UNIVERSITY OF CAPE TOWN
IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

Department of Computer Science

Computer Science Honours Final Paper 2016

Title:	RASPIED Web App: Managing Remote Shared Access to an Educational Robot
Author:	Muhummad Patel
Project abbreviation:	RASPIED
Supervisor:	Gary Stewart

Category	**Min**	**Max**	**Chosen**
Requirement Analysis and Design	0	20	10
Theoretical Analysis	0	25	-
Experiment Design and Execution	0	20	15
System Development and Implementation	0	15	15
Results, Findings and Conclusion	10	20	10
Aim Formulation and Background Work	10	15	10
Quality of Paper Writing and Presentation	10		10
Quality of Deliverables	10		10
Overall General Project Evaluation	0	10	
Total marks			80

RASPIED Web App: Managing Remote Shared Access to an Educational Robot

Muhummad Patel
ptlmuh006
University of Cape Town
muhummad.patel@gmail.com

ABSTRACT

This paper presents the development of the web app component of an educational robot platform. The web app was required to manage remote shared access to a single robot and also allow students to program the robot and view a live video stream of the robot. The web app was evaluated based on its perceived usability and whether or not the initial requirements were met. Two rounds of usability testing were conducted with seven participants in the first round and six participants in the second round. Participants chosen for the usability tests were expert users and were all Computer Science Masters and Honours students who had served as tutors or teaching assistants for introductory Computer Science courses. The final version of the web app was found to have an 'excellent' usability rating and participant feedback indicated that it would be a useful tool for teaching introductory Computer Science courses. The final web app has met all the initial requirements and is fully functional and ready for deployment.

CCS Concepts

•**Social and professional topics** → **Computer science education;** *CS1;* •**Applied computing** → *Interactive learning environments;* •**Information systems** → Web applications;

Keywords

Computer Science; Education; CS1; Web App

1. INTRODUCTION

The potential use of robots as pedagogical tools has long been recognised and much work has been done in this field. This project forms part of the larger Raspberry Pi Educational Robot (RASPIED) project. In particular, this paper outlines the design, implementation, and evaluation of the supporting web app component of the RASPIED platform. This section introduces the RASPIED project and situates the contents of this paper in the context of the larger RASPIED project.

1.1 Project Description

The RASPIED project had the aim of designing and implementing a complete educational robot platform built using easily accessible and commercially available hardware and software. This platform was developed with the intention of being used in the CSC1010H Python-based introductory Computer Science course at UCT. The robot platform consists of a Raspberry Pi based robot, and a controlling web app. The distinguishing feature of the RASPIED project is it's use of a single robot to be shared among the entire class.

The robot is able to move and supports various 'extra' features (viz. a camera with colour detection capabilities, an ultrasonic distance sensor, infra-red obstacle detection, and three coloured LEDs). The robot is programmable in Python and exposes a simple API for use by students.

The subject of this paper is the design, implementation, and evaluation of the controlling web app. The web app facilitates student interactions with the robot by providing a live video stream of the robot and allowing students to write and upload code to the robot. Access to the robot is managed by requiring students to book time slots during which they are allowed to upload and run code on the robot. This booking functionality facilitates the shared access to the robot and ensures that only a single student has control over the robot at any given time.

1.2 Work Allocation

In order to develop the larger RASPIED project, the system was split into three parts which were then divided among three Computer Science Honours students. The core robot along with the basic movement functionality and associated API was developed by Josh Di Bona. The 'extra' features as well as the API for these features were developed by Jeremy Coupland. Both of these related papers can be found on the project website as indicated in Appendix A. Lastly, the controlling web app (the subject of this paper) was developed by the author.

2. BACKGROUND

This section examines the hardware and software of the various approaches to educational robot taken by previous studies in this field, as well as their implications on this project.

2.1 Hardware

Many of the surveyed studies used robots that were custom built to suit the requirements of that specific application while others used pre-built robots or modified pre-built robots. Key hardware considerations are the cost of the robot and the features supported. In this section, we focus on custom built robots with particular attention given to robots built using Lego Mindstorms Kits and robots built using custom hardware.

2.1.1 *Lego Mindstorms*

The Lego Mindstorms robotics kit consists of a selection of standard Lego components, along with a programmable controller brick and a number of input sensors and physical actuators (e.g. motors, lights, etc.). The user is required to build the robot and connect the sensor/output components to the controller brick which can then be programmed to achieve the desired behaviour.

The Lego Mindstorms sensors and motors come packaged in convenient, plug-and-play type modules that have been made specifically to work with the Mindstorms platform; however, these official modules are far more expensive that the raw electronic components. The feature set supported by these official components does not appear to be a limiting factor and it has been successfully used to teach such advanced concepts as Artificial Intelligence at a tertiary level[9].

The major limitations of the platform include the limited on-board memory and the lack of support for useful wireless communication protocols[9]. The Lego Mindstorms platform only supports three input ports but this limitation can be circumvented by 'stacking' sensors to the same port and interpreting the raw input in the software[9].

Another common criticism levelled at the Lego Mindstorms platform - particularly in the context of tertiary-level courses - is that the official graphical (drag-and-drop) programming language is too simple[9]. There are, however, third-party tools allowing the controller to be programmed using many popular languages including among others, Java, Ada, and C++. One of the studies surveyed overcomes this problem by using the Not Quite C (NQC)[2] programming environment[9], while another used the Ada/Mindstorms2.0[6] programming environment[8].

The surveyed studies that used Lego Mindstorms based robots - such as Klassner, F. (2002)[9] - generally found it to be a cost effective solution; however, we argue that the cost of the platform is still prohibitively high. Furthermore, an extensive year long study to measure the effectiveness of robots in teaching Computer Science found that the Lego robots were ineffective and that students using the robots performed worse than the control group[8]. This negative result was attributed mainly to a lack of access to the robots and this hypothesis is supported by the fact that other studies using cheaper robots provided to every student reported more positive results[14].

2.1.2 *Custom Hardware*

A number of the surveyed robot-aided curricula made use of fully custom hardware or modified pre-built robots. Motivations behind these approaches include cost reduction, the inclusion of novel features, and support for particular applications/use-cases.

