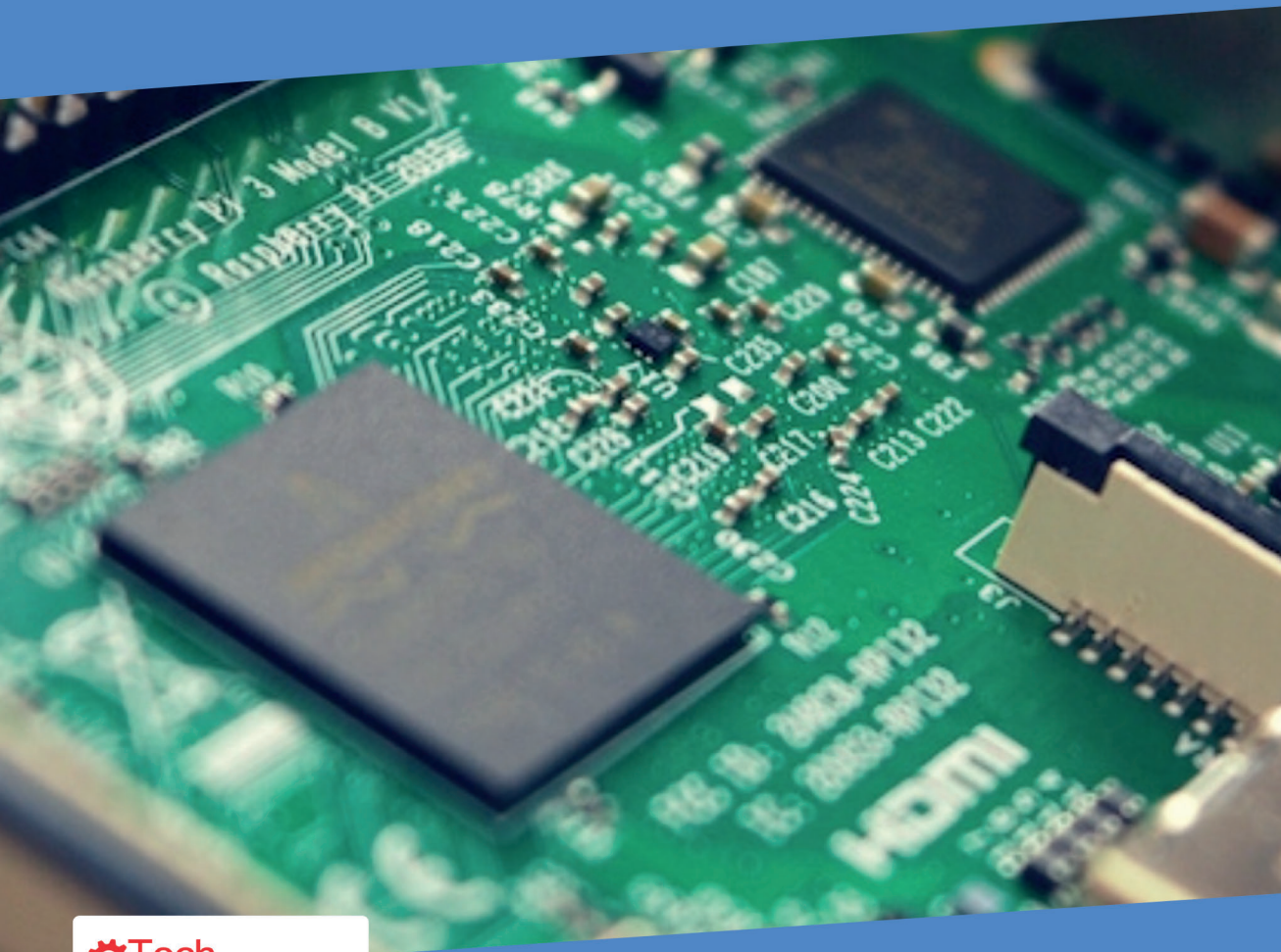


A comprehensive course that will teach
you how to build a modern IoT application

Raspberry Pi Full Stack



Raspberry Pi Full Stack



Dr. Peter Dalmaris



elektor

LEARN > DESIGN > SHARE

● This is an Elektor Publication. Elektor is the media brand of
Elektor International Media B.V.
78 York Street, London W1H 1DP, UK
Phone: (+44) (0)20 7692 8344

● All rights reserved. No part of this book may be reproduced in any material form, including photocopying, or storing in any medium by electronic means and whether or not transiently or incidentally to some other use of this publication, without the written permission of the copyright holder except in accordance with the provisions of the Copyright Designs and Patents Act 1988 or under the terms of a licence issued by the Copyright Licensing Agency Ltd., 90 Tottenham Court Road, London, England W1P 9HE. Applications for the copyright holder's permission to reproduce any part of the publication should be addressed to the publishers.

● Declaration

The author and publisher have used their best efforts in ensuring the correctness of the information contained in this book. They do not assume, or hereby disclaim, any liability to any party for any loss or damage caused by errors or omissions in this book, whether such errors or omissions result from negligence, accident or any other cause..

● British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

Cover designer: Harmen Heida

Prepress Production: D-Vision, Julian van den Berg

Printed in the Netherlands

First Printing: 2020

ISBN: 978-1-907920-95-0

ISBN (PDF): 978-3-89576-378-6

ISBN (epub): 978-3-89576-379-3



Elektor is part of EIM, the world's leading source of essential technical information and electronics products for pro engineers, electronics designers, and the companies seeking to engage them. Each day, our international team develops and delivers high-quality content - via a variety of media channels (including magazines, video, digital media, and social media) in several languages - relating to electronics design and DIY electronics. www.elektormagazine.com

LEARN > DESIGN > SHARE

About the author	10
How to read this book	11
Requirements	11
The book web page	12
Did you find an error?	12
Part 1: Getting started with the Raspberry Pi Full Stack	13
Chapter 1 • What is this book about?	14
Chapter 2 • A walk-through the Full Stack project.	16
Chapter 3 • Required hardware.	20
Chapter 4 • How to get help.	24
Chapter 5 • The code repository	26
Part 2: Raspberry Pi, Arduino, and Raspberry Pi Zero W	27
Chapter 6 • Raspberry Pi vs Arduino high level comparison	28
Chapter 7 • Need for efficiency: the Raspberry Pi Zero W.	33
Chapter 8 • Need for speed: the Raspberry Pi 4 (and 3)	36
Part 3: How to set up the operating system.	38
Chapter 9 • Operating systems for the Raspberry Pi	39
Chapter 10 • What is a 'headless' operating system	42
Chapter 11 • How to download and install Raspbian	45
Chapter 12 • How to set up SSH and WiFi in headless mode	48
Chapter 13 • How to set a hostname.	50
Chapter 14 • Boot into Raspbian for the first time.	53
Chapter 15 • How to set a fixed IP address	58
Chapter 16 • Basic configuration.	59
Chapter 17 • Working as the 'root' user	61
Part 4: How to backup and restore your SD card	64
Chapter 18 • Backup an SD card - MacOS	65
Chapter 19 • Restore an SD card - MacOS	67
Chapter 20 • Backup an SD card - Windows	69
Chapter 21 • Restore an SD card - Windows.	71
Part 5: Pins, GPIOs and how to control them with Python	73
Chapter 22 • Raspberry Pi pins, roles, and numbers	74

