

Group Project Report: Robin

Robert Bastian
Paavan Buddhdev
Aiken Cairncross
James Dai
Alistair Gavin

Supervisor: Ivo Sluganovic
Corporate Sponsor: *Palantir*

University of Oxford

May 2015

Contents

1 Specification and planning	2
1.1 The brief	2
1.2 Our initial ideas	3
1.3 Our chosen idea (and the reasons why we chose it)	3
1.4 How the game goes	4
1.5 Collaboration methods and group roles	5
1.6 Design decisions: back-end	6
1.7 Design decisions: front-end	6
1.8 Use cases	6
2 System integration and report	9
2.1 Design/layout mockups	9
2.2 Site structure and state diagram	10
2.3 Tests	12
2.4 Extensions	12
A Desktop screenshots	13
B Mobile screenshots	19

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 How the game goes

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 challenge 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. Leader-board updated with points of each participant
7. Challenge archived

1.5 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.

1.6 Design decisions: back-end

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, paired with 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.

1.7 Design decisions: front-end

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 Bootstrap (<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.

1.8 Use cases

The main use cases are in Fig. 1.1-4

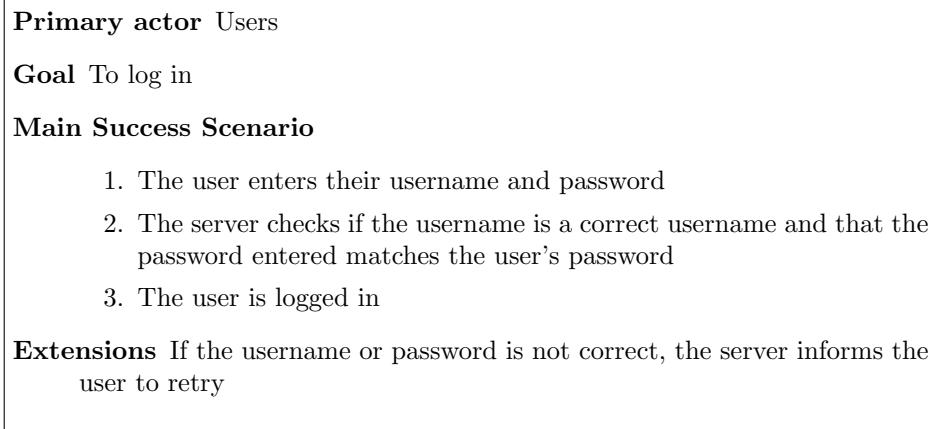


Figure 1.1: Logging in

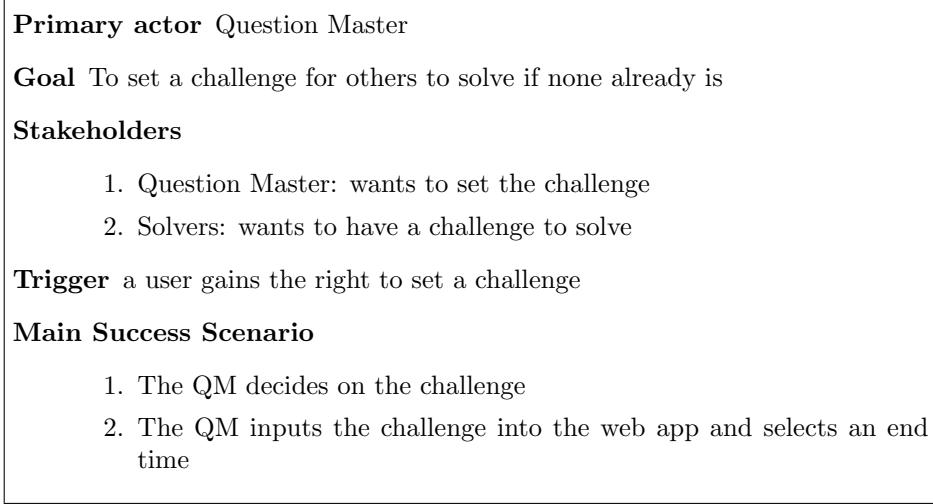


Figure 1.2: Setting a challenge

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

Chapter 2

System integration and report

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

The following are some of the initial drawings from one of our first sessions for what the site structure will look like.