A major benefit of using custom hardware is the ability to control the cost of the robot and tailor its functionality. For example, one of the papers surveyed presents a custom-built, low-cost platform tailored for use with 3$^{\text{rd}}$ and 4$^{\text{th}}$ grade students[16]. The platform allowed for the entire system to be set up in five minutes and consisted of five simple robots that were entirely custom built for €15 each. These five robots were then connected (via bluetooth) to a central Raspberry Pi server that allowed for the robots to be programmed wirelessly through a web-based, drag-and-drop programming interface. Another study conducted at Bryn Mawr College used the pre-built Scribbler robot from Parallax Inc. with a custom add-on for further functionality such as a camera and wireless bluetooth connectivity[3]. The customised robot costs $110 and is now available in kit form from the Institute for Personal Robotics in Education (IPRE)[10].

Other customised hardware solutions have been implemented in a similar manner to those described above and fall into three main categories, viz. fully custom built robots[16], modified pre-built robots[10], and standard pre-built robots[11].

As previously stated, custom built robots can be tailored to suit specific curriculum and budget requirements but, crucially, may require more effort to build, set up, and maintain. Modified pre-built robots reduce the time investment required to build, set up, and maintain the robots, whilst still allowing some control over the functionality of the robot. The modified off-the-shelf robots, however, are generally more expensive than fully custom built robots. Standard pre-built robots are generally the most expensive option but require very little to no build time.

2.2 Software

When using robots as educational tools, particularly in the context of Computer Science education, the method and language used to program the robot should be carefully considered. The languages currently being used to program educational robots include (in descending order of popularity) Java, C++, Not Quite C (NQC), C, Python, and Scheme[12].

2.2.1 *Graphical Programming Languages*

Graphical programming languages like Scratch[13] have been widely used to introduce fundamental programming concepts. Robots built using the Lego Mindstorms platform are, by default, programmed using a graphical, drag-and-drop interface provided by Lego. There are many popular third-party tools allowing the Lego Mindstorms robots to be programmed using more conventional languages (or subsets thereof). These third-party tools allow for increasingly complex programming of the Mindstorms platform and have been used with some success in an educational context[9].

Graphical programming languages are well suited to younger audiences but have not been widely used to teach Computer Science at a tertiary level as they are too simplistic. Furthermore, introducing students to Computer Science using a graphical programming language designed around maximising fun may leave students with a feeling of 'bait-and-switch'[3] if later languages are too different.

2.2.2 *Traditional Programming Methods*

At a tertiary level, it would be more appropriate to use actual programming languages, or subsets thereof for all teaching activities. A common design pattern in this space is to design an API in a common language that allows users to program the robots easily whilst still allowing sufficient complexity. One of the better known examples of this design pattern is the Arduino programming language which is based on C/C++ and is used to program Arduino micro-controller prototyping boards. It allows novice users to write trivial programs whilst still allowing more experienced users to exploit most of the features of C++. The Arduino IDE also hides most of the complexity involved in compiling the code and uploading it to the Arduino device which greatly simplifies the process for new users; however, more experienced users are free to use their preferred toolsets (e.g. Makefiles). The Arduino platform is a good example of an easy-to-use tool that scales well with user experience, which has been identified a a key factor in engaging students[10]. The Raspberry Pi implements a similar idea by allowing novice users to control its GPIO pins using a Python library.

2.2.3 Robot Simulations

The use of robot simulations as an educational tool has also been explored. Using a software simulation of a robot greatly reduces the cost and possible hardware issues involved. Two examples of such simulation packages include Karel[15], and Alice[5]. A four semester study using Karel-like simulation software for the duration of an introductory Computer Science course was able to reduce the percentage of failing students by 77.4% (from 43.1% to 9.8%)[18]. Similarly positive results for simulation based approaches have also been reported in a previous review of the field[12]. Whilst a simulation based approach reduces the cost and hardware issues, it also removes the tangible aspect of using a robot as a teaching aid. A possible solution to this is a hybrid approach that uses both a physical robot, and a software simulation (e.g. Using Lego Mindstorms robots and a compatible simulator for students to test their programs[7]). This allows students to work with the robot without having to worry about hardware problems whilst still retaining the tangible aspect of using an actual robot.

2.3 Implications for This Project

Studies relating to the use of robots as an educational aid with particular focus on applications in introductory Computer Science courses at a tertiary level have been reviewed. The hardware and software used to build and program the robots in the various studies considered has been discussed along with the challenges faced when using robots as teaching aids.

Much of the early work using robots in education made use of the Lego Mindstorms platform. More recent work has shifted to focus more on cheaper, custom built robot hardware and modified pre-built robots. This shift could possibly be attributed to the introduction of low-cost, hackable, computing devices like the Arduino, and Raspberry Pi. Studies using the cheaper custom built solutions have shown similar success rates compared to studies using the more expensive Lego Mindstorms platform.

While some work has been done to investigate low-cost robots in primary education[16], there does not appear to be much work done on low-cost robots at a tertiary education level. One reason for this may be that the increased feature set required to support the Computer Science curriculum inevitably pushes the cost up. The Institute for Personal Robotics in Education (IPRE) has produced a robot and curriculum designed for a Computer Science course where every student is provided with a robot but this may still prove too costly for departments with a low-budget (e.g. in a South African context). There is still space for work to be done to reduce the cost of educational robots while still providing a feature set that supports the Computer Science curriculum.

It has been shown that insufficient access to the robot is a major frustration for students and may result in worse performance than traditional teaching methods[8]. One solution to this is to provide every student with their own personal robot. A more cost effective solution is to, rather than purchase more robots, give students access to a simulation of the robot[7]. Simulation based approaches have been shown to be at least as successful as approaches using physical robots[12].

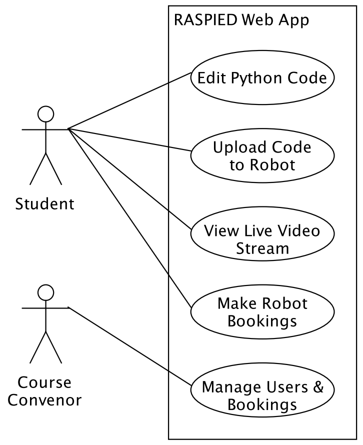


Figure 1: Use-case diagram showing the core functional requirements of the web app.

The use of robots in Computer Science education has been shown to be effective. One of the key determinants of success is sufficient access to the robot for students[8]. This project aims to improve access to the robot whilst still keeping costs low by allowing shared access to a single robot through a controlling web app. Students are then able to program and debug their solutions off-campus without needing to buy their own robot.

3. DESIGN

3.1 System Requirements

This project entailed the design and implementation of a web app to serve as the student-facing interface with the RASPIED educational robot. The specific functional requirements were generated in consultation with the course convenor of CSC1010H course (the target course). These requirements are illustrated in Figure 1 and further detailed in the following sections.

