

# Fhir On Pi – English version



## Introduction

Trying to succeed in things that are too hard for you can be fun sometimes. This is how my project “Fhir On Pi” began. I’m experimenting quite some years with Arduino and Raspberry boards and consider myself being an “experienced amateur” in this area. Being “business architect” in healthcare-projects got me acquainted with the main concepts of “FHIR”, and I was challenged by the idea to combine the Raspberry Pi with a FHIR-server.

Spoiler-alert: I finally succeeded, although it took much more time than I expected at the start.

I decided to document the steps taken, hoping that more people would like to have a “FHIR-server” scaled at the size of a smartphone.

I published my adventures in the Dutch language on LinkedIn. Looking at the people who read these seven issues of “Fhir On Pi” I saw

Location	10284
ExplanationOfBenefit	31173
Condition	6079
MedicationStatement	3453
DocumentReference	4093
MessageHeader	4765
Coverage	6158
Appointment	2112
ClaimResponse	3083
Procedure	1885
AuditEvent	1203
Practitioner	12478
MedicationRequest	13072
Organization	5265
Binary	4450
PractitionerRole	7505
Medication	5911
Device	5022
LXI	1005
Provenance	4405
CarePlan	2718

<Hapi /> HAPI-FHIR  
fhir made simple.

You are accessing the public FHIR server **UHN\_HAPI Server (R4 FHIR)**. This server is hosted elsewhere on the internet but is being accessed using the HAPI client implementation.

⚠️ This is not a production server! Do not store any information here that contains personal health information or any other confidential information. This server will be regularly purged and recreated with test data.

Server	UHN Test Server (R4 Resources)
Software	HAPI FHIR Server - 5.3.0-SNAPSHOT/5584548ee02020-12-03
FHIR Base	<a href="http://hapi.fhir.org/baseR4">http://hapi.fhir.org/baseR4</a>

Server Actions

Retrieve the server's **conformance statement**

Retrieve the update **history** across all resource types on the server.

Post a bundle containing multiple resources to the server and store all resources within a single atomic transaction.

Bundle \* (place transaction bundle body here)

quite a number of interested persons from outside the Netherlands. So I promised to put up a (condensed) English version of this story, a promise I hereby fulfilled.

I will not to elaborate on every mistake I made, but when you follow these descriptions you'll end up with a working FHIR-server, running on a Raspberry Pi, that's certain.

## The start

For this project I'm using a Raspberry Pi 4 model B with 8Gb memory on board. There are a lot of places where you can buy (or post-order) this "single board computer". The Pi runs the Raspbian operating system. You'll need some experience with Unix/Raspbian because I'm not going to explain every command.

Since the Pi will be working hard, especially during the "build" of Java-applications I do not advice to use a smaller Raspberry (for example a Raspberry Pi 3B, or a Pi 4 with less memory). I don't say it can't be done though, I just didn't try.

Firstly I used Raspbian Buster, "full" version, but because my SD-card filled up completely I turned to Raspbian Buster Lite. Raspbian Lite on a 16Gb SD-card worked smoothly. So the advised configuration is:

- Raspberry Pi 4 Model B, minimal 8 Gb internal memory
- Raspbian "Buster" Lite on a 16Gb SD card

You'll need an Internet-connection from your Raspberry to download all packages, a power supply for the Pi, an HDMI monitor, USB keyboard and mouse. I also use a (Windows) PC because it is easy to cut-and-paste between documentation (like this PDF) and the "terminal screen" in for example Putty. Apart from that, several described steps require a browser on a PC (Chrome, Opera, Edge...) in the same network to see results.

I bought a simple casing for the Pi to protect it against falling screwdrivers, pencils etc.



## Configuration

After connecting all the peripherals with the Pi you'll have to load Raspbian onto it. There are many good descriptions of this process so I'm not going to repeat those steps.