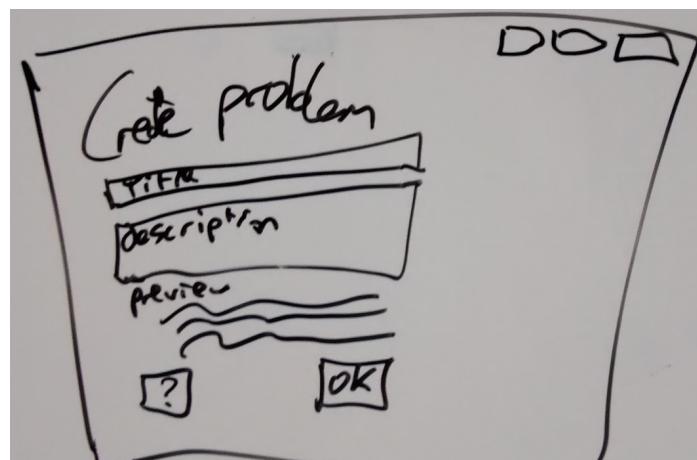


Figure 2.1: Challenge submission

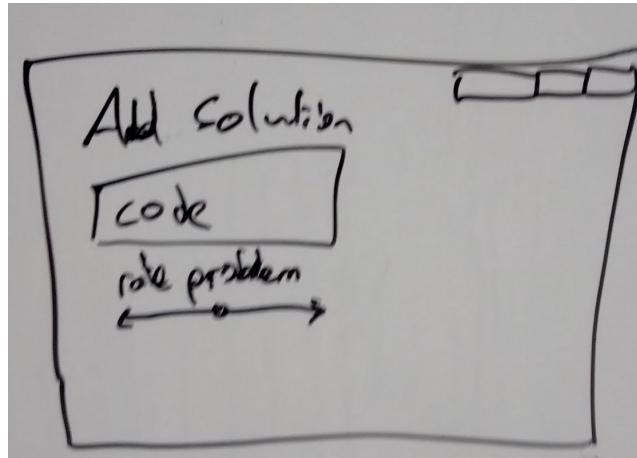


Figure 2.2: Solution submission

2.2 Site structure and state diagram

The following diagrams are the initial outlines for the app's database (model), its website structure (views), and a state diagram.

