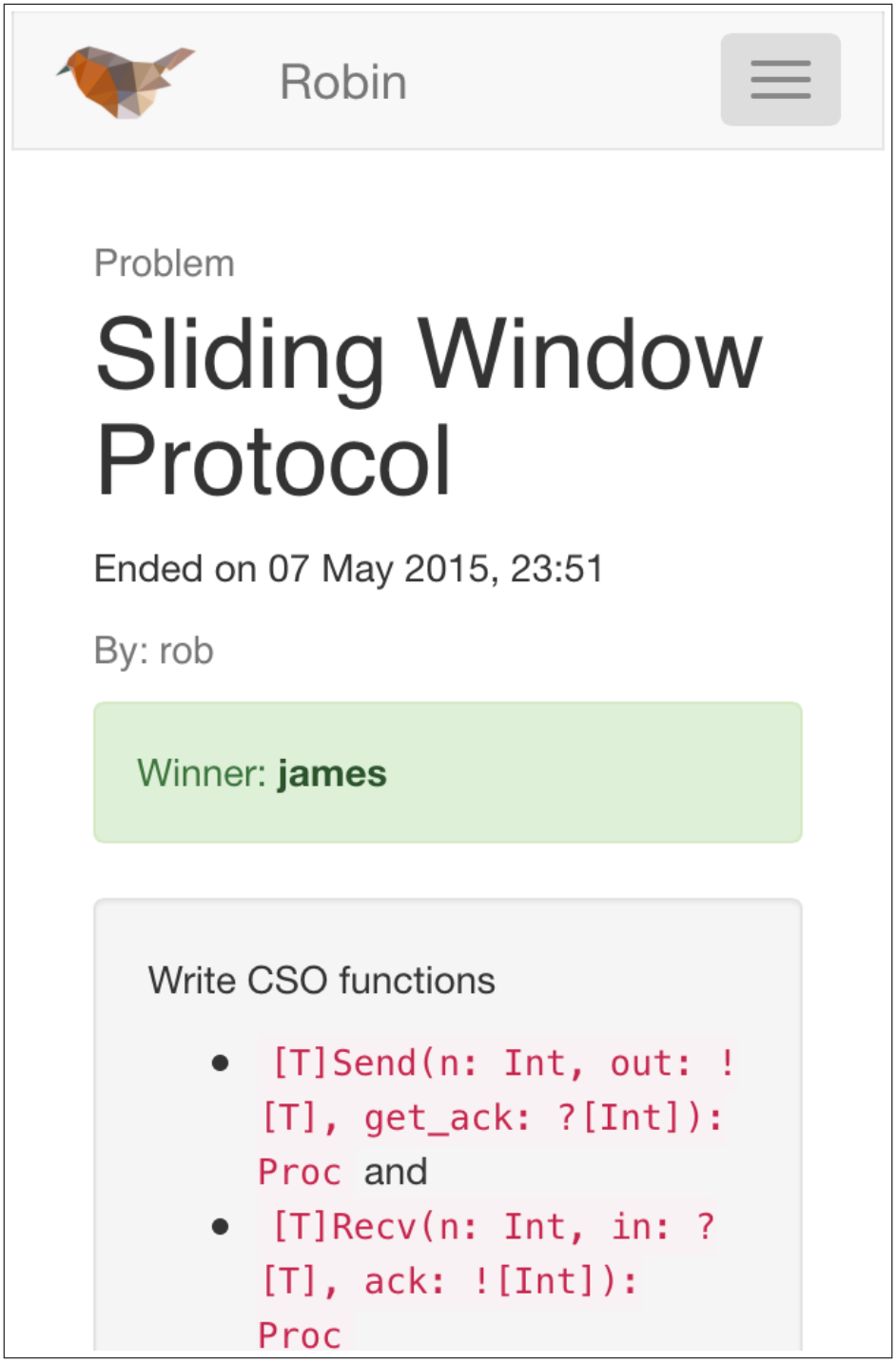

Figure B.1: Problem page