Having Raspbian installed you'll go ahead configuring the Raspberry using raspi-config. (**sudo raspi-config**):

1. Change the standard password for the user “pi”;
2. Connect the Raspberry with the network, either using a cable or Wifi;
3. Enable SSH. We’re going to use this protocol to connect to the Pi from a “normal” PC;
4. Enlarge the file system (Option 6 Advanced Options, Option A1 Expand FileSystem).

[illegible]

When all this is done you can issue the command “**sudo reboot now**”. The Raspberry restarts and after logging in you can ask the Raspberry it’s IP-address:

### Hostname -l (Capital I from India)

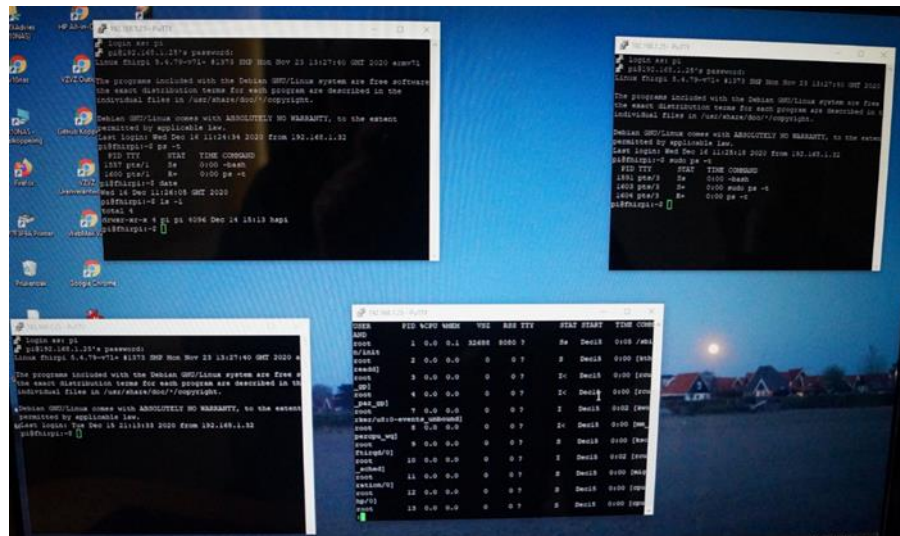
Write this IP-adress down, you'll need it very often!

## Connect a PC

I like working on my Raspberry using a PC with a “terminal” app. This way I can cut-and-paste between texts (like in this document) and the Pi and it is also easy to transfer files.

The Pi is a “real” computer, so you can have multiple terminal-sessions. For example one session to view the logfiles and another to give commands. I’m using Putty as a terminal-app.

After installing the terminal app you can connect to the Raspberry using the IP-address you just wrote down.



## APT

With the Raspbian operating system the tool “APT” was installed on your Raspberry. APT stands for “Advanced Packaging Tool” and it helps to install, update and remove packages on your Raspberry. We’re going to use APT to install the right version of Java on the Pi. To deploy the FHIR-server you’ll need the Java Runtime Engine as well as the Java Development Kit. Raspbian Lite doesn’t come with Java, as you can see with:

**java -version** (two minus-signs before “version”)

You’ll see “command not found” : your Raspberry doesn’t know Java yet.

1. Firstly we make sure that APT knows about all recent versions of the packages:

**sudo apt update**

2. After this we’re installing the most recent Java production version:

**sudo apt install default-jdk**

3. Finally we’re going to check what the Raspberry has learned:

**java -version** (two minus-signs before “version”)

Your Raspberry will answer with something like:

```
openjdk 11.0.9.1 2020-11-04
```

```
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Raspbian-1deb10u2)
```

```
OpenJDK Server VM (build 11.0.9.1+1-post-Raspbian-1deb10u2, mixed mode)
```

Operation succeeded and ready for the next step!

## Tomcat

(Apache) Tomcat is an HTTP webserver, built with Java. Tomcat is able to provision functionality using Internet protocols. It is possible to build web portals using Tomcat, to create applications communicating using XML messages, to expose databases to the outside world etc.