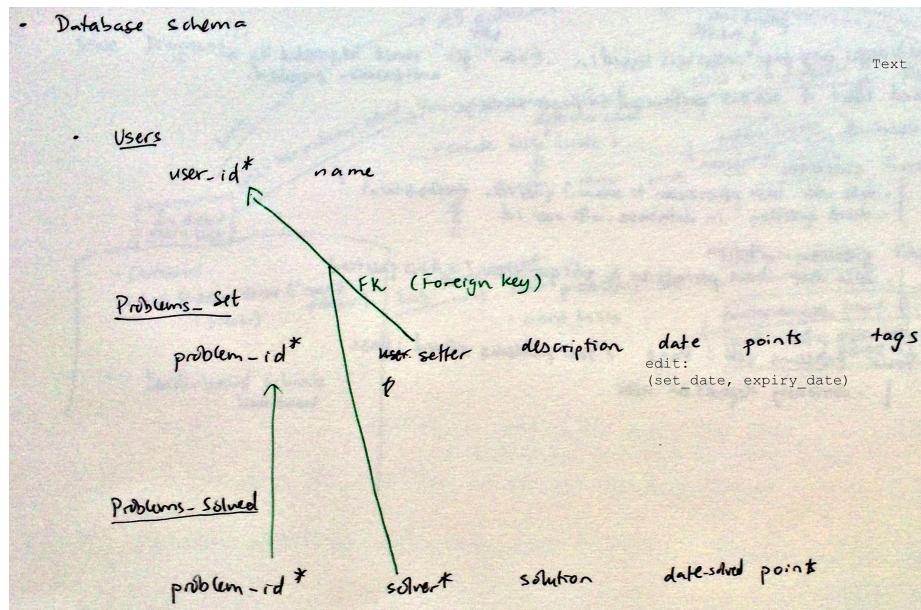


Figure 2.3: Database schema

- Leaderboard
 - ^{user} should be able to have some control of what he sees on user's leaderboard e.g.

Time	Sort By	Question Tag
All time	Setter Rank } based on average or Solver Rank } total points?	Scala ✓
Last year ✓		C ✓
Last 8 months		Bash ✓
Last month		
Last week		
- These options parameterize our database query e.g.


```
SELECT uid, name, solve-points
FROM Users IX Problems_Solved
WHERE date within range (15 Apr 2014, 15 Apr 2015)
and (tags has "Scala" or tags has "Bash")
↳ actually not sure. Can a database cell contain a list datatype?
```
- We then run a ranking algorithm e.g. solver-rank on the list of tuples returned and display the results.

Figure 2.4: Leaderboard functionality

- User authorisation
 - Nobody should be able to access any page unless he is logged in. Each "get" should be preceded by an ^{HTTP} authorisation procedure.
 - Content of pages and effect of actions performed in pages should be parameterised by user id
 - eg submit question : Action
 - | - does user have permission to submit question (is it his turn) ?
 - | - record question in database with user id
 - edit question : Action
 - | - does user have permission to edit question (is it his question)
 - Your problems set : Page / Your problems solved : Page
 - | - obviously depends on user

Figure 2.5: User authorization

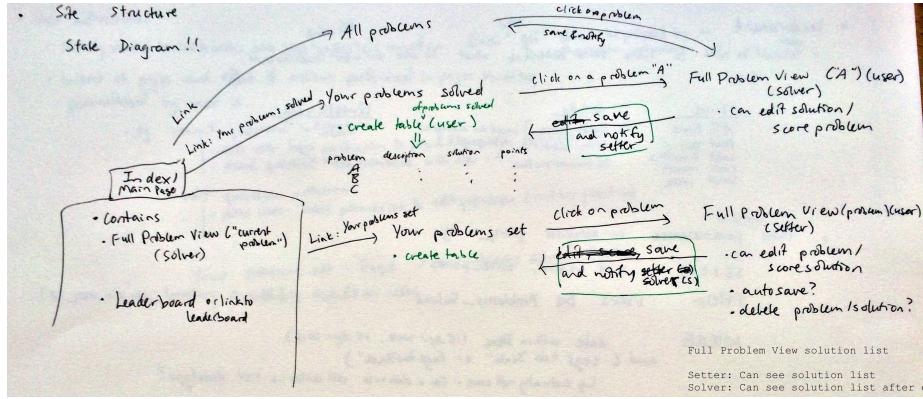


Figure 2.6: Site structure

2.3 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.

2.4 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>

The screenshot shows the ROBIN application's archive page. At the top, there is a navigation bar with links for 'Current Problem', 'Leaderboard', 'Archive', 'FAQ', and a user profile for 'alistair'. Below the navigation bar, the title 'All problems' is displayed. A table lists three problems:

Title	Set by	Start time	End time	Points
Sliding Window Protocol	rob	04 May 2015, 23:20	07 May 2015, 23:20	0
Fibonacci Numbers	aiken	03 May 2015, 23:09	04 May 2015, 23:09	178
Functional Curried Ackermann	james	02 May 2015, 23:09	03 May 2015, 23:09	37

At the bottom of the page, a footer notes: 'The ROBIN application was developed by Robert Bastian · Paavan Buddhdev · Aiken Cairncross · James Dai · Alistair Gavin'.