Chapter 23 • A taste of Python on the Command Line Interpreter	76
Chapter 24 • Python functions	80
Chapter 25 • A simple Python program	85
Chapter 26 • Wire a simple circuit.	91
Chapter 27 • Control an LED with GPIOZERO	94
Chapter 28 • Control an LED with rpi.gpio	97
Chapter 29 • Read a button with GPIOZERO.	100
Chapter 30 • Read a button with RPi.GPIO	102
Chapter 31 • Control an LED with a button.	104
Chapter 32 • Set up the DHT22 sensor with Git	107
Chapter 33 • Use the DHT22 sensor	111
Part 6: Set up the Web Application Stack.	114
Chapter 34 • The Web Application Stack	115
Chapter 35 • The Python Virtual Environment.	119
Chapter 36 • Increase the disk swap file size	122
Chapter 37 • Set up system Python - preparation	124
Chapter 38 • Download, compile and install Python 3	125
Chapter 39 • Set up the app Python virtual environment	129
Chapter 40 • Set up Nginx	131
Chapter 41 • Set up Flask	133
Chapter 42 • A tour of a simple Flask app	136
Chapter 43 • UWSGI installation	138
Chapter 44 • Nginx configuration	140
Chapter 45 • UWSGI configuration	143
Chapter 46 • UWSGI and Nginx configuration testing	146
Chapter 47 • Configure systemd to auto-start uwsgi	148
Part 7: Set up the database	151
Chapter 48 • Install the SQLite3 database.	152
Chapter 49 • Hand-on with the SQLite3 CLI	154
Part 8: Styling with Skeleton	158
Chapter 50 • Static assets and the Skeleton boilerplate CSS	159
Chapter 51 • Set up the static assets directory.	162

Chapter 52 • Introducing the Skeleton boilerplate CSS	164
Chapter 53 • Copying files using SFTP	168
Chapter 54 • Flask templates	172
Chapter 55 • Debugging a Flask app	176
Part 9: Capture and record sensor data	179
Chapter 56 • Introduction to Part 9	180
Chapter 57 • Install the DHT library and the rpi-gpio module	181
Chapter 58 • Display the current sensor values in the browser	184
Chapter 59 • Create a database to store sensor data.	189
Chapter 60 • Capture sensor data with a Python script	191
Chapter 61 • Schedule sensor readings with Cron	194
Chapter 62 • Update the application and template file	196
Part 10: Implement the date range selection feature	201
Chapter 63 • Introduction to Part 10	202
Chapter 64 • Prototype datetime range of records in SQLite CLI	204
Chapter 65 • Prototype datetime range in the browser	206
Chapter 66 • URL querystring validation	210
Chapter 67 • Quick tidying up.	214
Chapter 68 • Use radio buttons for easy timedate range selection.	218
Chapter 69 • Provision the Python script to work with the radio buttons	221
Part 11: Google Charts and datetime widgets	224
Chapter 70 • Introduction to Part 11	225
Chapter 71 • Implement Google Charts	229
Chapter 72 • Test Google Charts	234
Chapter 73 • The datetime picker widget	236
Chapter 74 • Implement the datetime picker widget	238
Chapter 75 • Test the datetime picker widget	241
Part 12: Dealing with time zones	243
Chapter 76 • Adjust datetimes to local time zone on the client side.	244
Chapter 77 • Introduction to Arrow	248
Chapter 78 • Implement Arrow	251
Chapter 79 • Upload timezone changes and test.	254

Chapter 80 • Link the two pages of the application	256
Part 13: Charting with Plotly	259
Chapter 81 • What is Plotly and how to install it	260
Chapter 82 • Try out Plotly on the CLI	264
Chapter 83 • Implement Plotly support on the client side.	267
Chapter 84 • Add Plotly support on the server side	272
Chapter 85 • How to debug Javascript	276
Chapter 86 • Server side debugging example	281
Part 14: Access your application from the Internet.	284
Chapter 87 • How to access your application from the Internet?	285
Chapter 88 • Set a static IP address	287
Chapter 89 • Expose your app to the Internet with port forwarding.	290
Chapter 90 • Create a self-signed certificate for application	295
Chapter 91 • Edit Nginx configuration to use SSL	300
Chapter 92 • Test SSL in Firefox, Safari, Chrome	303
Part 15: Data logging with Google Sheet	307
Chapter 93 • What is data logging, and why Google Sheet?	308
Chapter 94 • Set up Google API credentials	311
Chapter 95 • Set up the Python libraries and Google Sheet	322
Chapter 96 • Implement of Google Sheet data logging	328
Part 16: Set up a remote Arduino sensor node with the nRF24.	330
Chapter 97 • Why set up an Arduino remote node?	331
Chapter 98 • The Arduino node wiring	333
Chapter 99 • The Arduino node sketch.	336
Chapter 100 • Raspberry Pi and nRF24 wiring	340
Chapter 101 • The Raspberry Pi nRF24 receiver script	343
Chapter 102 • How to install the Python nRF24 modules on the Raspberry Pi.	348
Chapter 103 • Test the nRF24 communications.	353
Chapter 104 • Modify the front end of the application to show remote node data	355
Part 17: If This Then That alerts	359
Chapter 105 • An introduction to If This Then That	360
Chapter 106 • Create an IFTTT webhook and applet	362

Chapter 107 • Add IFTTT code in the application and testing	371
Chapter 108 • Install the node listener script as an systemd service	374
Part 18: Wrapping up	376
Chapter 109 • Make lab_env_db page update every 10 minutes	377
Chapter 110 • Recap and what's next	378
Part 19: Project extension: Text messaging using Twilio	380
Chapter 111 • What is this project extension all about?	381
Chapter 112 • An introduction to Twilio	383
Chapter 113 • Set up a Twilio account	385
Chapter 114 • Create a useful bash shell script.	391
Chapter 115 • Add Twilio support to Raspberry Pi	394
Chapter 116 • Install Twilio CLI	396
Chapter 117 • Create local and public DNS hostnames	400
Chapter 118 • Create trusted SSL/TLS certificate	405
Chapter 119 • Send text alert messages	413
Chapter 120 • Receive text message commands.	416
Index	426

About the author

Dr. Peter Dalmaris is an educator, electrical engineer, electronics hobbyist, and maker. He is the creator of online video courses on DIY electronics and author of several technical books. Peter is the author of 'Maker Education Revolution', a book about how making is changing the way we learn and teach in the 21st century.