3.1.1 Edit Python code

Students need to be provided with a space to edit their Python programs for the robot. Python was chosen because the target course is Python-based. The editor needs to be familiar to students (similar to their current editor) and should not be confusing/cumbersome to use.

3.1.2 Remote Access to the Robot

In the survey of related works, adequate access to the robot (particularly outside the classroom) was identified as a key determinant of the success of robot aided curricula. The web app should therefore facilitate increased access to the robot by allowing students to work with the robot remotely. This includes being able to program the robot remotely and see what the robot is doing.

3.1.3 Upload Code to the Robot

The web app needs to provide students with an easy mechanism to upload their code to the robot and run it. Ideally, this should be done in much the same way as their current IDE runs code.

3.1.4 Live Video Stream

As students are expected to use the robot remotely, the web app needs to provide a live video stream of the robot. This will allow students to verify that the robot behaves as expected when they run their code.

3.1.5 Shared Robot Booking System

The RASPIED project only makes use of a single robot. The web app need to manage shared access to the robot so that only a single student can upload code to the robot at any given time. To this end, students should be required to book out time slots before being allowed to upload/run code on the robot. The web app should put in place measures to prevent hogging of the shared robot and ensure that no clashes occur. Furthermore, only authorised students (students who are registered for the target course) should be allowed to book time slots and use the robot.

3.2 Technology Selection and Rationale

A two week investigation into web frameworks was conducted in order to determine which would be best suited for the web app. Python Django, Ruby on Rails, and Flask, among others, were tested. Most frameworks seemed adequately qualified for this use case and reasonably simple to use.

Ultimately, the Python Django framework was chosen due partly to the authors prior experience with it. Django also provides a number of useful features out-of-the-box including simple generation of administration interfaces and an easily extensible authentication and authorisation infrastructure. Furthermore, the Django Channels package added the possibility of real-time full-duplex communications between the client and server over WebSockets. This real-time communication between the client and server greatly simplifies the process of remotely interacting with the robot (see Section 4.1.3).

The default database used with the Django framework is SQLite, which is a lightweight, self-contained, server-less database. For our use case, SQLite is perfectly adequate[17] as we are not expecting a heavy load. The database is mainly needed to manage users, user sessions, bookings, and robots. Using a lightweight, server-less database like SQLite also means that we keep to a minimum the set-up and configuration needed to get the web app up and running.

MaterializeCSS was chosen as the front-end framework used to develop/style the user interface(UI). MaterializeCSS is a CSS framework similar to Bootstrap that provides a number of UI components, a useful grid system to aid in page layout, and pre-defined style classes to ensure a consistent look-and-feel. Unlike Bootstrap, the MaterializeCSS framework is based on Google's Material Design guidelines. The Material Design guidelines have been widely adopted and should be familiar to users of Android-based smart-phones or any Google services such as Google Inbox, YouTube, etc. Using MaterializeCSS allowed us to produce a well-polished UI that made use of design metaphors and UI components that students were already familiar with.

JavaScript with jQuery was used as the client-side scripting language to handle user interaction. Additionally, AJAX was used for asynchronous communication with the server and to avoid full page reloads for every action. This keeps the site speedy and responsive to user actions. Finally, a node.js server running on a Raspberry Pi with a Raspberry Pi camera was used to capture, encode, and stream the live video stream of the robot.

3.3 Software Development Methodology

The software development methodology followed for this project drew on ideas from Agile software development methodologies and, in particular, Extreme Programming(XP). It involved many short iterations each building up the web app by adding new features and improvements. The methodology used, much like XP was low on superfluous ceremony and documentation in order to remain extremely agile and open to course-corrections.

Regular meetings with the project supervisor (the de-facto product owner) and team members were held to verify that the project was proceeding as planned. The low level of documentation was accounted for by constantly re-factoring the code-base to adhere to best-practices and the appropriate design patterns. This constant re-factoring ensured that the code remained decoupled and modular and technical debt was kept to a minimum, thus making the addition of new features an easy, painless process.

Compatibility between the web app and the robot was tested after each iteration. A task queue in the form of a Trello board [1] was used to prioritise and track tasks/issues from inception to completion.

Before beginning the implementation of the web app, an initial two week experimentation period was used to investigate the available frameworks and sketch out the rough architecture of the system. At the end of this experimentation period, Python Django was selected as the web framework for this project. The video streaming feature was identified as the feature with the highest technical risk and was therefore tackled first. An initial prototype of the web app was built with the video streaming capabilities roughly implemented. This prototype was presented as the initial feasibility demo. It was then re-factored and used to seed further iterations of the project. The remaining iterations were then used to add the other required features and improvements.

3.4 System Architecture

There are three major components to the web app, viz. the website itself (consisting of the back-end and front-end), the robot, and the video stream. The website was built using Python Django, JavaScript, jQuery, and AJAX. The robot was built using a Raspberry Pi and exposes a simple Python API to control it.[2] Lastly, the live video stream was implemented using a Raspberry Pi (with an attached Raspberry Pi camera) and a node.js server. The web app makes use of the other two components and presents a unified UI to the student.

The web app back-end communicates with the robot over SSH and serves as a mediator for all interactions between the client and the robot. This allows us to manage/control access to the robot. Client interactions with the robot (through

[1]The Trello board is available at https://trello.com/b/H8UTZY8d/raspied

[2]See Appendix A regarding access to related papers

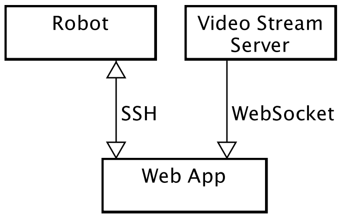


Figure 2: Architecture diagram showing the components of the web app and their relationships.

the web app) are carried out over a WebSocket connection (implemented on the server using Django Channels). This allows for real-time feedback, such as the result of print statements, to be sent to the user.

Automatically reconnecting WebSockets were used on the client side to connect to the video streaming server. All video stream connection related output (e.g. displaying the stream, telling the user that the stream is connected/disconnected, etc.) is handled on the client. The architecture diagram in Figure 2 illustrates the three components described above and their relationships.

3.5 Limitations

The inclusion of a robot simulation to mirror the actual robot was initially considered. This would have helped speed up the feedback loop by allowing students to test their code even while other students were using the robot. Unfortunately, due to time constraints, we decided not to add this proposed simulation feature to the web app.