Figure A.1: Archive

The screenshot shows a solution creation page for a problem titled 'Write CSO functions'. The problem statement asks to implement two functions: `[T]Send(n: Int, out: !(T), get_ack: ?[Int]): Proc` and `[T]Recv(in: Int, in: ?[T], ack: !(Int)): Proc`, which correctly send frames `T` over a lossy network modelled by `lossy`, using the sliding window protocol with windows of size `n`. Note if `n = 1` this is the alternating bit protocol.

A code editor displays the following Scala code:

```

def lossy(in: OneOne[T], out: OneOne[T], get_ack: OneOne[Int], ack: OneOne[Int]): Proc = {
    val r = new scala.util.Random
    serve(
        out :=> (x: T) => if (r.nextInt(2) == 0) in(x)
        ack :=> (x: Int) => if (r.nextInt(2) == 0) get_ack(x)
    )
}
val in, out = OneOne[T]
val get_ack, ack = OneOne[Int]

```

Below the code editor is a text input field labeled 'Enter your solution'.

At the bottom, there is a rating scale from 'Not so much' to 'A lot' for the question 'How did you like this problem?' followed by a 'Submit solution' button.

Figure A.2: Solution creation page



Robin

Problem

Sliding Window Protocol

Ended on 07 May 2015, 23:20

By: rob

Write CSO functions

- `[T]Send(n: Int, out: [T], get_ack: ?[Int]): Proc` and
- `[T]Recv(n: Int, in: [T], ack: ![[Int]): Proc`

that correctly send frames `T` over a lossy network modelled by `lossy`, using the sliding window protocol with windows of size `n`. Note if `n = 1` this is the alternating bit protocol.

```

def lossy(in: OneOne[T], out: OneOne[T], get_ack: OneOne[Int], ack: OneOne[Int]): Proc = {
    val r = new scala.util.Random
    serve {
        out >= (x:T => if (r.nextInt(2) == 0) in(x)
                 ack >= (x:Int => if (r.nextInt(2) == 0) get_ack(x)
                )
            )
    }
    val in, out = OneOne[T]
    val get_ack, ack = OneOne[Int]
}

```

Figure A.3: Current problem



Robin

Current Problem Leaderboard Archive FAQ alastair ▾

Frequently asked questions

This section contains a wealth of information about Robin and how to use it. If you cannot find an answer to your question here, please [contact us](#).

What is Robin?
Robin is a way of challenging colleagues or friends to solve interesting challenges which each of the people in the group have come up with.
How does Robin work?
Who is the Question Master??
What do these points mean?
How do I join?
Why is it called Robin?

Figure A.4: Frequently asked questions



Robin

[Current Problem](#) [Leaderboard](#) [Archive](#) [FAQ](#) [aiklein](#) ▾

Leaderboard

User	Challenges	Solutions	Wins	Points
aiklein	1	1	1	310
james	1	1	0	106
rob	1	2	1	105
paavan	0	2	0	7
alistair	0	0	0	0
thomaslove	0	0	0	0
laure	0	0	0	0

The ROBIN application was developed by
Rohert Bastian · Paavan Burdhaev · Aileen Calmres · James Dai · Alistair Gavin

Figure A.5: Leaderboard



Robin

[Current Problem](#) [Leaderboard](#) [Archive](#) [FAQ](#) [aiklein](#) ▾

Problem

Fibonacci Numbers

Ended on 04 May 2015, 23:09

By: aiklein

Winner: rob

Write a definition for an infinite list of the fibonacci numbers in Haskell

Solutions

Description	Points	User
...Why functional programming? --It's clean and...	69	james
These questions aren't hard at all.	0	paavan
The solution is trivial. I think I should be th...	100	rob