The HAPI FHIR-server we're deploying on the Raspberry can be seen as a Tomcat-"application". We are going to deploy other Tomcat-applications as well since the Tomcat management portal and the Tomcat documentation also are "Tomcat applications".

Tomcat itself consists out of:

- **Catalina** (a servlet container. A servlet handles network-requests, like a http-request. The servlet container takes care of a number of generic tasks);
- **Coyote** (a HTTP-connector)
- **Jasper** (a JSP-engine, a technical component, serving Java Server Pages).

Let's install Tomcat:

1. We're creating a new user "tomcat"

```
sudo adduser tomcat
```

The "adduser"-script creates an environment for the user "tomcat" and asks for a password with this account.

**New password:** <remember this password!>

**Retype new password:** < enter the same password>

The script asks several other things (like given name, family name and room number) but you don't have to answer here, just give < enter> .

2. Create a new /tmp subdirectory in /home/pi

**cd /home/pi**

**mkdir tmp**

3. In this /tmp directory we put the Tomcat "tarball" (zip-file):

**cd /tmp**

and next:

**curl -O <https://apache.newfountain.nl/tomcat/tomcat-9/v9.0.41/bin/apache-tomcat-9.0.41.tar.gz>**

(This command should be given on one line)

4. Now we're creating a directory for the tomcat-installation. This will be a subdirectory of /opt

**sudo mkdir /opt/tomcat**

5. Unpack the tomcat-"tarball" in the new directory

**sudo tar xzvf apache-tomcat-\*tar.gz -C /opt/tomcat --strip-components=1**

6. Adjust the file protections in the directories:

**cd /opt/tomcat**

and after this:

**sudo chgrp -R tomcat /opt/tomcat**

and next:

**sudo chmod -R g+r conf**

and:

**sudo chmod g+x conf**

and finally:

**sudo chown -R tomcat webapps/ work/ temp/ logs/**

## Nano, the editor

We're going to make sure that Tomcat can be started. This means editing a file on the Pi and the tool we're going to use is "nano". But before editing we need to know where Java is installed. This can be seen with:

**sudo update-java-alternatives -l** (de last character is a lowercase L).

Your Pi answers with something like:

```
java-1.11.0-openjdk-amd64 1081 /usr/lib/jvm/java-1.11.0-openjdk-amd64
```

JAVA\_HOME is the piece of tekst at the end (in the example: /usr /lib/jvm/java-1.11.0-openjdk-amd64 ). If you're using Putty or another terminal-program on the PC you have an advantage since you can copy the text and put it in Notepad for the time being.

Nu we're going to edit:

**sudo nano /etc/systemd/system/tomcat.service**

After starting Nano, paste the following lines in the new file (tekst continues on next page):

**[Unit]**

**Description=Apache Tomcat Web Application Container**

**After=network.target**

**[Service]**

**Type=forking**

**Environment=JAVA\_HOME=/usr/lib/jvm/java-1.11.0-openjdk-amd64**

**Environment=CATALINA\_PID=/opt/tomcat/temp/tomcat.pid**

**Environment=CATALINA\_HOME=/opt/tomcat**

**Environment=CATALINA\_BASE=/opt/tomcat**

**Environment='CATALINA\_OPTS=-Xms512M -Xmx1024M -server -XX:+UseParallelGC'**

**Environment='JAVA\_OPTS=-Djava.awt.headless=true -  
Djava.security.egd=file:/dev/./urandom'**

**ExecStart=/opt/tomcat/bin/startup.sh**

**ExecStop=/opt/tomcat/bin/shutdown.sh**

**User=tomcat**

**Group=tomcat**

**UMask=0007**

**RestartSec=10**

**Restart=always**

**[Install]**

**WantedBy=multi-user.target**

After pasting these lines, substitute the tekst after “JAVA\_HOME=” with your own JAVA\_HOME (written down in Notepad).

Save the file with **<ctrl> o**, after this **<return>**, and **<ctrl>x** to leave Nano.