Currently, the web app only supports a single robot; however, the infrastructure around interactions with the robot has been designed to use a robot identifier. Support for multiple shared robots can therefore be added in future with just a few minor changes to the web app.

4. IMPLEMENTATION

4.1 Web App Features [3]

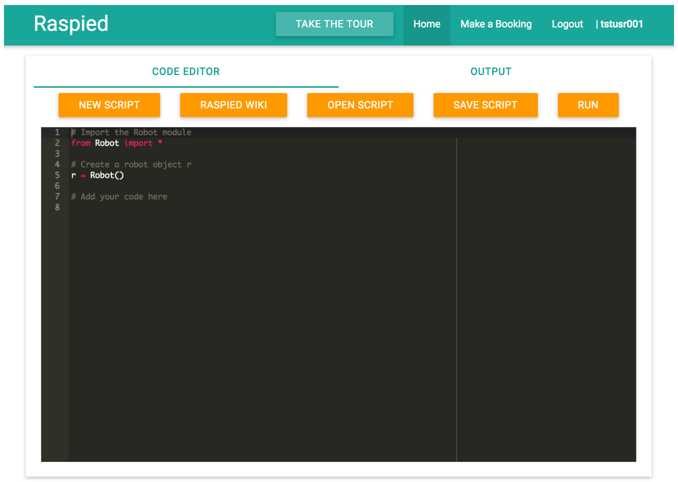
4.1.1 Access Control

We need to manage access to the robot such that only students registered for the relevant courses are able to use the robot. To this end, the course convenor can use the admin interface to upload a text file containing a list of student numbers to be marked as 'Whitelisted Usernames'. The authorization and authentication framework provided by Django was extended to only allow users with whitelisted student numbers to register on the web app. In this way, we ensure that only students who have been authorised to do so, can sign up and use the web app. The course convenor can also use the admin interface to delete users and users' bookings if necessary.

4.1.2 Code Editor

The code editor is implemented using the Ace Editor JavaScript library. It has been styled to resemble students' current IDE

[3]A git repository of the Web App code is available at https://github.com/muhummadPatel/Raspied



```
# Import the Robot module
from Robot import *

# Create a robot object r
r = Robot()

# Add your code here

```

Figure 3: Screenshot of the web app showing the code editor in the editor tab.

(Wing IDE) in order to provide them with a familiar environment in which to write their code.

The editor has been customised to show line numbers and highlight Python syntax. By default, the editor is set to contain some boilerplate code to get students started with their robot programs. This boilerplate code can be easily customised by the course convenor to suit specific assignments or sections being taught. Furthermore, the editor has been extended to include rudimentary auto-completion for the RASPIED robot API. Users' scripts are also cached at regular intervals (stored in localstorage) to prevent accidental loss of work (e.g. by refreshing the page without saving the script).

Five action buttons have been placed above the editor much like in the students' current IDE. The 'New Script' button allows students to quickly reset the code editor to the default state with the boilerplate code. This is roughly equivalent to the 'File > New Python Script' action in most Python IDE's. The 'RASPIED Wiki' button opens the documentation for the robot's Python API in a new tab. The 'Open Script' button allows users to open a script that they have saved on their computer. The 'Save Script' button downloads the contents of the code editor as a Python file. When this button is clicked the students are presented with a prompt where they can specify the name to be used for the downloaded file. Lastly, the 'Run' button will upload the user's script from the code editor to the robot and run it. The 'Run' button also automatically takes the user to the live video stream /output tab. The screenshots in Figure 3 and Figure 4 show the code editor in the editor tab as well as the output tab with the live video stream.

4.1.3 Running Code

Students can upload their code to the robot and run it with a single click. When the user clicks the Run button, their code is sent to the web app over a WebSocket connection. The web app then verifies that the user is allowed (is authorised and has the current booking) to upload code to the robot and opens an SSH session with the robot. The web app opens a Python shell on the robot (over the SSH session) and transmits the submitted code line-by-line. Any

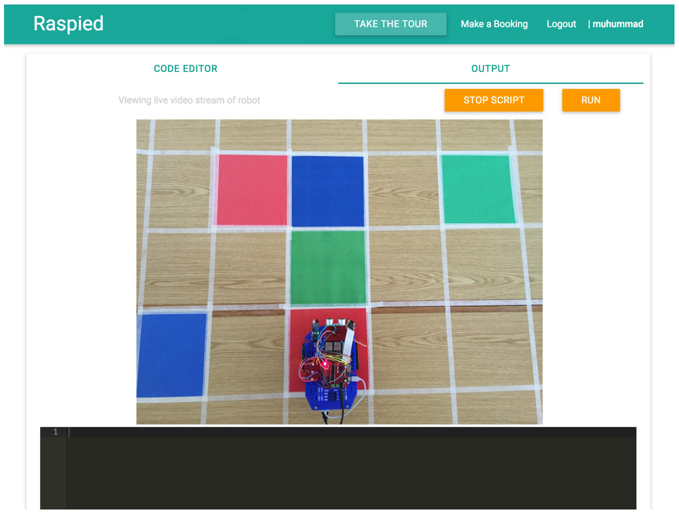


Figure 4: Screenshot of the web app showing the output tab with live video stream and output terminal at the bottom of the screen. *Output terminal truncated to conserve space*

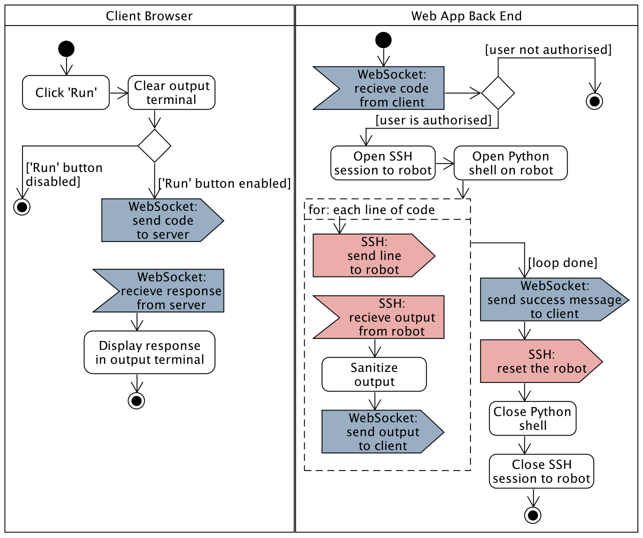

Client Browser
Click 'Run'
Clear output terminal
['Run' button disabled]
['Run' button enabled]
WebSocket: send code to server
WebSocket: recieve response from server
Display response in output terminal
Web App Back End
WebSocket: recieve code from client
[user not authorised]
[user is authorised]
Open SSH session to robot
Open Python shell on robot
for: each line of code
SSH: send line to robot
SSH: recieve output from robot
Sanitize output
WebSocket: send output to client
[loop done]
WebSocket: send success message to client
SSH: reset the robot
Close Python shell
Close SSH session to robot

Figure 5: Activity diagram showing the work-flow used to run code on the robot through the web app.

output from the robot is collected and sent back to the client over the open WebSocket.