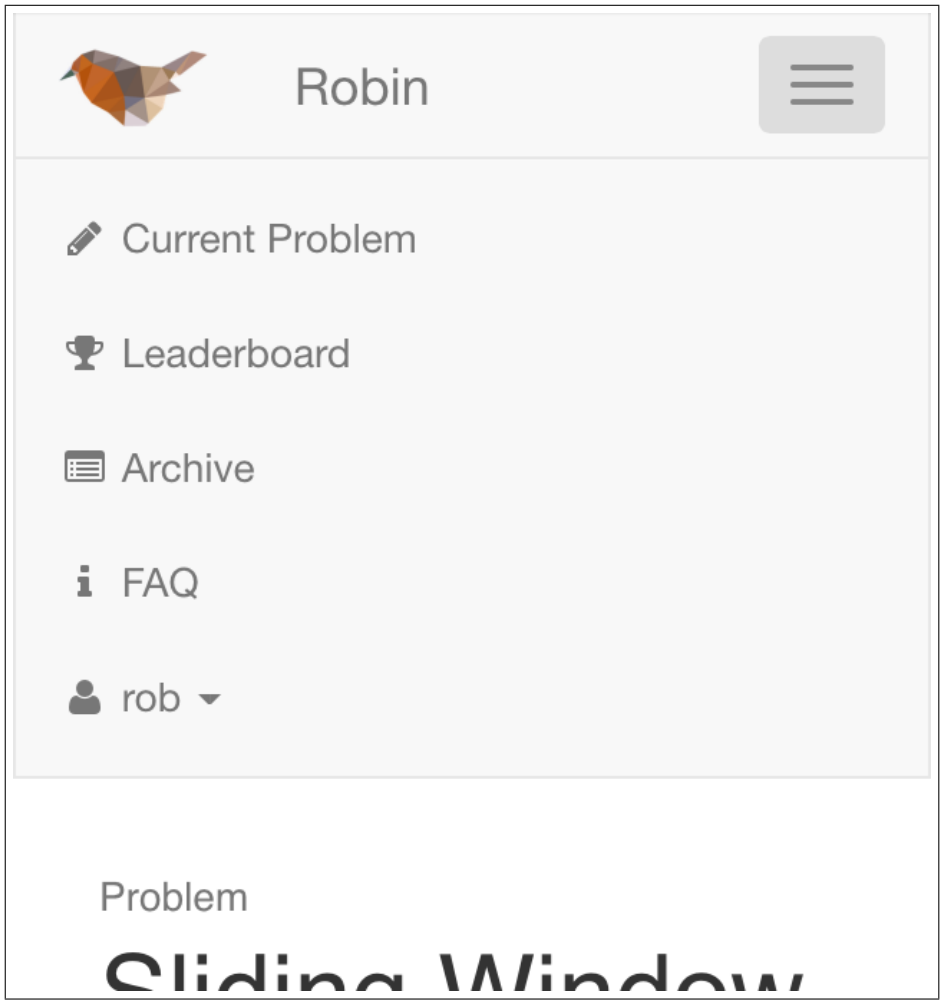

Figure B.2: Menu

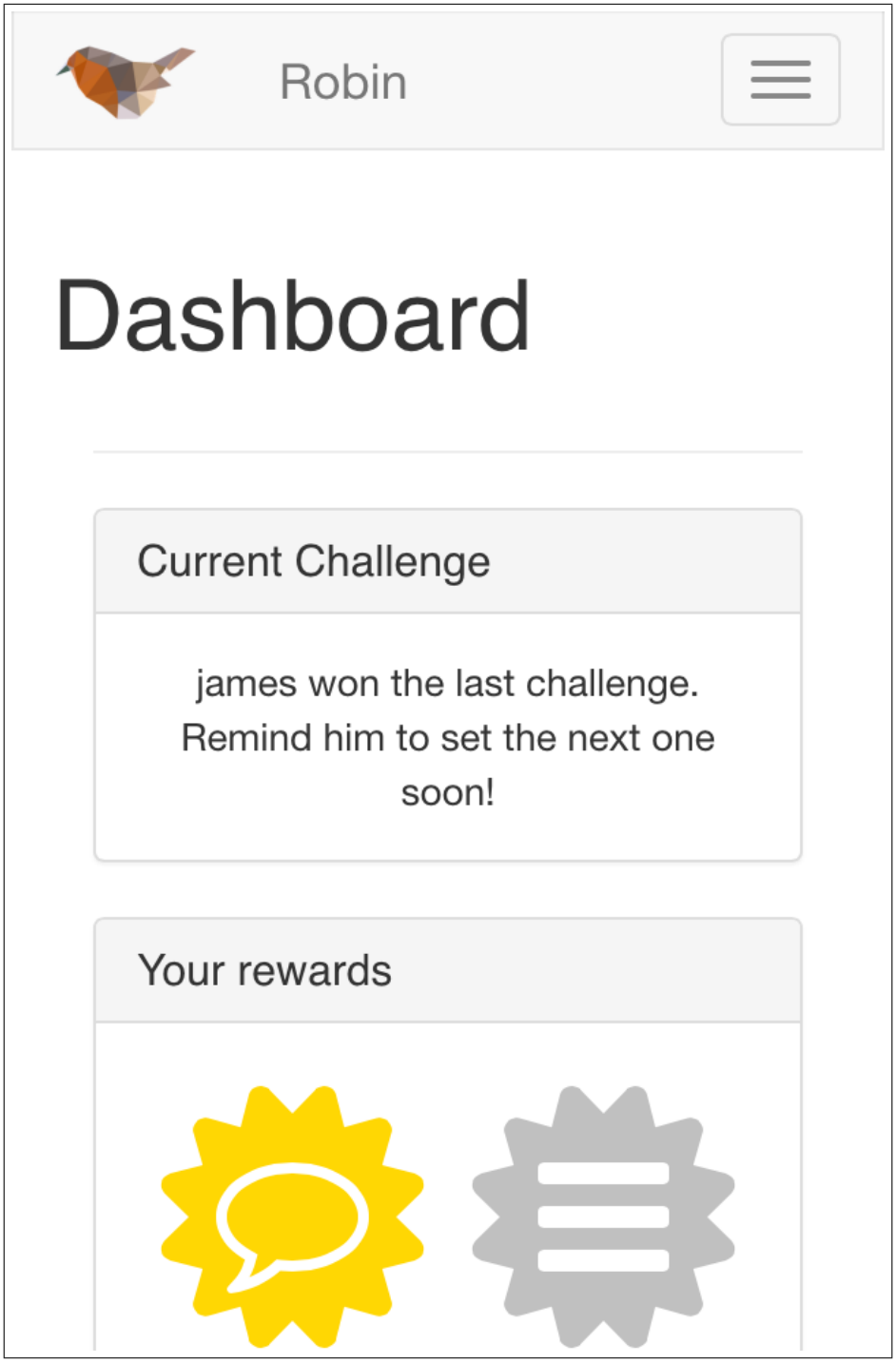