## **Systemctl**

When your Pi is started and the “kernel” is loaded, the system looks which extra applications have to be started. This will be different for each system: a mailserver needs other applications than a firewall-system. Systemctl is the tool to manage the Linux start- and stop-system and we’re going to use it now to start Tomcat.

Firstly we make sure that systemctl is aware of the actual situation:



**sudo systemctl daemon-reload**

After this we're going to start tomcat:

**sudo systemctl start tomcat**

Let's check if Tomcat is working:

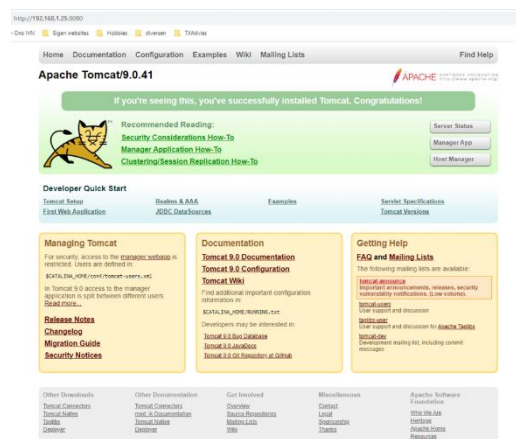
**sudo systemctl status tomcat**

```
Dec 15 11:50:35 fhirpi systemd[1]: Starting Apache Tomcat Web Application Container...
Dec 15 11:50:35 fhirpi startup.sh[694]: Existing PID file found during start.
Dec 15 11:50:35 fhirpi startup.sh[694]: Removing/clearing stale PID file.
Dec 15 11:50:35 fhirpi startup.sh[694]: Tomcat started.
Dec 15 11:50:35 fhirpi systemd[1]: Started Apache Tomcat Web Application Container.
lines 1-14/14 (END)
```

Systemctl shows the status of Tomcat in a tool called “less”. You’ll see date and time when Tomcat started. You succeeded in installing Tomcat!

Leave “less” with <q> or <ctrl>c.

Now, start a webbrowser on your PC and type in the Raspberry’s IP-address, followed by :8080 . In my case this would be <http://192.168.1.25:8080> (but your IP-address probably will differ).



You’ll see tomcat’s welcome-page on your screen! Your Pi now has become a webserver.

Now we’ve Tomcat installed, but after restarting your Pi you’ll have to enter again:

**sudo systemctl start tomcat**

because Tomcat doesn’t automatically start after boot. Because computers are better in repeating tasks than human beings we’re going to solve this first with:

**sudo systemctl enable tomcat**

Now Tomcat will start directly with a boot of the Pi, which you can verify with:

**sudo reboot now**

After a short time you can verify Tomcat started by a Putty session in which you give:

**sudo systemctl status tomcat**

or with your browser in which you give a “refresh” to

<ip-address-of-your-Raspberry>:8080 .

## Tomcat management portal

Now we’re going to install the Tomcat management portal. We’re not going to use HTTPs with certificates but still we’ve got to change some security settings in Tomcat since it is “out of the box” protected against configuration changes from outside.

What we’ve got to do is:

1. Tell Tomcat that there is a user who is allowed to change its configuration using the management portal;
2. Enable access towards the Tomcat management portal from outside (for example your PC), since normally the management portal is only accessible from the Raspberry Pi itself.

Firstly we’re going to change the file **tomcat-users.xml** . After following my earlier instructions you’ll be able to do this with