On the client side, when the 'Run' button is clicked, the output tab is selected and the student sees the live video stream of the robot. The output terminal area below the video is cleared of any text from previous runs and it informs the user that their code is being uploaded to the robot. As the lines of the user's code are run, they are displayed in the terminal output area along with any output they produced (e.g. from print statements or errors). When the user's code has completed, a message to this effect is displayed on screen to inform the user that their code has been executed and the robot is now being reset to its original position. This work-flow is illustrated in the activity diagram in Figure

In order to enable a faster feedback loop and allow students to correct runaway code, a 'Stop Script' that kills the current program being executed has been provided above the live video stream. When this 'Stop Script' button is clicked, a new SSH session to the robot is opened on the back end. We then search for and kill the offending Python process and run a custom recovery script to recover and reset the robot. This functionality is discussed further in Section 4.2.1. All SSH sessions to the robot are then closed and the user is shown a message indicating that their code has been successfully halted and the robot has been reset and is ready for use.

4.1.4 Video Stream

The web app presents students with a live video stream of the robot so that they can see their code running on the robot in real-time. This video stream is captured using a Raspberry Pi camera and is encoded by the connected Raspberry Pi using ffmpeg. This set-up was packaged as a standalone unit requiring only a stable internet connection and a power source.

A node.js server on the Raspberry Pi listens for incoming WebSocket connections and sends out the encoded video stream through these connected sockets.[4] The web app server provides the client with the streaming server address and the client opens a WebSocket connection and receives the video stream from the streaming server. This video stream is then displayed to the user in an HTML5 canvas element using the jsmpg JavaScript library.

4.1.5 Booking System

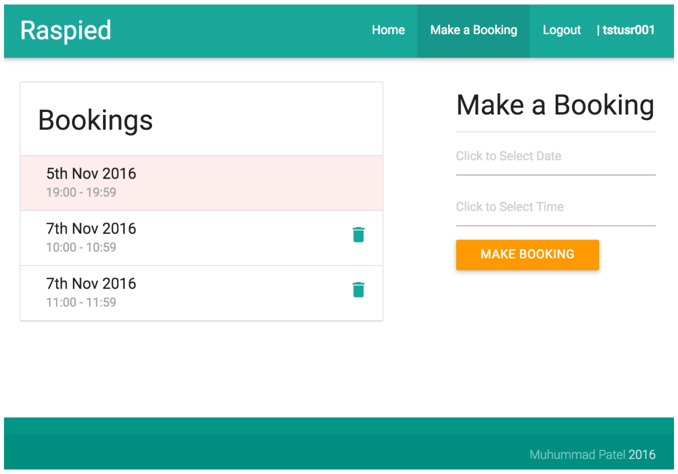
In order to facilitate shared access to the single robot, the web app uses a booking system whereby students are required to book/reserve the robot before uploading and running their code on it. On the client side, the 'Run' button is disabled when the student does not have the current booking, and on the server side, we check that the user trying to upload and run code on the robot has the current booking. Trying to upload code to the robot without having the current booking will result in a 'robot access denied' error message being returned to the user.

In order to make a booking, students navigate to the bookings page and select a date and time for the booking. The bookings page including the booking form are shown in Figure 6. The booking interval defaults to a one hour session; however, this booking interval can be easily customised by the course convenor as necessary.

The date and time picker used in the booking form have been designed to only allow students to select valid bookings. The date picker will only allow students to choose either today or a future date while the time picker will only allow students to choose future times that have not yet been booked. Booking requests are further validated when they arrive at the back end to ensure that only valid bookings are successfully added to the database.

On the bookings page, users can also see a sorted list of their upcoming bookings and the details of those bookings. If the user has booked a session that is currently under way, that session will be highlighted as seen in Figure 6. Bookings

[4]Video streaming server code is available at https://github.com/muhummadPatel/Raspied_StreamServer


Raspied

Home | Make a Booking | Logout | tstusr001

Bookings

5th Nov 2016
19:00 - 19:59

7th Nov 2016
10:00 - 10:59

7th Nov 2016
11:00 - 11:59

Make a Booking

Click to Select Date

Click to Select Time

MAKE BOOKING

Muhummad Patel 2016

Figure 6: Screenshot of the web app showing the bookings page including the new booking form.

from the list of upcoming bookings can also be deleted by clicking the delete icon on the list item. Note that the delete icon is obviously not available for the current booking as this would be unfair to other users.

The addition and deletion of bookings is implemented asynchronously using jQuery and AJAX. This means that these actions do not require a full page refresh and the list of upcoming bookings is updated as the actions are processed. For example, when a booking is successfully deleted, it will be automatically removed from the list of upcoming bookings and similarly, when a booking is successfully added, it will automatically be added to the list. The success or failure of any attempted booking additions and deletions is communicated to the user clearly via notification banners that self-dismiss after a few seconds.

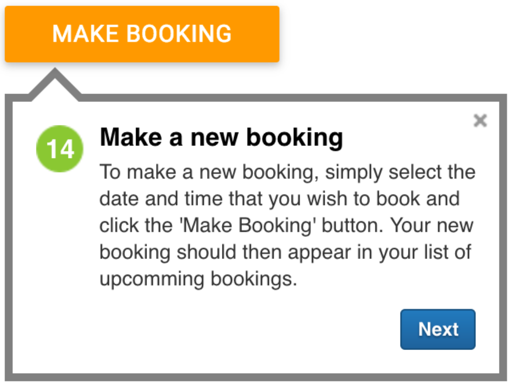
The history of bookings is available for audit purposes through the admin interface. The course convenor can see which students have made which bookings and can delete bookings if necessary. The course convenor can also adjust the number of bookings per month that students are allowed to make. By default, students are only allowed to make five bookings per one month period. This has been done to prevent students from monopolising the robot and excluding their peers.

4.1.6 *Web App Tour*

The web app makes use of a website tour feature to introduce students to the web app and its various features. When the user completes the registration process and is logged in to their home page for the first time, the website tour will start automatically. If the user wishes to redo the tour at any other time, they can simply click the 'Take Tour' button in the navigation bar.