Figure B.3: Dashboard

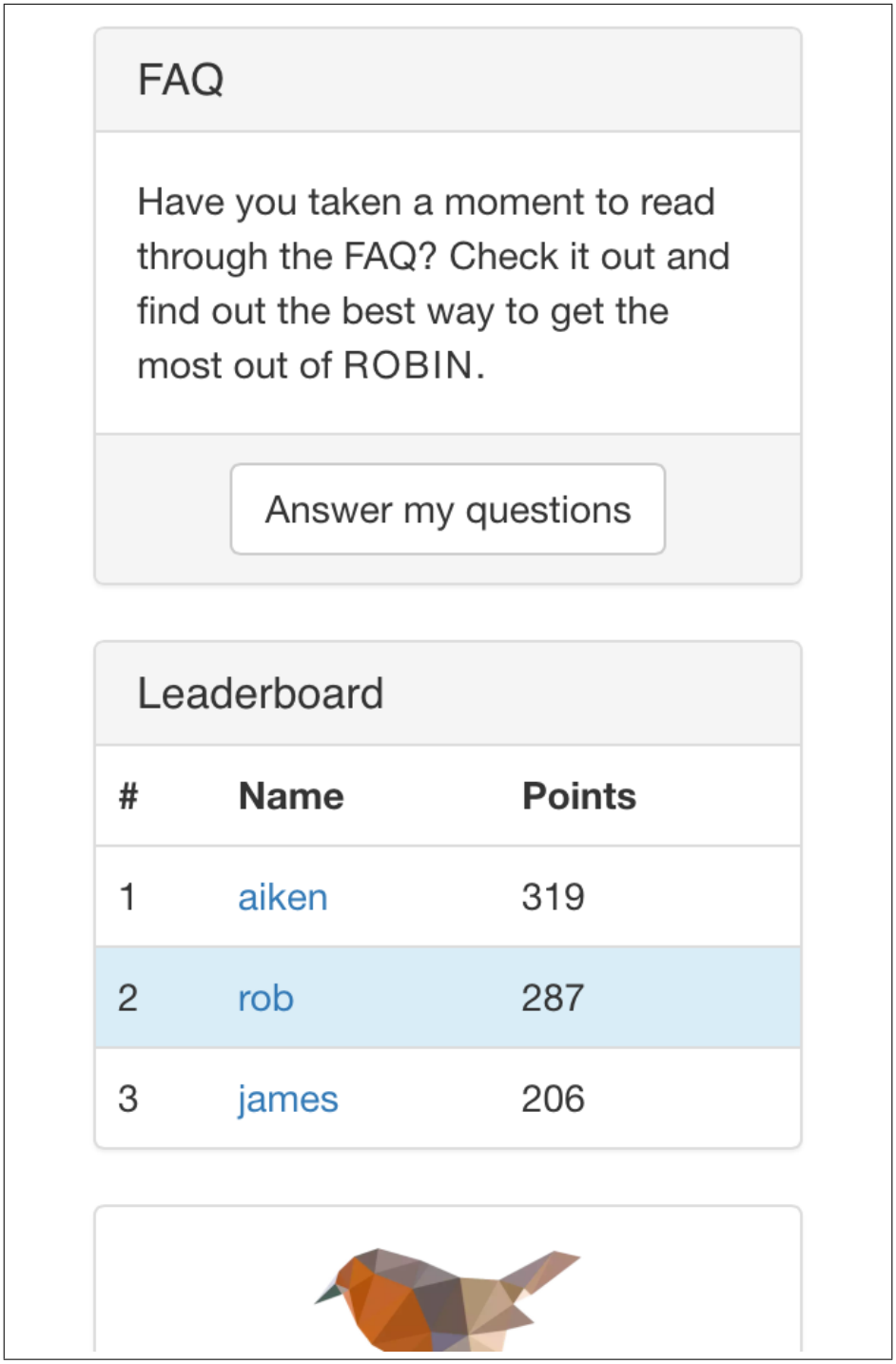

Figure B.4: Dashboard