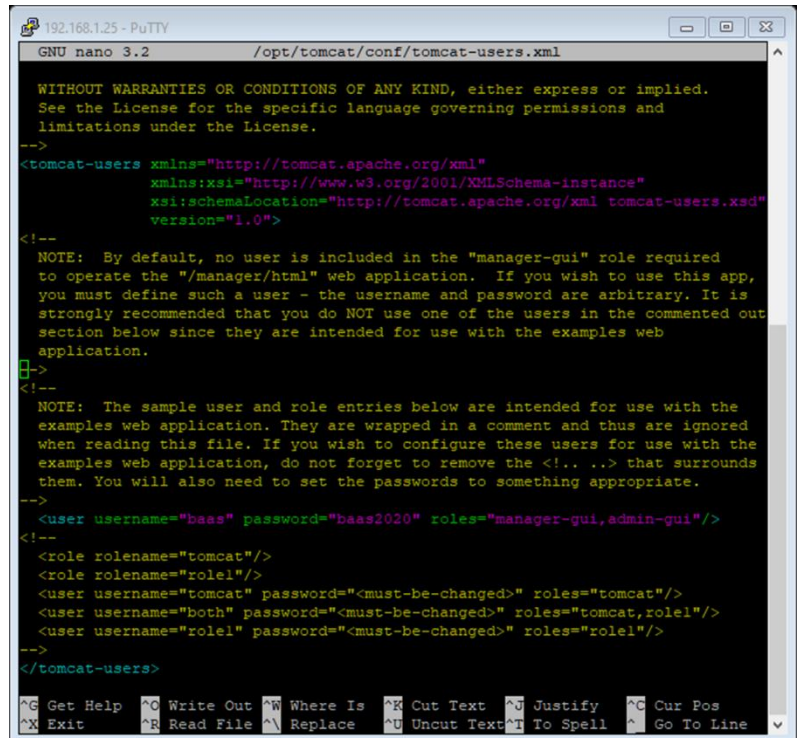
**sudo nano /opt/tomcat/conf/tomcat-users.xml**

You'll see lots of text. Most of it will be placed between `<!--` and `-->` indicating that this is "comment", not being processed by your Raspberry. (In my Putty sessions these comments are automatically coloured yellow).

Find a nice place *between* these texts (so: not between `<!--` and `-->`) and add the following line:

```
<user username="admin"
password="password"
roles="manager-gui,admin-gui"/>
```

In which "admin" and "password" are changed in a username and password of your own choice. In the screenprint you can see that I chose for the username "baas" and the password "baas2020". Please note that this username/password is not related to a "normal" user of the Raspberry Pi. They are only valid for the management portal.



```
192.168.1.25 - PuTTY
GNU nano 3.2 /opt/tomcat/conf/tomcat-users.xml

WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
-->
<tomcat-users xmlns="http://tomcat.apache.org/xml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd"
  version="1.0">
  <!--
  NOTE: By default, no user is included in the "manager-gui" role required
  to operate the "/manager/html" web application.  If you wish to use this app,
  you must define such a user - the username and password are arbitrary.  It is
  strongly recommended that you do NOT use one of the users in the commented out
  section below since they are intended for use with the examples web
  application.
  -->
  <!--
  NOTE: The sample user and role entries below are intended for use with the
  examples web application.  They are wrapped in a comment and thus are ignored
  when reading this file.  If you wish to configure these users for use with the
  examples web application, do not forget to remove the <!-- ... --> that surrounds
  them.  You will also need to set the passwords to something appropriate.
  -->
  <user username="baas" password="baas2020" roles="manager-gui,admin-gui"/>
  <!--
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <user username="tomcat" password="<must-be-changed>" roles="tomcat"/>
  <user username="both" password="<must-be-changed>" roles="tomcat,role1"/>
  <user username="role1" password="<must-be-changed>" roles="role1"/>
  -->
</tomcat-users>
```

Save the file with `<ctrl> O` and leave the Nano-editor with `<ctrl> X`.

This was the first step. Using the new username/password combination you can now use the Tomcat management portal, but only from the Pi itself. This is of no use because we're using Raspbian Lite and there is no real webbrowser in that environment. So the next step is to enable outside access to the Tomcat management portal. I say "management portal" but in fact Tomcat has two management portals:

- The tomcat manager portal
- the tomcat host manager portal

and these two portals result even in three different webpages!

At first we'll give external access to the tomcat manager app:

**`sudo nano /opt/tomcat/webapps/manager/META-INF/context.xml`**

In this file you'll see the start-tag:

```
<Context antiResourceLocking="false" privileged="true" >
```

and the end-tag:

```
</Context> .
```

Between these two tags you'll see the restrictions for portal access.