The tour was implemented using using the Hopscotch.js JavaScript library from LinkedIn. During the tour, small hovering boxes with useful information are attached to the relevant site components as seen in Figure 7. The student can easily stop the tour at any point by clicking the 'x' in the top-left corner of the information boxes.

The tour walks students through the important areas of the main page, including the editor and associated action buttons, the live video stream, and the output terminal.


MAKE BOOKING

14

Make a new booking

To make a new booking, simply select the date and time that you wish to book and click the 'Make Booking' button. Your new booking should then appear in your list of upcomming bookings.

Next

Figure 7: Screenshot showing step fourteen of the website tour with information regarding the 'Make Booking' button.

It also automatically walks students through the bookings page and explains the booking system and the need to make a booking before using the robot. At the end of the tour, the student will have been introduced to all the important features of the web app and will be ready to perform their assigned programming tasks.

4.2 Technical Challenges and Implementation Decisions

4.2.1 *Resetting and Recovering the Robot*

To provide every user with a consistent experience and to make designing exercises easier, we needed to be able to reset the robot to its starting position. There are two situations where a robot will need to be reset: when the the user's code has completed execution, or when the user clicks the 'Stop Script' button and we need to forcibly halt the execution of their script.

In order to support this functionality, the robot keeps track of its starting position and its current position. This position information is stored in a temporary file on the robot file-system which means that it is independent of the Python process that is controlling the robot. This temporary file is then updated every time the robot's positioning - including rotation - changes.

When the user submits their code to be run on the robot, the web app back end appends a few lines to their code. These lines of code call the robot's reset method which frees up the robot's resources and ensures that it returns to its original starting position. It is important to note that this reset method was designed to be idempotent and the Robot class of the robot API was designed as a singleton. This means that appending the clean-up code to the user's script is safe because when we create a Robot instance, we will either get the existing instance or a new one (thus avoiding any hardware resource allocation conflicts). Furthermore, when we call the reset method, it will behave well even if the user has already called the reset method themselves (because the reset method is idempotent). Appending clean-up and reset code to the user's script takes care of the first situation that was identified; i.e. when the user's code has completed execution and we need to reset the position of the robot.

In the second case, where the user has forcibly halted the code execution using the 'Stop Script' button, the web app will open an SSH session to the robot and will kill the offending Python process. Note that because we forcibly killed the python process, the robot's reset method will not have been called and the robot will be stranded in its current position. To fix this, we then call a custom recovery script that frees up any resources that have not yet been freed up. This recovery script then reads in the robots last position from the temporary file and moves the robot to this position. In both cases, the user is informed when their code ends and the robot is resetting itself to avoid confusion.

4.2.2 *Running Code on the Robot*

The web app and the robot were developed as distinct subsystems. When the user wants to run their code on the robot, it needs to be transferred from the client side to the robot. the pxssh Python library was used to facilitate communication - via SSH sessions - between the web app back end and the robot. The web app can therefore open an SSH session on the robot and control it through that SSH session.

It is important to note that these long running methods (that make use of SSH sessions) on the server are implemented using a task queue and separate worker processes. This means that they do not block or significantly slow down the server from handling other incoming requests. Furthermore, we can simply spawn more worker processes to handle heavier loads (for example if the system is scaled up to have multiple robots).

The main issues considered when designing this functionality were transferring the code from the client to the robot and transferring feedback from the robot back to the client. We identified two approaches to transferring the code to the robot: run the entire script as a single file, or run the script one line at a time. Running the entire script as a single file meant that transferring the file across to the robot was simpler; however this also resulted in all output being buffered and returned to the user only once the entire script had completed. An early prototype of this approach uploaded the script from the web app to a private GitHub Gist and then opened an SSH session on the robot to pull down the user script from the gist and run it. The output of the entire script was then collected and sent to the user after the script had completed but this was not ideal as it could be confusing to the user.

The final approach used to implement this feature was to run the user script line-by-line in a Python shell on the robot. To do this, the web app back end splits the submitted script into its constituent lines. The web app then opens an SSH session with the robot and starts a Python shell. Each line of the user script is then evaluated one at a time with the output of each line being sent back to the client as soon as it is received. This approach was chosen as it allows us to display the output of the user's code in real-time which is much less confusing for the students.

4.2.3 *Streaming the Video*

A Raspberry Pi with an attached Raspberry Pi camera was used to capture the video for the live video stream. These hardware choices kept the cost down and resulted in a portable, easily reproducible setup.

Some of the streaming solutions investigated included using ffserver with nginx, and HTTP Live Streaming(HLS). The main issues considered were high latency/laggy video as well as ease of use. The final approach to the live streaming used ffmpeg to encode the captured video. A custom Node.js server was then used to stream the encoded video over incoming WebSocket connections. On the client side, once the WebSocket connection to the video stream server has been established, the jsmpg JavaScript library is used to display the video in an HTML5 canvas. This solution was relatively easy to set up and resulted in a live stream with a very small amount of lag. This set up was tested between Cape Town and Durban and the observed video stream lag was found to be acceptable.

The major challenge in implementing this feature as described above, was getting ffmpeg and x264 (for encoding the video) on the Raspberry Pi. There were, at the time of writing, no pre-built binaries for the Raspberry Pi and we therefore had to build these libraries from source. Unfortunately, trying to compile these enormous libraries from source on the Raspberry Pi itself was found to be wholly infeasible as it would be extremely time consuming (many, many hours). The software therefore had to be cross-compiled on a faster machine and then transferred over to the Raspberry Pi. In particular, a Vagrant box (a headless virtual machine) was set up to provide a safe sandbox - with the required tooling - in which to compile the code. The compiled binaries were then transferred over to the Raspberry Pi.

5. EVALUATION

The web app developed for this project has been evaluated based on its usability and whether or not it meets the initial requirements as laid out in Section 3.1.

5.1 Testing Methods

5.1.1 *Measuring Usability*

The System Usability Scale(SUS)[4] was used to determine the usability of the web app. The SUS is a survey consisting of ten simple questions (using a Likert scale) that is presented to users after they have been allowed to interact with the system. Responses to these questions are then scored to produce a single value representing a measure of the overall perceived usability of the system. These SUS scores range between 0 and 100, with 100 being the maximum (subjectively very good usability).