with leaderboard

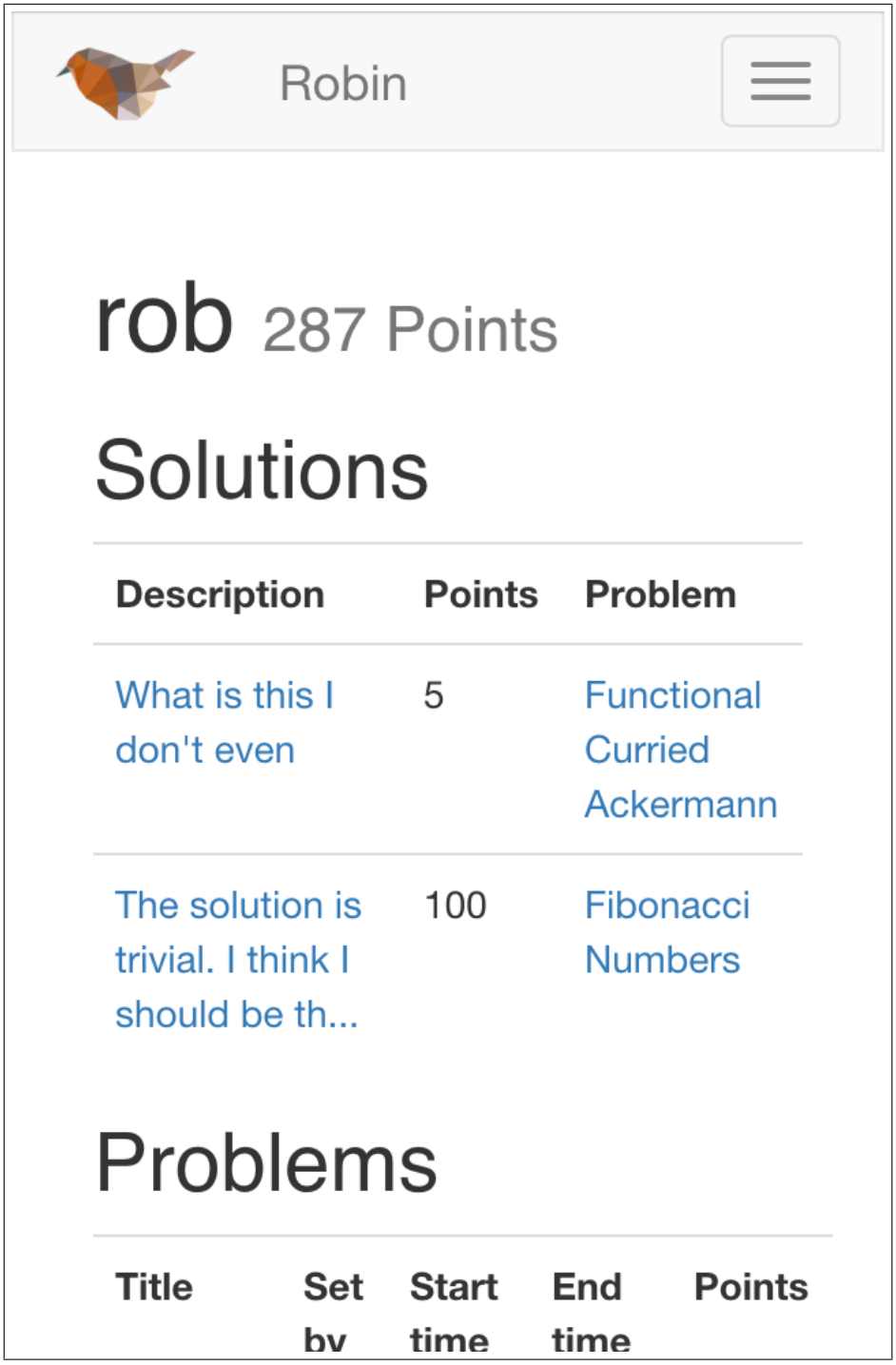

Figure B.5: User page