As a Chief Tech Explorer since 2013 at Tech Explorations, the company he founded in Sydney, Australia, Peter's mission is to explore technology and help educate the world.

Tech Explorations offers educational courses and Bootcamps for electronics hobbyists, STEM students, and STEM teachers.

A lifelong learner, Peter's core skill lies in explaining difficult concepts through video and text. With over 15 years of tertiary teaching experience, Peter has developed a simple yet comprehensive teaching style that students from all around the world appreciate.

His passion for technology and the world of DIY open-source hardware has been a dominant driver that has guided his personal development and his work through Tech Explorations.

How to read this book

All examples, descriptions and procedures have been tested on a Raspberry Pi Zero W, running Raspbian Buster. I have also tested the exact same project on a Raspberry Pi 4, Raspberry Pi 3 and Raspberry Pi 1.

This book has a web page with resources designed to maximise the value it delivers to you, the reader. Please read about the book web page, what it offers and how to access it in the section 'The book web page', later in this introductory segment.

Finally, you may be interested in the video course version of this book. This course contains detailed demonstrations and explanations of the Full Stack project. The video lectures capture techniques and procedures that are just not possible to do in text.

Please check in the book web page for updates on this project. Be sure to subscribe to the Tech Explorations email list so I can send you updates.

Requirements

To make the most out of this book, you will need a few things. You probably already have them.

The list of hardware requirements is available on the Tech Explorations website:

<https://techexplorations.com/parts/rpifs-parts/>

Please source these items before you embark on this project.

The book web page

As a reader of this book, you are entitled access to its online resources.

You can access these resources by visiting the book's web page at <http://txplo.re/rpifsp>.

The two available resources are:

1. **The book discussion space on the Tech Explorations Community.** This is a place where you can ask book-related questions and have a conversation about your projects. I will be spending time in the forum weekly, answering questions and participating in discussions.
2. **An errata page.** As I correct bugs, I will be posting information about these corrections in this page. Please check this page if you suspect you have found an error. If you have found an error that is not listed on the errata page, please use the error report form in the same page to let me know about it.

From time to time, I will be posting additional Raspberry Pi Full Stack resources and updates on this page, so please check regularly. By subscribing to the Tech Explorations email list, you'll be sure to receive my regular book updates and news. The subscription form is in the book page.

Did you find an error?

Please let us know.

Using any web browser, go to <http://txplo.re/rpifsp>, and submit a ticket.

Please take care to provide enough details in your ticket so I can find the bug. Please remember to include the page number of the PDF version of the book.

I'll get it fixed right away.

Part 1: Getting started with the Raspberry Pi Full Stack

Chapter 1 • What is this book about?

Think of this book as your guide to a adventure for determined learners.

Yes, an adventure.

An adventure of courage, determination, and collaboration.

If you choose to stay to path, you will be rewarded with knowledge.

And, with my admiration.

Because this adventure is not going to be a walk in the park.

You will struggle.

You will fail.

Many times.

You will curse me :-)

But each time, you will get up, dust yourself off, and continue your journey to complete this full stack application.

I will be there to help you at every step of the way.

I will guide you and show you the way.

But you will have to do the hard walk.

You will have to write the code, solder the circuits, test the connections and search the database

At the end, you will have more than knowledge: you will feel the sweet sense of accomplishment, and you will know that you are strong enough to conquer the next challenge.

The adventure I am talking about is a project, in which your weapon is a Raspberry Pi. Your objective is to create a useful application that runs on the Raspberry Pi and spans the Internet, and your enemy is your lazy self.

While most books are filled with micro-projects, this one is the opposite. The whole book is one BIG project.

This project brings together multiple technologies: electronics, computers, operating systems, networking, multiple programming languages, a database, Internet of Things platforms, and more.

You will learn how to use these technologies to create something that actually works.

This will be an application with many moving parts. In terms of its architecture and the specific components that it contains, it is an accurate analog of commercial Internet of Things applications in domains such as Smart Homes, Smart Cities and Manufacturing.

The application you will build is scalable to a global scale with a few targeted modifications, and it is a platform on which you can build.

Chapter 2 • A walk-through the Full Stack project

Ok, so you know that this project is going to be awesome.

But what exactly is it that you are going to do?

What exactly is it that you are going to learn?

I'm glad you asked. Let's dive in.

The image below depicts the path you will follow through this course (Figure 2.1).

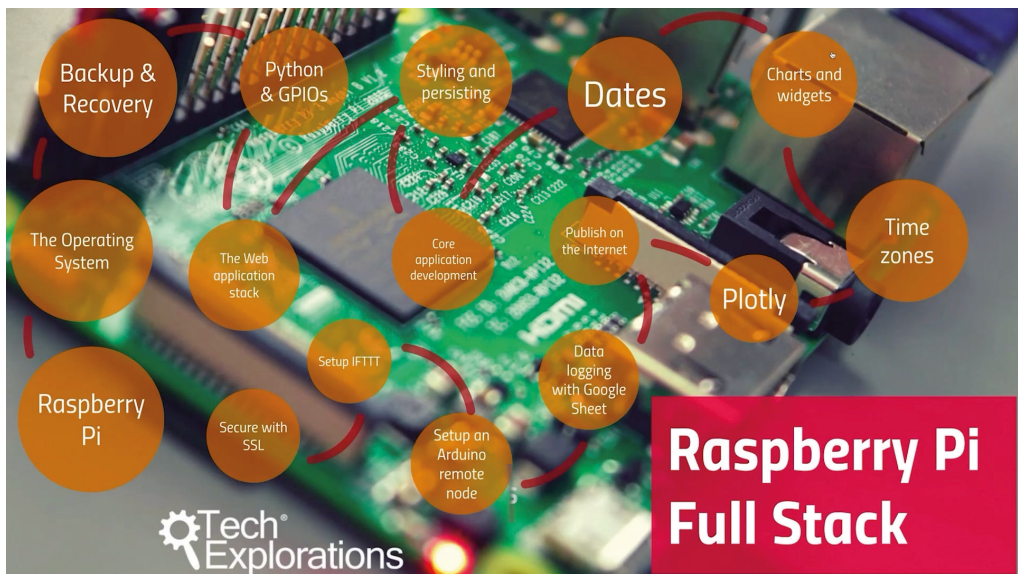


Figure 2.1: The path you will follow through this course.