Figure A.6: Finished challenge



Figure A.7: Challenge marking

The screenshot shows the Question Master's view of a challenge. At the top, there is some initial text and a code snippet:

```

• [ij]parametrize: arr, out: Int, get_ack: Boolean; final; private arr
• [ij]Recv(n: Int, in: [T], ack: ![Int]): Proc

```

that correctly send frames `T` over a lossy network modelled by `lossy`, using the sliding window protocol with windows of size `n`. Note if `n = 1` this is the alternating bit protocol.

```

def lossy(in: OneOne[T], out: OneOne[T], get_ack: OneOne[Int], ack: OneOne[Int]): Proc = {
    val r = new scala.util.Random
    serve(
        out +=> (x:T => if (r.nextInt(2) == 0) in\|x)
        ack +=> (x:Int => if (r.nextInt(2) == 0) get_ack\|x)
    )
}
val in, out = OneOne[T]
val get_ack, ack = OneOne[Int]

```

Below this is a table titled 'Solutions' with columns 'Description', 'Points', and 'User'.

Description	Points	User
Bob	0	alastair
This is a solution	0	aiken
This is another test solution	0	paavan
I'm really bored now	0	james
hey	0	laure

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

The screenshot shows the ROBIN application interface. At the top, there's a navigation bar with links for 'Current Problem', 'Leaderboard', 'Archive', 'FAQ', and a dropdown for 'aiken'. Below the navigation is a user profile section for 'aiken' (310 Points). It includes a 'Solutions' section with a table showing a single entry: `f g = foldl (g 1, g)` (points 132) for the problem 'Functional Curried Ackermann'. There's also a 'Problems' section with a table showing a single entry: 'Fibonacci Numbers' (set by 'aiken' on 03 May 2015 at 23:09, end time 04 May 2015 at 23:09, points 178). A note at the bottom states: 'The ROBIN application was developed by Robert Bastian · Paavan Buddhe · Aiken Cairncross · James Dal · Alistair Gavin'.

Figure A.9: Profile

The screenshot shows the ROBIN application interface. At the top, there's a navigation bar with links for 'Current Problem', 'Leaderboard', 'Archive', 'FAQ', and a dropdown for 'aiken'. Below the navigation is a 'Solution' section for the problem 'Functional Curried Ackermann' (submitted on 03 May 2015 at 03:09 by 'aiken' in Haskell). The code is displayed in a monospaced box:

```
# g = foldl (g 1, g)
-- ack_c x gives a function from Nat -> Nat.
-- ack_c 0 gives succ,
-- ack_c 1 gives f succ,
-- and so on.
-- This can be explained by noticing that
-- the partially applied function
-- ack(x,_) is a function defined in terms of ack(x, _).
-- if we expand the definitions we see that ack(x+1, y) =
-- ack(x, ack(x+1, y-1)) = ack(x, ack(x, ack(x+1, y-2))) + ... +
-- ack(x, ...y)(ack(x+1, y-y)) = ack(x, ...y)(ack(x, 0))
-- this is precisely foldl(ack(x, 0), ack(x, _)) y.
```

A note at the bottom states: 'The ROBIN application was developed by Robert Bastian · Paavan Buddhe · Aiken Cairncross · James Dal · Alistair Gavin'.

Figure A.10: Solution

Appendix B

Mobile screenshots

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



Robin

Problem

Sliding Window Protocol

Ended on 07 May 2015, 23:51

By: rob

Winner: **james**

Write CSO functions

- `[T]Send(n: Int, out: ! [T], get_ack: ?[Int]): Proc` and
- `[T]Recv(n: Int, in: ? [T], ack: ![Int]): Proc`