The SUS was chosen as it is a well-established standard for testing the usability of systems, particularly in the context of computer systems. The SUS survey used in this project also had an extra question (not used to determine the score) asking users to rate how likely - on a scale from 0 to 10 - they were to recommend this system to a friend. This question was added to get a better sense of the users' overall sentiment towards the system.

5.1.2 *Testing Procedure*

Two iterations of usability testing were conducted with both iterations following the same procedure. Participants would arrive at the designated testing location at the arranged time. We then discussed the details of the test and what it would involve. Participants were then asked to sign an informed consent form and advised of their right to withdraw at any time should they wish to. The participants were presented with four programming tasks to complete using the web app to program the robot. These four tasks were designed to make use of the robot and the features to demonstrate and teach a particular fundamental CS concept (e.g. loops). The participant was asked to register on the web app and complete the task while thinking aloud if necessary. We then observed the participant and noted all critical incidents. Once the participant had completed the tasks, they were presented with the SUS survey for the web app. Once the participant had completed the survey, they were debriefed and any general feedback and suggestions was recorded. Finally, the participant was thanked for their time and the session was concluded. The average duration of the testing sessions was roughly 45 minutes.

The feedback from the first iteration of testing was then compiled and recurring pain points, comments, and suggestions were identified. These were then used to formulate a list of improvements to the web app. This list of improvements has been provided in Appendix B.

The identified improvements were then implemented and a second round of testing was conducted. The second round of testing was to verify that the identified issues had been resolved and to measure the usability of the improved final version of the web app. The average SUS score obtained from the second round of testing could then be compared to that from the first round of testing to verify that the system had in fact been improved. The final average SUS score was then used to determine the final usability of the system based on the adjective mapping proposed by Bangor, A.[1].

5.1.3 Participants

The participants chosen to test the usability of the web app were all Computer Science(CS) students [5]. For the first round of testing, a group of seven participants, consisting of five CS honours students and two CS masters students, were surveyed. Unfortunately, one of the masters students was unable to participate in the second round of testing.

All selected participants had, in some capacity, served as tutors and/or teaching assistants for first year Computer Science courses. The participants were thus well positioned to comment on the usability of the web app from the perspective of typical first year students. Furthermore, in their capacity as tutors and/or teaching assistants, the participants were also able to view the web app as a teaching tool and could comment on its usability relative to other similar tools currently being used.

5.1.4 Limitations

Ideally the usability of the web app would have been tested with actual first year students, who are the actual target demographic of the system. Unfortunately, due to unforeseen circumstances and time constraints, this was not possible. We tried to mitigate this by choosing participants that were the next closest substitute, i.e. the teaching assistants and tutors for those first year courses.

Due to limited resources, the usability testing was only conducted on a relatively small group of participants. As the robot could only be used by a single participant at a time, the testing could not be conducted in parallel. Furthermore, due to space restrictions, the testing area had to be set up and taken down for each testing session. This meant that the participants were only able to interact with the web app for the duration of their testing session. Had the participants had more time to interact with the system, we may have been able to solicit more feedback to further improve the web app.

[5]Ethical clearance was obtained for the use of UCT students as participants in the usability study. The approved ethical clearance forms can be accessed on the RASPIED project website as indicated in Appendix A.

Participant #	Recommendation Score	SUS Score
1	10	95
2	8	82.5
3	9	90
4	8	87.5
5	9	72.5
6	8	65
7	8	90
Average	**8.6**	**83.2**

Table 1: SUS and recommendation scores obtained from the first testing session.

Participant #	Recommendation Score	SUS Score
1	10	97.5
2	10	100
3	9	92.5
4	8	87.5
5	9	90
6	8	80
Average	**9**	**91.3**

Table 2: SUS and recommendation scores obtained from the second testing session.

5.2 Findings and Results

5.2.1 Meeting Initial Requirements

The final web app produced in this project has successfully met the core requirements laid out in the initial plan. The web app provides students with an editor to write and edit their Python code. It also allows them to upload this code to the robot and run it. A live video stream of the robot is also provided so that students can see how the robot behaves when running their code. The web app successfully manages shared access to the robot by only allowing authorised students to register and requiring students to book sessions with the robot before uploading code to it.

Based on consultations with the CSC1010H course convenor and product owner, the web app was deemed to have met the required level of functionality. Furthermore, the feedback received from the expert users in second round of testing was overwhelmingly positive and they indicated that this would be a useful tool for teaching introductory Computer Science courses. Thus, the web app has successfully met the initial requirements and provides the necessary functionality to allow students to interact with the robot.

5.2.2 Usability Results

The SUS was was used to measure the usability of the web app in both rounds of testing. Additionally, users were asked to rate how likely they were to recommend the web app to others on a scale from 0 to 10, with 0 being ’not at all’ and 10 being ’definitely’. This likelihood of recommendation rating will hereafter be referred to as the ’recommendation score’.

The web app obtained an average SUS score of 83.2 and an average recommendation score of 8.6 in the first round of testing. Both these scores showed significant improvement in the second round of testing with the final average SUS score being 91.3, and the final average recommendation score being 9 as seen in Table . The results for the first and second testing sessions are illustrated in Table 1 and Table 2 respectively.

In order to interpret the final SUS score, we will rely on the work of Bangor, Kortum, and Miller[1] who present a mapping from SUS scores to the letter grade scale and an adjective rating scale as shown in Figure 8. Based on this scale, we see that the final usability score of 91.3 falls well into the 'acceptable' region, with a letter grade score of 'A', and an adjective rating of 'excellent'. From this we see that the final version of the web app developed for this project has, by all indications, an 'excellent' usability score and should serve the target users well.

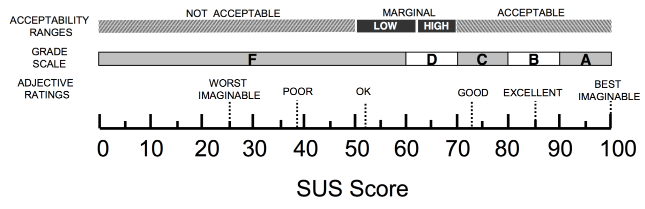


Figure 8: Mapping from SUS scores to an adjective scale for easier interpretation[1].

6. CONCLUSIONS AND FUTURE WORK

This project involved the design, implementation, and evaluation of the web app component of the RASPIED educational robot platform. A fully functional web app was developed to allow students to write and upload code to the robot developed as part of the larger RASPIED project, provide a live video stream of the robot, and manage shared access to the robot. The web app also provides the course convenor with the ability to manage the users and their bookings and authorise which students are allowed to sign up.