The journey begins with a look at the base hardware, the Raspberry Pi. The application that you will eventually create works well on any Raspberry Pi, however in this book I have concentrated on the Raspberry Pi Zero W (for "Wireless"). The Raspberry Pi Zero represents an excellent balance of price versus performance versus size. I am amazed by how much you can achieve with this low-cost computer. In the video version of this book, I have implemented the exact same application on the Raspberry Pi 4. The implementation of the application only has a couple of small differences between the Raspberry Pi 4 and the Raspberry Pi Zero, which I will highlight when the time comes.

In the first section of the book, I will introduce you to the Raspberry Pi, and focus on the Raspberry Pi Zero as our base hardware platform. This base hardware represents the lowest level of the Full Stack application.

In the second section of the book, step two of the project, you will learn about the Raspbian operating system. This is the software that makes it possible for us, the application developers, to implement our application using high-level tools like the Python programming language and the SQLite3 database. You will learn that the Raspberry Pi, as a true multi-purpose computer, can work with a wide range of operating systems, including various flavours of Linux and even Microsoft Windows. In this project, you will be using a minimal version of the Raspberry Pi preferred operating system, Raspbian.

Because it is only a matter of "when", and not "if" for disaster to strike, **in section three of this book I will show you how to make backups of your Raspberry Pi's SD card, and then how to restore from a backup file.** Making a mistake during the implementation of a project which makes it hard or even impossible to continue is something that you should expect when you are learning something new. I have been there many times. If you are prepared for such occurrences, with full backups taken at regular intervals, you will be always be able to restore your project to a previous working state and continue from there, instead of having to rebuild from scratch.

The real work begins in section four. There, you will learn about the Raspberry Pi's General Purpose Inputs and Outputs (GPIO), and how to use them with simple Python programs. I will explain how to refer to a specific GPIO and show you the basic Python commands you need to perform simple tasks, like turn an LED on and read the state of a button. All big things start with the first simple step, and for this project, the first step is to blink an LED.

In section five, you will learn about the application stack and the components and services that are present in each level. This is where you will create the framework of this project, and learn about where in the stack various back-end and front-end technologies are placed such as the Flask micro-framework for building Python web applications, the uWSGI application server, the Nginx high-performance web server, the SQLite3 database server, HTML/CSS/JavaScript/JQuery, and, of course, Internet of Things platforms like Google's various Cloud APIs, IFTTT and Plotly. Yes, you will learn how to use all of these technologies but not in one big brain dump. We'll spread the learning throughout the full project, so you can relax.

In section six, you will start building the front end of your application, and give it persistence. Persistence is the ability of an application to store data, and retrieve it later. This is where you will set up the first version of a simple Flask Python-powered web application, and give it a basic (but elegant) user interface using HTML and CSS. For persistence, you will use SQLite3, a simple yet powerful open-source database server. From section seven and onward, you are really getting into the nitty-gritty of the application development process.

In section seven, you will create the core of your application, and flesh-out that most important functions: you will implement the sensor integration so you can get readings and store them in the database, and automate measurements so that data-logging begins.

In section eight, you will learn how to work with dates in the database, retrieve a subset of sensor records from the database using the URL query string, and how to integrate graphical widgets in your application's front end to make it easy for the end user to select date and time ranges.

You are now into the most rewarding and fast-paced part of the project, where with each chapter you implement new features.

In section nine, you will learn how to create charts from the stored data, and display multiple datapoint across selectable date and time ranges graphically.

In section ten, you will revisit the date and time topic that you started in Section eight, but this time at the front-end. You will learn how to use JavaScript/JQuery and Python to convert the date and times stored in your application's database into the correct date/time for your timezone. Of course, once you complete the implementation of this feature, the conversion will happen automatically, without any input from the user.

In section eleven, you will revisit one more topic, the graphical representation of data, and learn how to create temperature and humidity charts using Plotly. This is an opportunity to learn how to interact with a powerful Cloud service, and increase the value of your application by giving the user the opportunity to process their environment data with Plotly's powerful graphical analysis tools.

In section twelve, you will learn how to access your Raspberry Pi Full Stack application from anywhere in the world. By doing so, you will be able to check your application on your phone no matter where you might happen to be in the world. Most people would stop here and call it a day, but not us. There's more to do.

In section thirteen, you will learn to record data from your sensors on a Google Sheet using the Google Drive and Sheet APIs. This work will give you a unique insight into how you can integrate a sophisticated Cloud service, like Google Drive and Sheet, including the confusing processes around creating and using its authentication and encryption certificates.

In section fourteen you will learn how to expand your Raspberry Pi Full Stack application by connecting (wirelessly) Arduino nodes with their own sensor. With this extension, your Raspberry Pi will be recording data from an arbitrary number of Arduino nodes (as well as from its own sensor), both on its local database, and on Google Sheet.

With the newly expanded application from section fourteen, you will jump into section fifteen and learn how to use another popular Cloud service, If This Then That (IFTTT). You will learn how to implement a configurable email alert app that will email you when the temperature or humidity in one of your application nodes exceeds a threshold. Once you know how to do this, you will be able to integrate any IFTTT capability with your application. It is incredible what you can achieve in just two steps (step 1: "if this", step 2: "then that").

To conclude this project, in Section sixteen, you will learn how to secure your application with SSL. This is important especially if you have chosen to expose your application to the Internet, as you learned in Section twelve. Once you implement the security components and configuration on the web server, you will be able to access your application page with the secure HTTPS protocol instead of the un-encrypted HTTP.

There you have it, this is the project you are about to embark on.

Ready?

Chapter 3 • Required hardware

This project is hands-on. While you will be spending most of your implementation time on the software side, you will need to assemble the hardware first. Luckily, the hardware requirements are few and simple.

Below, you will find the list of hardware components you will need. Please use this list as a quick reference, but check out the online list of parts to ensure that you have the latest details about the hardware.

You can find the online parts list here: <https://techexplorations.com/parts/rpifs-parts/>

A question I frequently receive from students of Raspberry Pi Full Stack is "which Raspberry Pi" and "which Arduino" should I use?

Here are my guidelines:

Which Raspberry Pi?

It doesn't really matter. I tested this project on the original Raspberry Pi 1 Model A. In fact, my Raspberry Pi 1 is still running the original code for this project, going strong since 2014.

I also tested with the Raspberry Pi 2, 3 and 4 (all, Model B), as well as the Raspberry Pi Zero W. In all cases, the application works flawlessly.