Put a `<!--` in front of these restrictions and a `-->` after the end.

By doing so you change everything between the start-tag `<Context antiResourceLocking="false" privileged="true" >` and the end-tag `</Context>` into "comment".

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Licensed to the Apache Software Foundation (ASF) under one or more
contributor license agreements.  See the NOTICE file distributed with
this work for additional information regarding copyright ownership.
The ASF licenses this file to You under the Apache License, Version 2.0
(the "License"); you may not use this file except in compliance with
the License.  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
-->
<Context antiResourceLocking="false" privileged="true" >
  <!-- <CookieProcessor className="org.apache.tomcat.util.http.Rfc6265CookieProS
        sameSiteCookies="strict" />
  <Valve className="org.apache.catalina.valves.RemoteAddrValve"
        allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" />
  <Manager sessionAttributeValueClassNameFilter="java\.lang\.(?:Boolean|IntegerS
  -->
</Context>
```

Save the file with `<ctrl> O` and leave nano with `<ctrl>X`.

Now we're going to do exactly the same for the host manager portal:

```
sudo nano /opt/tomcat/webapps/host-manager/META-INF/context.xml
```

Here too you put a `<!--` after the Context-starttag and a `-->` before the `</Context>` endtag. Use `<ctrl>o` to save the file and `<ctrl>X` to leave nano.

Now restart Tomcat:

**sudo systemctl restart tomcat**

and take a look at the management portal. From you PC go to the following address with your browser:

**http://<IP-address of your Raspberry>:8080/manager/html.**

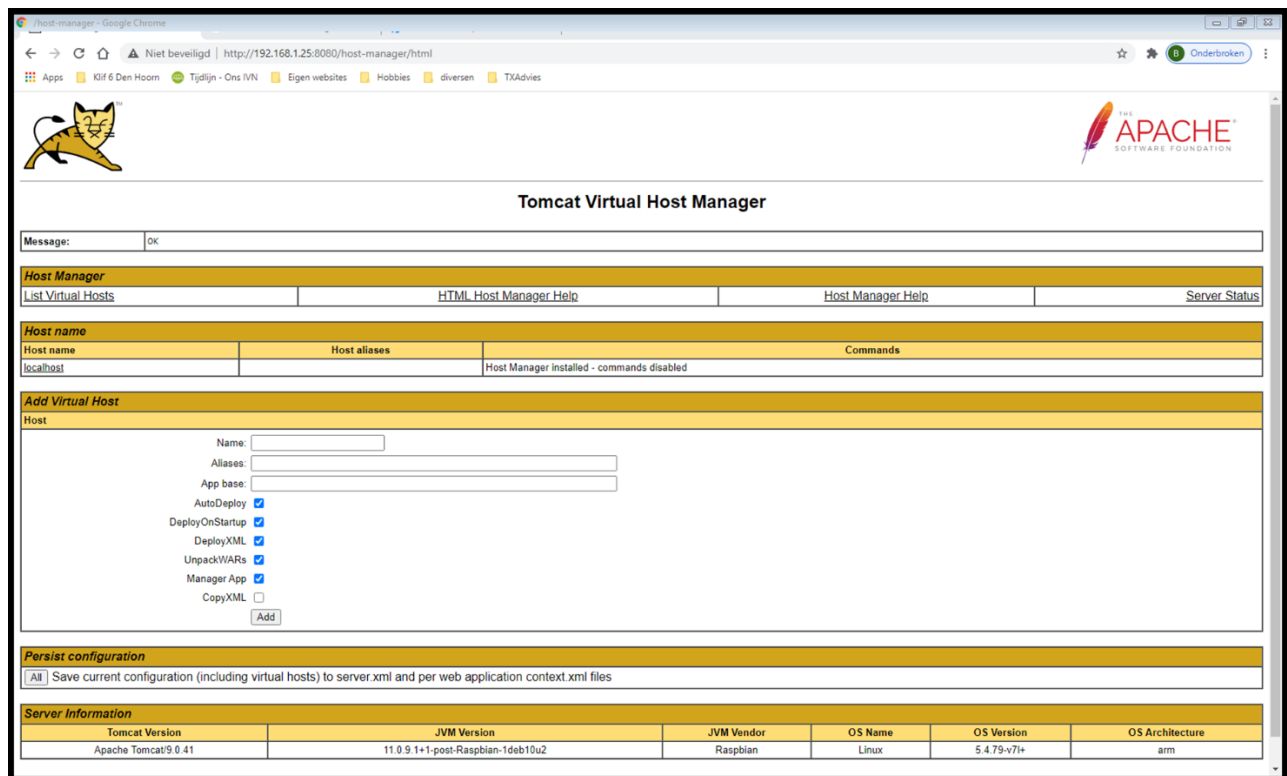
In my configuration this is: <http://192.168.1.25:8080/manager/html> . You'll have to log in, using the username and password you've put some minutes ago in tomcat-users.xml , and after that you'll see the following screen:

The screenshot shows the Tomcat Web Application Manager interface in a web browser. The browser's address bar shows the URL `http://192.168.1.25:8080/manager/html`. The page features the Apache Software Foundation logo and the title "Tomcat Web Application Manager". Below the title, there is a "Message" field showing "OK". The main content area is divided into several sections:

- Manager**: Includes links for "List Applications", "HTML Manager Help", "Manager Help", and "Server Status".