The web app was implemented using Python Django with a SQLite database for the back end and HTML, CSS, and JavaScript with jQuery and AJAX for the front end. The developed web app is fully functional and can be easily deployed with minimal configuration required. The software methodology used in this project was based on the Extreme Programming methodology with minimal ceremony and documentation, regular meetings with project stakeholders and quick, short iterations.

Two rounds of usability testing were conducted with seven participants in the first round and six participants in the second round. All participants were expert users who had, in some capacity, served as either tutors and/or teaching assistants for introductory Computer Science courses. The usability of the web app was measured using the SUS survey and the final average SUS score obtained was 91.3 which can be described as 'excellent'. Users also rated the likelihood of them recommending the app to others, on a scale from 0 to 10. The average recommendation score from the second round of testing was 9.

The web app developed in this project has successfully met initial requirements and has achieved an 'excellent' usability score. We have achieved the initial goal of this project, which was to develop a supporting web app for the RASPIED educational robot platform. We hope that this work will be of benefit to future generations of budding Computer Scientists.

While we consider this project to have been a success, there is still room for future work. Some improvements that could be tackled by future projects include adding a simulator to the web app. This would speed up the feedback loop for students by allowing them to program the robot simulation when other students have booked the actual physical robot. The web app could also be extended to use multiple physical robots. This would involve more robots and changes to the front end to change the way users book robots and upload/run their code. Further improvements could also be found by testing with actual first year students. Their perspective would no doubt highlight new lines of inquiry for the web app.

7. ACKNOWLEDGEMENTS

I would like to thank my team members, Jeremy Coupland and Josh Di Bona, for their continued patience, good humour, and assistance. I would also like to thank my supervisor, Mr. Gary Stewart for his invaluable guidance without which this project could not have been completed.

8. REFERENCES

[1] A. Bangor, P. Kortum, and J. Miller. Determining what individual sus scores mean: Adding an adjective rating scale. *Journal of usability studies*, 4(3):114–123, 2009.

[2] D. Baum and J. Hansen. Nqc programmer's guide. 2003.

[3] D. S. Blank and D. Kumar. Assessing the impact of using robots in education, or: How we learned to stop worrying and love the chaos. In *AAAI Spring Symposium: Educational Robotics and Beyond*, 2010.

[4] J. Brooke et al. Sus-a quick and dirty usability scale. *Usability Evaluation in Industry*, 189(194):4–7, 1996.

[5] S. Cooper, W. Dann, and R. Pausch. Alice: a 3-d tool for introductory programming concepts. In *Journal of Computing Sciences in Colleges*, volume 15, pages 107–116. Consortium for Computing Sciences in Colleges, 2000.

[6] B. Fagin. An ada interface to lego mindstorms. *ACM SIGAda Ada Letters*, 20(3):20–40, 2000.

[7] B. Fagin. Ada/mindstorms 3.0. *Robotics & Automation Magazine, IEEE*, 10(2):19–24, 2003.

[8] B. Fagin and L. Merkle. Measuring the effectiveness of robots in teaching computer science. In *ACM SIGCSE Bulletin*, volume 35, pages 307–311. ACM, 2003.

[9] F. Klassner. A case study of lego mindstorms' suitability for artificial intelligence and robotics courses at the college level. In *ACM SIGCSE Bulletin*, volume 34, pages 8–12. ACM, 2002.

[10] D. Kumar, D. S. Blank, T. R. Balch, K. J. O'Hara, M. Guzdial, and S. Tansley. Engaging computing students with ai and robotics. In *AAAI Spring Symposium: Using AI to Motivate Greater Participation in Computer Science*, pages 55–60, 2008.

[11] T. Lauwers, I. Nourbakhsh, and E. Hamner. Csbots: design and deployment of a robot designed for the cs1 classroom. In *ACM SIGCSE Bulletin*, volume 41, pages 428–432. ACM, 2009.

[12] L. Major, T. Kyriacou, and O. P. Brereton. Systematic literature review: teaching novices programming using robots. *Software, IET*, 6(6):502–513, 2012.

[13] D. J. Malan and H. H. Leitner. Scratch for budding computer scientists. In *ACM SIGCSE Bulletin*, volume 39, pages 223–227. ACM, 2007.

[14] S. A. Markham and K. King. Using personal robots in cs1: experiences, outcomes, and attitudinal influences. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*, pages 204–208. ACM, 2010.

[15] R. E. Pattis. *Karel the robot: a gentle introduction to the art of programming*. John Wiley & Sons, Inc., 1981.

[16] M. Saleiro, B. Carmo, J. M. Rodrigues, and J. H. du Buf. A low-cost classroom-oriented educational robotics system. In *Social Robotics*, pages 74–83. Springer, 2013.

[17] SQLite. SQLite Website: Use Cases. https://sqlite.org/whentouse.html.

[18] A. Yadin. Reducing the dropout rate in an introductory programming course. *ACM inroads*, 2(4):71–76, 2011.

APPENDIX

A. RELATED RASPIED PAPERS

The RASPIED project was developed as part of three projects: The robot, the robot features, and the web app. This paper presents the development of the web app. Papers discussing the development of the robot and the robot features can be found on the RASPIED project web page which can be accessed at http://pubs.cs.uct.ac.za under the 'Honours Project Archive' section for 2016.

B. LIST OF IMPROVEMENTS

The feedback from the first iteration of testing was used to compile a list of recurring pain points, comments, and suggestions. These were then used to formulate a list of improvements to the web app that were implemented before the second testing session. These improvements are listed below:

- Improve the registration and login work-flows
- Add boilerplate code to the editor to allow students to get started faster
- Add a 'Reset' button to reset the contents of the editor to its default boilerplate code
- Fix the site responsiveness, with a focus on the navbar, for smaller screen sizes (some users' were observed to be using the web app on half the screen with the documentation open on the other half which caused the navbar to behave unexpectedly)
- Add a button to forcibly halt the currently executing script
- Add basic auto-completion for the robot API functions to the code editor
- Provide the user with more feedback to indicate when their code ends and the robot is resetting itself
- Automatically start the website tour when the user first logs in
- Add a 'Run' button above the video stream to speed up the feedback loop when users want to re-run their code
- Save the current user script in the background so the user doesn't lose their work when they refresh the page
- Clear the output terminal after each run to avoid confusion
- Show the current booking in the list of upcoming bookings
- Improve the feedback provided to the user when making a booking and streamline this process
- Allow the user to specify a file-name when saving scripts to their machine
- Various bug-fixes and UI rearrangements to increase productivity based on observed user work-flows