Your choice of Raspberry Pi for this project comes down to these considerations:

- Do you have a spare Raspberry Pi?
- Do you mind waiting a little longer for the software compilation and installation to complete?
- Do you prefer not to use the more recent and expensive Raspberry Pi's (especially the Raspberry Pi 4 with 4GB RAM).
- Do you prefer a smaller footprint?

Once you have completed the application and it is working on your Raspberry Pi, it will be hard to "feel" the performance difference between the various Raspberry Pis, with the exception of the Raspberry Pi 1. The Raspberry Pi 1 is and feels slow to the end user, so I would recommend against it.

So, let's exclude the Raspberry Pi 1 from the remainder of this discussion.

The application is currently running on a Raspberry Pi 4, 3 and Zero and I can hardly feel the difference.

Next, consider the performance of each computer during development. Installing and compiling C code is computationally intensive. To compile the Python interpreter and RF24 drivers you will need anywhere between 30 minutes to one hour.

This is where the Raspberry Pi 4 beats all other Raspberry Pi's by a long way.

It is fast, and this means you will not have time to get coffee or tea during compilation. The Raspberry Pi Zero is the slowest of the modern Raspberry Pi's because of its low clock speed and single core. Still, I didn't find the performance unworkable. I just went to get coffee or worked on my email queue while the Raspberry Pi Zero was "sweating" with the compilation.

If you are the kind of person that goes for top raw performance, go for the Raspberry Pi 4. It has so much power that you can easily run the Full Stack application using the full GUI version of the operating system, as well as run other applications on it, like the Workbench Automation Computer application, and more.

The Raspberry Pi 4 is super-fast, but it also quickly gets super-hot. This heat tends to effect the temperature sensor which consistently shows a higher temperature on the Raspberry Pi 4, skewing your temperature data by one or two degrees Celsius.

If you are concerned by this, and want to ensure more accurate measurements, you should avoid the Raspberry Pi 4, or just ignore the data from its on-board sensor and instead use data from Arduino nodes.

If you are not thrilled by the expensive Raspberry Pi 4 and the amount of heat it produces, consider the Raspberry Pi Zero W (Wireless). It is not crazy fast, but is very cheap, fast enough during development, and has a tiny footprint. You can use it with a regular USB power supply (instead of the more expensive and specially-designed power supply for the Raspberry Pi 4).

The Raspberry Pi Zero W is my recommendation.

As you can see in Figure 3.2, the Raspberry Pi Zero W, with the custom project HAT that contains the sensor, button, LEDs and RF24 transceiver measures around 7 cm in length, 4 cm in height and 5 cm in width.

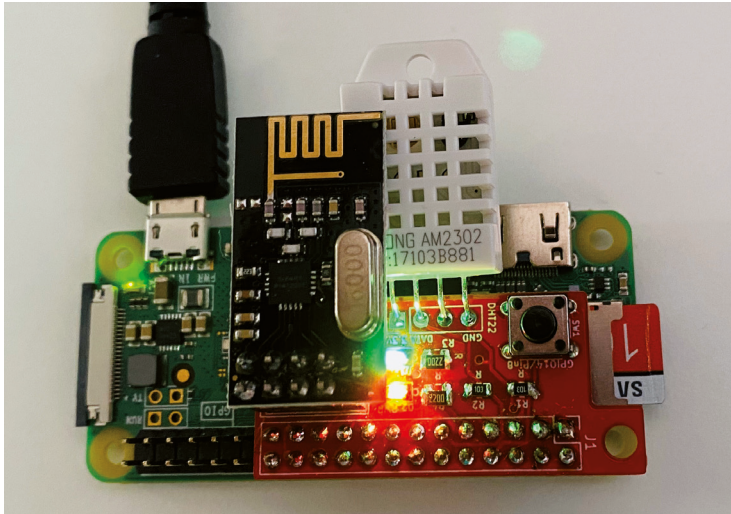


Figure 3.2: My Raspberry Pi Zero W with my custom project HAT.

It emits minimal heat and costs less than \$20. Unless you have specific reasons to use a different model, go for the Zero.

Which Arduino?

For the Arduino, the guidelines are much simpler. Just use the Arduino Uno or something compatible, or whichever Atmega328-based Arduino board you happen to have.

I have tested the circuit and sketch with an Arduino Uno and Arduino Pro Mini boards, and the operation in both cases was flawless.

List of hardware

For a more complete and up-to-date list of hardware components, please check <https://techexplorations.com/parts/rpifs-parts/>. The following is a simplified list that is centered around the Raspberry Pi Zero W.

- A Raspberry Pi Zero W
- An Arduino Uno or 100% compatible
- SanDisk Ultra 8GB (or larger, I use 16GBBytes) Class 10 UHS-I MicroSDHC Memory Card with Adapter
- 2 x 10K Ω resistors (a pull-up for the button, and a pull-up for the sensor Data pin)
- 2 x 330 Ω resistors (for the LEDs)
- 2 x LED Diodes (choose your preferred colours)
- 2 x DHT22 temperature and humidity sensors
- 2 x NRF24L01+ 2.4GHz Wireless RF Transceiver
- A breadboard-friendly momentary button
- Mini breadboard
- Jumper wires

The HAT PCB

I designed a tiny HAT that I invite you to use in your project. You can see a photograph this HAT in Figure 3.2.

I found using this Raspberry Pi HAT instead of the breadboard made my work with the application faster. I did not have to worry about loose jumper wires, and it also looks great. You can download a copy of its Gerber files [here](https://www.pcbway.com/project/shareproject/Raspberry_Pi_Full_Stack_RF24_and_DHT22_HAT.html), and order your boards from your preferred manufacturer.

Alternatively, you can go to this URL and order your PCBs from my shared project:

https://www.pcbway.com/project/shareproject/Raspberry_Pi_Full_Stack_RF24_and_DHT22_HAT.html

If you choose to work with the custom PCB instead of the breadboard, you will not need to source the through-hole components from the list, numbered 4 to 11. Instead consult the components list in my [shared project](#) for the SMD components you will need to solder on the PCB.

Chapter 4 • How to get help

At Tech Explorations, we support our students through our community spaces.

Raspberry Pi Full Stack has its own space, which you can reach at <https://community.techexplorations.com/c/raspberry-pi-full-stack>.

If you purchased this book directly from the Tech Explorations website, you should have already received an invitation to join our community. Please look for it in your inbox (also check your spam folder). In this email, you will find a link to accept my invitation. Click on the link to accept the invitation and fill in the form to create your free community account. If you purchased this book from one of our partners, you can still join our community, but you will need to complete these steps:

Go to our support page at <https://txplo.re/support>.

Click on the support option "Join the Makers Club".

In the field "Where did you purchase your course", choose the most appropriate option from the list.

In the text box "Comments", indicate that you wish to become a member of the Raspberry Pi Full Stack community space. Important: please copy your book order details.