Figure B.1: Problem page

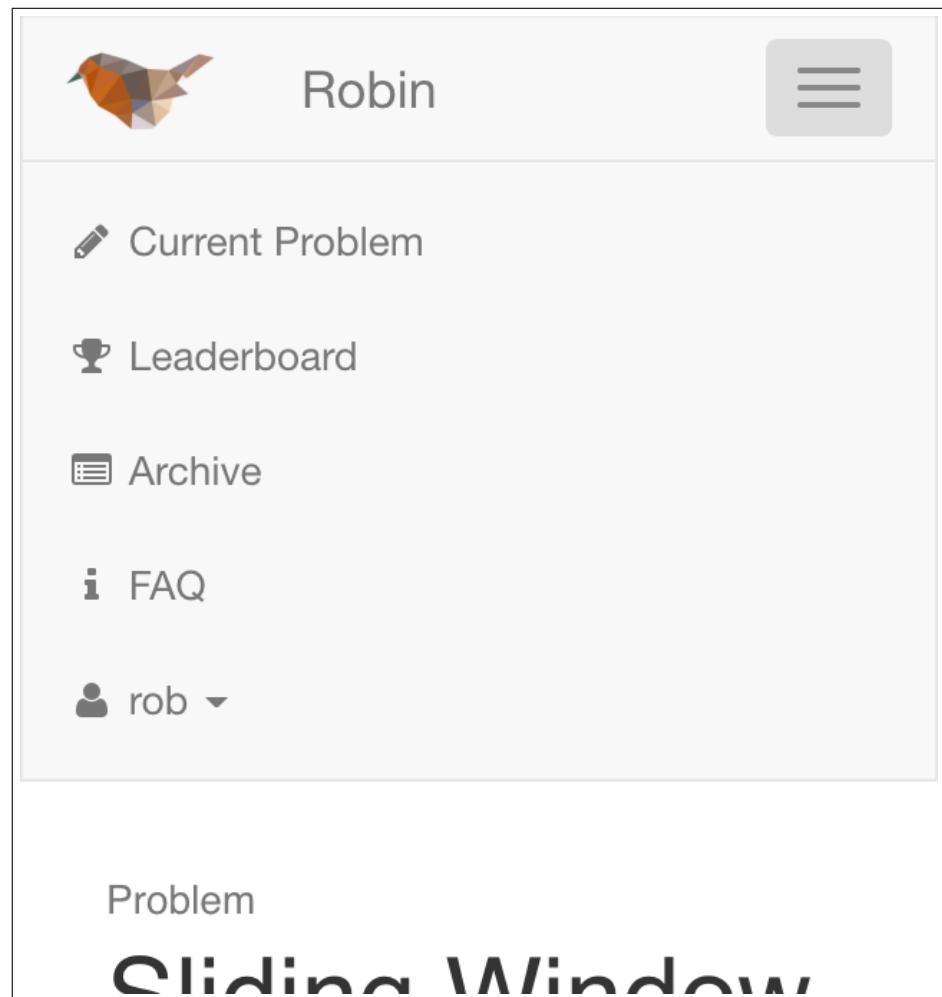


Figure B.2: Menu

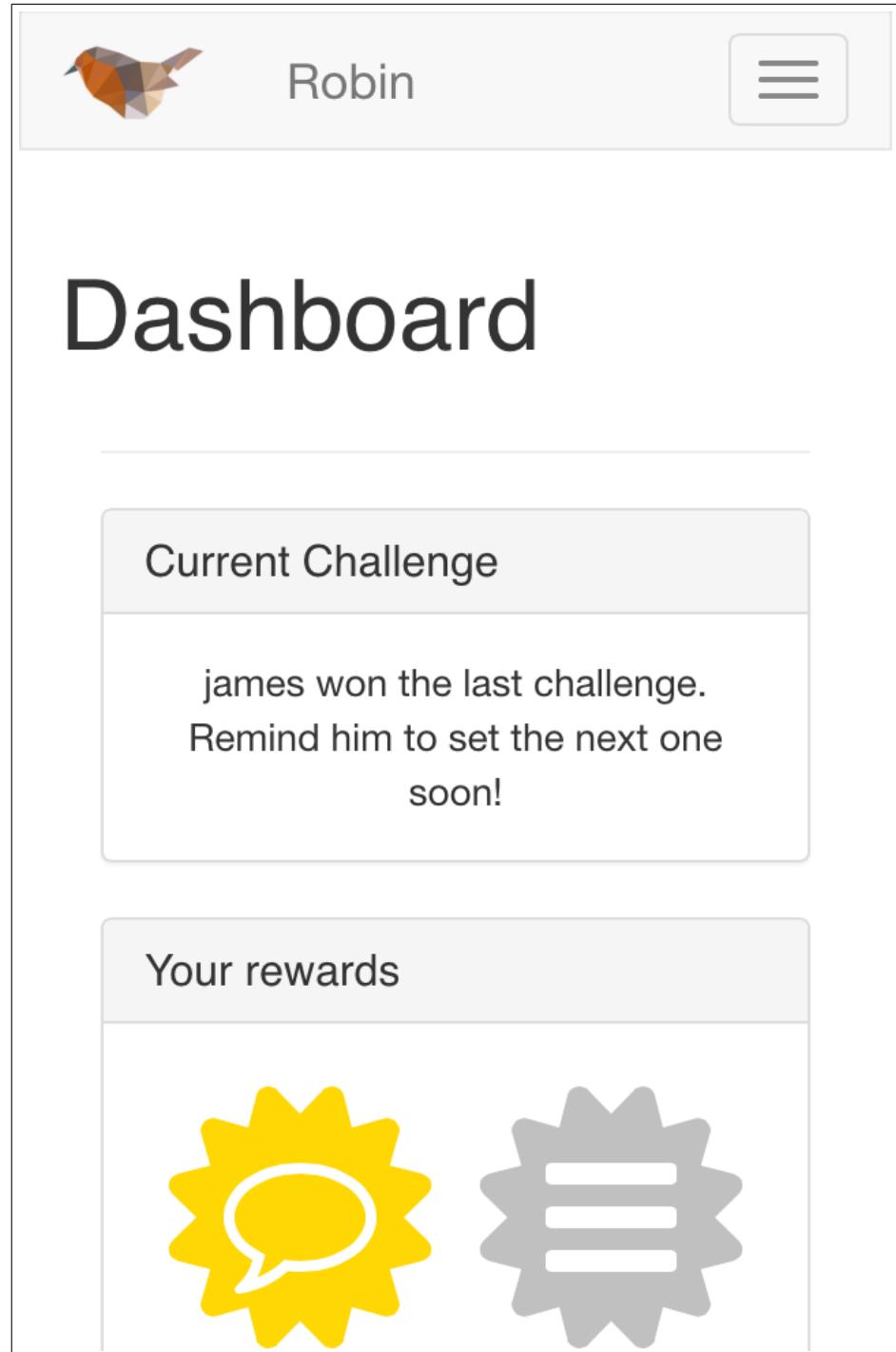


Figure B.3: Dashboard

FAQ

Have you taken a moment to read through the FAQ? Check it out and find out the best way to get the most out of ROBIN.

[Answer my questions](#)

Leaderboard

#	Name	Points
1	aiken	319
2	rob	287
3	james	206



Figure B.4: Dashboard with leaderboard

The image shows a user interface for a mobile application. At the top, there is a header bar with a small orange bird icon on the left, the name "Robin" in the center, and a menu icon (three horizontal lines) on the right. Below the header, the user's name "rob" and "287 Points" are displayed. A large section titled "Solutions" follows, containing two rows of data. Each row has three columns: "Description", "Points", and "Problem". The first solution is titled "What is this I don't even", worth 5 points, and is associated with the problems "Functional", "Curried", and "Ackermann". The second solution is titled "The solution is trivial. I think I should be th...", worth 100 points, and is associated with the problem "Fibonacci Numbers". Below this section, another large title "Problems" is present, followed by a table with five columns: "Title", "Set", "Start time", "End time", and "Points". The "Title" column contains the text "bv".

Title	Set	Start time	End time	Points
bv				

Figure B.5: User page