In the field "Email configuration to", include the email address where you would like us to send you the invitation.

That's it. Please allow us up to 24 hour to process your request.

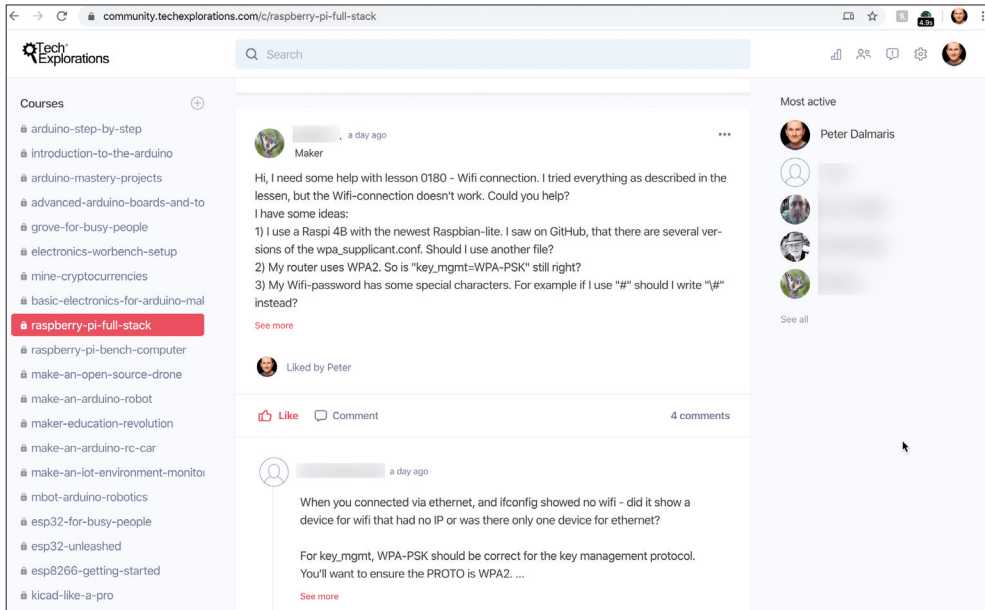


Figure 4.3: The Community Space for the Raspberry Pi Full Stack project.

Chapter 5 • The code repository

You can find the source code for this project in the course code repository on Github.

Bookmark this URL: https://github.com/futureshocked/RaspberryPiFullStack_Raspbian

The repository contains the source code in various stages of development. You can copy this code into your project instead of typing it. This will save you time and frustration that is inevitable with typing.

Apart from Python code, the repository also contains my command line sessions and various configuration files. I have tested each and every one of these files and I am confident that they work.

When you run into a problem, check the source code at Github so that you can trust that it works, and use it to try and identify the problem in your code.

Please note that the code in the repository is "alive". This means that I frequently update it to fix bugs or improve functionality. It is likely that you will find differences between the code I have captured in this book, and the "live" code in the repository.

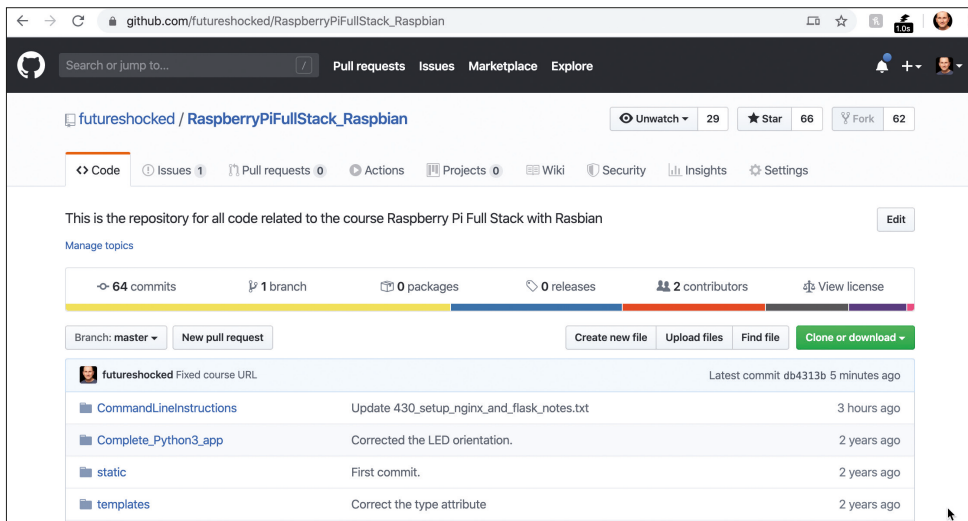


Figure 5.4: The Raspberry Pi Full Stack project code repository on Github.

Part 2: Raspberry Pi, Arduino, and Raspberry Pi Zero W

Chapter 6 • Raspberry Pi vs Arduino high level comparison

For many of us, the Arduino is the technology that introduced us into the world of programmable electronics. It is relatively simple to learn, forgiving of many common beginner mistakes, with a whole universe of documentation, example code, circuits, and projects.

We start exploring other technologies once we feel that we have grasped at least the basics of the Arduino. For many of us, the next logical step is the Raspberry Pi. The Raspberry Pi opens up new possibilities, especially when it comes to interfacing our projects with advanced online or computationally intensive features.

A question I regularly receive is about the differences and similarities between the Raspberry Pi and the Arduino. I think it is a very good question because once you understand the differences and similarities you will be able to devise ways for the two to work together, in a simple project.

The Raspberry Pi and Arduino, or, to be more precise, the Arduino Uno, are complementary technologies. While they share some common characteristics, they are different in almost every aspect that matters. And yet, because of their differences, they are each other's perfect complement.

In this chapter, I present an high-level comparison between the Raspberry Pi and the Arduino, based on the table in Figure 6.5.

Raspberry Pi vs Arduino	
Raspberry Pi	Arduino
Microcomputer	Microcontroller
Needs an operating system	Does not need an operating system
Complicated	Simple
Video out, Camera, Ethernet ports, Wifi, Bluetooth, USB, I2C, SPI, UART etc. on board	USB only for power and serial in/out, I2C, SPI, UART
Best for general computer	Best for small tasks that constantly repeat
Capable of performing a huge range of tasks	Optimised for sensing and controlling the world around it
Best for more advanced makers	Best for beginners
Programmed in many languages, including C/C++, Python, Ruby	Programmed in C/C++
Relatively high power consumption	Relatively low power consumption


Raspberry Pi Full Stack 

Figure 6.5: A high-level comparison between the Raspberry Pi and the Arduino Uno.

So, here are the 10 most important differences between the Raspberry Pi and Arduino.

Microcomputer vs Microcontroller

The Raspberry Pi is a regular computer, similar to the one on your desk. It just comes in a small package. The Raspberry Pi has all the usual features and capabilities that you expect a computer to have.

For example:

- You can connect a large, high-definition monitor, capable of playing 1080p video.
- You can connect a keyboard, a mouse, and external hard disks.
- It can connect to the Internet via Ethernet and WiFi.
- It requires an operating system, and can run your choice of Linux or Windows.
- For a tiny computer, it has respectable specifications, including a 64-bit quad core CPU with hardware video acceleration and 4GB RAM.
- It has multiple USB connectors for peripherals.

For the price and size of the Raspberry Pi, these specifications are impressive.

On the other hand, the Arduino is a totally different machine. The most important difference between the Arduino and the Raspberry Pi, is that the latter is not a computer. It is a microcontroller.

Think of a microcontroller as a very small computer without any of the things that we expect from a desktop or a laptop computer.

Take the RAM, for example, and focus on the Arduino Uno. The Arduino Uno has tiny amount of RAM, just 2 kilobytes, and can't to connect to a network like the Internet for example, without additional components. Compare this to the 4 gigabytes of RAM that are available on the Raspberry Pi 4!

The Arduino Uno only has 32 kilobytes of flash memory for storing your program, whereas the Raspberry Pi can use very large amounts of storage space on removable SD cards (and you can also connect terabytes of external disk space via USB).

Another big difference is performance. The Arduino Uno is powered by an 8-bit, 16Mhz microcontroller that contains a single-core CPU. The Raspberry Pi is powered by a 64-bit quad core ARM CPU, clocked at 1.5GHz. The comparison is far from fair. The Raspberry Pi can do real-time video processing on applications like object recognition and artificial intelligence, while the Arduino Uno would be stretched to it limits trying to identify a loud noise.

The Arduino Uno does not use an operating system. Instead, your programs are compiled to run directly on the hardware, and that is why their compiled version is called "firmware".

As you can see, the two devices that are totally different. That's because a microcontroller is designed for doing tasks that are small, constantly and repeatedly. A microcomputer is designed as a general-purpose device, with a very large repertoire of possible applications.

For example, if you want a gadget that measures the temperature and humidity in a room, and then depending on those single two values to turn on a fan or an air conditioner, then something like the Arduino or a microcontroller in general would be the best technology on which you can build such a gadget.

If you want something that can recognize a person's face from a video stream and then sent out an alert or turn on an alarm and log that event in an online database, play a movie on Netflix, and check your email, then you will need a Raspberry Pi.

The Raspberry Pi is great for general computer tasks, whereas the Arduino is great for things that are small in complexity, small in programming size, and repeat constantly.

A great aspect of the Arduino is all of its input and output pins and ability to connect them to actuators and sensors, so things such as relays, switches and motors, then the Arduino is really a good choice, really optimized for sensing and controlling the world around it.

Just like the Arduino, the Raspberry Pi does have a bunch of general-purpose input output pins that you can connect to actuators and sensors, just like the Arduino.

So there is definitely a good case to be put forward that says that the Raspberry Pi can actually do everything that the Arduino can, and therefore you don't really need an Arduino. But there is a flip side to that: complexity and robustness.

The great thing about the Arduino is that it's simple and forgiving.

The fact that there is no operating system to worry about means that you don't have to learn how to use one just to turn on a relay.

For beginners, I always say that the best way to get started with electronics and with control applications is through the Arduino, gain an understanding of its simple hardware, and build-up skill using the bare minimal infrastructure that a microcontroller provides before moving onto something more complicated.

When a new person comes into the world of electronics and control, their objectives tend to be fairly simple; just blinking an LCD very often is a great start. A simple start like that is a significant achievement that can lead to much greater things.

You can achieve wins like this very quickly with the Arduino.

Programming

Another big difference between the Arduino and the Raspberry Pi is programming.

The Arduino is programmed in C++. Normally, C++ is not a language that I would recommend to a beginner. It is fairly complicated, and many concepts and techniques that are not intuitive. A language like Python or Ruby is much more suitable as a first programming language. However, the Arduino did not become so popular despite the difficulty in

learning C++.

The people that created Arduino were keenly aware of this, so they created a collection of libraries that hide most of the complexity of C++. It makes programming so simple that learners don't realise they are using C++ until well into their learning journey. We often refer to these libraries as the "Arduino language", even though we are really using the hard-core C++ that programmers use to create the Linux operating system.

In addition to the language, the Arduino developer team created a very simple development environment that hides all of the complexity of "regular" C++ development environments and compilers that, again, can confuse even those of us with many years experience of using them.

In summary, the Arduino is an excellent choice for beginner because it is based on relatively simple hardware, and has a very simple programming interface and environment. On the other hand the Raspberry Pi, being a full microcomputer, has a wide range of languages that you can use to program it.

Any language that you can use in Linux is also available for Raspberry Pi.

By default it comes with support for C and C++ and Python. You can also install other languages, like Ruby, PHP, JavaScript, Lua, etc. Virtually any programming language is available on the Raspberry Pi. If you're proficient in any language, chances are that you'll be able to use it with Raspberry Pi.

Keep in mind that most of the Raspberry Pi documentation is written for Python. Python is, by many accounts, the most widely used high level language in the world.

Compared to C++, it has a very short learning curve. Even if you've never used Python, you'll be able to pick it up fairly quickly.

Hardware

Hardware-wise, the Raspberry Pi is more complicated than the Arduino Uno because of the amount of peripheral hardware that is present on its circuit board.

The Raspberry Pi 4 comes with one or two HDMI video output ports, a camera port (so you can do things like face or object recognition), a 1 gigabit Ethernet port, Wi-Fi and Bluetooth, and multiple USB ports (so you can connect your keyboards, mice, external video cameras, and anything else that pretty much has a compatible driver for the Raspberry Pi running linux).

The Raspberry Pi is unique among computers because it exposes general-purpose input/output pins that you can use to interface with other external modules using communications protocols such as I²C, SPI, and UART serial. Sensors motor controllers, LCD displays, relays, and many other devices can be connected directly to the Raspberry Pi via GPIOs.

In contrast, the Arduino Uno looks like it's lacking in almost every respect. It doesn't have a USB host suitable for things like a keyboard or external storage. Its single USB port allows you to connect your Arduino to your computer for power and serial communication only. However, the Arduino Uno does contain a small number of GPIO pins, that you can use to connect sensor, actuators, displays and other peripherals using the I²C, SPI and UART protocols.

Power

Finally, I'd like to mention a few things about power consumption.

As a computer with a lot more resources and hardware to power, the Raspberry Pi consumes much more current compared than the Arduino. The Raspberry Pi consumes around 2.8 W of power when it's idle and not doing anything useful. It requires more than that if you have connected things such as touch screens and cameras. It requires a power supply with at least 3A at 5V.

The Arduino, on the other hand, has very low power consumption needs. This makes it suitable for low-power, battery applications. It can operate with as little as 20 milli amps, for example, versus 700 milli amps for the Raspberry Pi. For a microcontroller, 20 milli amps is not negligible, but with the power management features built into the chip, it can be reduced significantly.

What is power management? It is one of the advantages of microcontrollers over regular computers. Microcontrollers are built to be able to programmatically control the power consumption of their various components when not needed in order to reduce power draw. It is possible to program a custom version of an Arduino to operate for many months on a small alkaline battery by turning off its subsystems and "waking" them up when needed. When the job is done, the Arduino can go back to sleep and conserve power.

The ability to programmatically control the power consumption of the microcontroller is a very important advantage of the Arduino and is particularly useful when you want to build a gadget that needs to be deployed out in the field, away from power sockets.

Index

A

Application framework 117
Application server 17, 115
Arduino 27, 330
Arduino node 333
Arduino Uno 28
ARM 29
Arrow 248
auto-start 148

B

Backup 17, 43, 64
Bash 391
Boot 45, 53
Breadboard 22, 107
Button 17, 22, 100, 102, 104

C

C++ 30, 40
charting 250, 309
Cloud 119
code repository 26
Command Line Interpreter 76
community space 24
Compilation 21, 34, 122
Compiler 31, 40
Configuration 19, 59, 140, 143, 300
CPU 29, 33
Cron 194
Crontab 194
CSS 159, 164

D

data logging 308, 328
Database 151, 189
Datetimepicker 236
Debugging 176, 281
DHCP 56, 287
DHT22 22, 107, 111
Diode 22
Diskutil 65
dynamic asset 159

E

Efficiency 33
env_log.py 191
Etcher 46, 71

F

Firmware 29
Flask 133, 136, 172, 176
Full Stack 16

G

Git 107
GitHub 26, 49
Google API credentials 311
Google Charts 224
Google Sheet 308, 322, 328
GPIO (General Purpose
Inputs and Outputs) 17, 97, 181
GPIO Zero 75, 94
GPIOZERO 94, 100
GUI 41, 116

H

HAT 21, 23
Header 33, 74
Headless 42, 48
heat sink 36
Hostname 50, 400
HTML 137, 159,
HTTP request 116, 303
HTTPS 19
Humidity 22, 107

I

IFTTT 359, 371
IP address 50, 58, 285, 287

J

JavaScript 162, 218
jQuery 218, 236
jsTimeZone-Detect 224

L

lab_app.db 155, 189, 202

-
- lab_app.py 184, 196, 213
 - lab_env_db 50, 159, 377
 - lab_env_db.html 159, 197, 229
 - lab_temp 185,
 - LCD 30, 31

 - M**
 - MacOS 48, 65, 67
 - Makers Club 24
 - Microcomputer 29
 - micro-framework 17, 117, 133

 - N**
 - network scanner 50, 55
 - Nginx 131, 136, 140, 300
 - nRF24 wiring 340

 - O**
 - Olson Timezone 245
 - operating system 17, 38

 - P**
 - PCB 23, 333
 - Persistence 17
 - Pinout 75
 - Plotly 259
 - port forwarding 285
 - power supply 32
 - Power 32, 43, 53
 - Programming 30
 - Public IP address 285
 - Python 3 76, 125
 - Python functions 80
 - Python interpreter 20, 34, 88, 120
 - Python interpreter 34, 88, 120
 - Python Package Manager 97, 120, 134
 - Python Virtual Environment 119, 129
 - Python 73, 85, 191, 221

 - Q**
 - querystring 210

 - R**
 - RAM 29, 36
 - Raspberry Pi 4 36
 - Raspberry Pi Zero W 27, 33
 - Raspbian Lite 41, 116
 - Raspbian 45, 53
 - Resistor 22, 91
 - Restore 17, 64
 - RF24 20, 331
 - RF24Network 343
 - Root 52, 61, 129
 - Rootfs 51
 - Router 50, 53
 - rpi-gpio 181

 - S**
 - SD card 34, 64
 - self-signed certificate 295
 - Sensor 107, 179, 330
 - Service Port number 285
 - SFTP 168, 325
 - Skeleton 158, 164
 - SQLite3 152, 180
 - SSH 42, 48, 168, 395,
 - SSL private key 296
 - SSL public certificate 296
 - SSL 300, 405
 - static asset 159
 - static IP address 287
 - Sudo 59, 61
 - system Python 120, 124
 - systemd 148, 374

 - T**
 - Tech Explorations 24
 - Temperature 21, 107, 111
 - Templates 117, 172
 - Timezone 243
 - Transceiver 22, 331
 - Twilio 380

 - U**
 - URL 18, 210
 - uWSGI 115, 138

 - V**
 - Vim 85
 - Virtual Machine 119

W

Web Application Stack	114
Web server	17, 117, 281
Webhook	362
WiFi	33, 48
Win32 Disk Imager	69
Windows	17, 36, 69, 71,
Workbench Automation Computer	21, 37

A comprehensive course that will teach you how to build a modern IoT application

Raspberry Pi Full Stack

This book will take you on a whirlwind tour of full-stack web application development using Raspberry Pi. You will learn how to build an application from the ground up.

You will gain experience and know-how of technologies including:

- The Linux operating system and command line.
- The Python programming language.
- The Raspberry Pi General Purpose Input Output pins (GPIOs).
- The Nginx web server.
- Flask Python web application microframework.
- JQuery and CSS for creating user interfaces.
- Dealing with time zones.
- Creating charts with Plotly and Google Charts.
- Data logging with Google Sheet.
- Developing applets with IFTTT.
- Securing your application with SSL.
- Receiving SMS notifications to your phone using Twilio.

This book will also teach you how to set up a remote wireless Arduino sensor node and collect data from it. Your Raspberry Pi web application will be able to process Arduino node data in the same way it processes data from its onboard sensor.

Raspberry Pi Full Stack will teach you many skills essential to building Web and Internet of Things applications.

The application you will build in this project is a platform that you can extend upon. This is just the start of what you can do with a Raspberry Pi and the software and hardware components that you will learn about.

This book is supported by the author via a dedicated discussion space.



Dr. Peter Dalmaris is an educator, an electrical engineer and Maker. Creator of online video courses on DIY electronics and author of several technical books. As a Chief Tech Explorer since 2013 at Tech Explorations, the company he founded in Sydney, Australia, Peter's mission is to explore technology and help educate the world.