- Applications**: A table listing deployed applications with columns for Path, Version, Display Name, Running status, Sessions, and Commands.
- Deploy**: A section for deploying new applications, including fields for Context Path, Version, XML Configuration file path, and WAR or Directory path.

Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle > 30 minutes
/docs	None specified	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle > 30 minutes
/examples	None specified	Servlet and JSP Examples	true	0	Start Stop Reload Undeploy Expire sessions with idle > 30 minutes
/host-manager	None specified	Tomcat Host Manager Application	true	1	Start Stop Reload Undeploy Expire sessions with idle > 30 minutes
/manager	None specified	Tomcat Manager Application	true	1	Start Stop Reload Undeploy Expire sessions with idle > 30 minutes

The host-manager portal can be accessed in the same way, just by putting `/host-manager/html` after the `:8080` (in my configuration : <http://192.168.1.25:8080/host-manager/html>)



When you've got a smartphone connected to the same network as your Pi, it will be fun to look to the management portals with that device. Simply enter the same addresses in you phone's browser (<IP-adress-of-Raspberry>:8080/manager/html and (<IP-adress-of-Raspberry>:8080/host-manager/html ) and you'll see the management portal adapting to the other screen format and the "touch screen".

## Maven

Maven is used to "build" applications. Maven looks for all the components that are used in an application, checks for the right versions and compiles it all together to the "target".

It is quite easy to install maven:

**sudo apt-get install maven**

and after this you can check if everything went right:

**mvn --version** (two minus-signs before "version").

## Github

Github is a company, founded in 2008 and nowadays property of Microsoft. Github uses the opensource application “git” that is used to create and maintain distributed “repositories”.



A “repository” can -simply said- be seen as a code-library. A git-repository (and Github also) can be used to store source-code, documents, scripts etc etc. If you want to, all these “artefacts” can also be used by other git- or Github-users.

We’re going to “pull” the code for our HAPI-Fhirserver from Github, so firstly we’re going to install git on our Raspberry:

**sudo apt-get install git**

And a fast check of the installation:

**git --version** (two minus-signs before “version”)

## HAPI Fhir-server

A Fhir-server “speaks” FHIR, information in JSON or XML format, transferred using REST API’s.

This way of communication is perfect for computer systems talking with each other, but less self-explaining for us, humble humans. The FHIR-server we’re now going to install provides a “test client” which makes it easier for us to communicate with the FHIR-server. The test client accompanying our HAPI-server is able to communicate with our own HAPI FHIR-server, but also with one other (open) FHIR server using the Internet. This means that we don’t have to “populate” our server with thousands of patients, encounters and diagnoses to get an impression off the possibilities of FHIR.

Let’s first create a directory to put our download in. Make sure that you are in the “home” directory of the user “pi”:



**cd /home/pi** (being the user pi you could also use **cd ~** )

Create a directory “hapi” under /home/pi

**mkdir hapi**

Move into this newly created subdirectory:

**cd hapi**

If you want to verify that you’re in the right place, give:

**pwd**

and if everything is right, the Pi answers with:

**/home/pi/hapi**

Now we’re going to get (“clone”) the HAPI server from github:

**git clone https://github.com/hapifhir/hapi-fhir-jpaserver-starter**

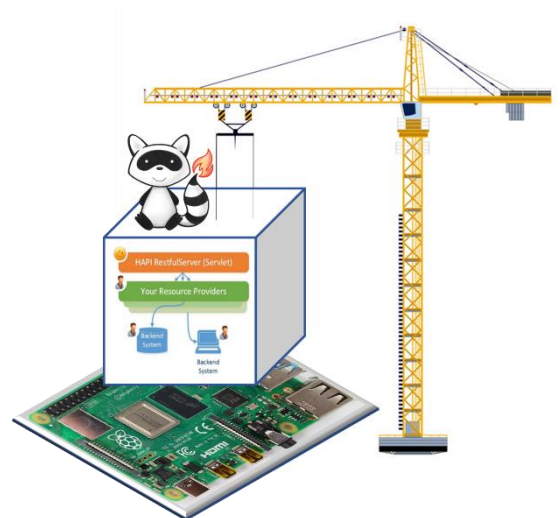
You’ll see a lot of information going by, after Pi’s reaction **cloning into hapi-fhir-jpastarter...** you can see “objects” being retrieved. When the Pi is ready, you’ll have to check-out the cloned code:

**cd hapi-fhir-jpaserver-starter**

**git checkout**

Ok. Everything is ready now to “build” your HAPI FHIR-server. Normally Maven will verify a lot of steps in this build by testing the intermediate results (unit-tests and integration tests).

In my case some of these tests failed, resulting in an interruption of the build process. The errors causing the failures however weren’t errors in building the FHIR-server, but errors in the programs used to do the testing.



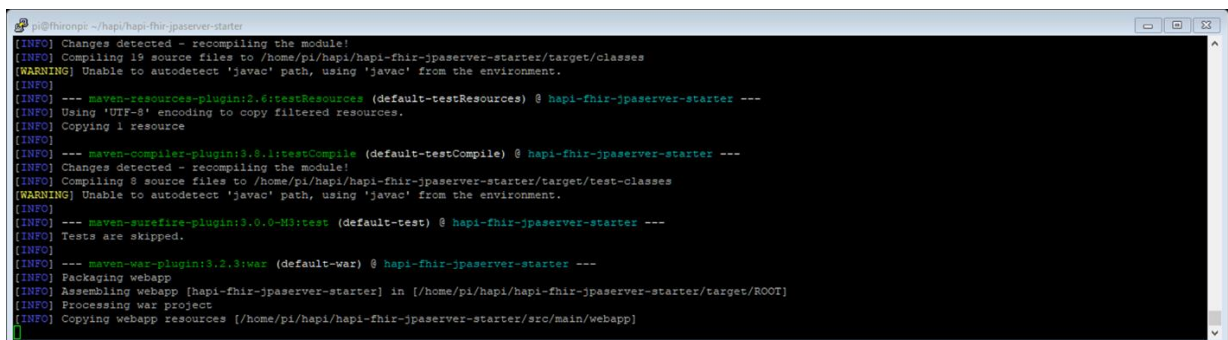


Skipping the tests speeded up the build process significantly and resulted in a perfectly working FHIR-server.

### **mvn clean install -DskipTests**

(standing in the directory /home/pi/hapi-fhir-jpaserver-starter).

Now Maven presents a lot of information. Sometimes you'll see warnings (in my case about the javac-path not being found and some about duplicate names) but the build continues.

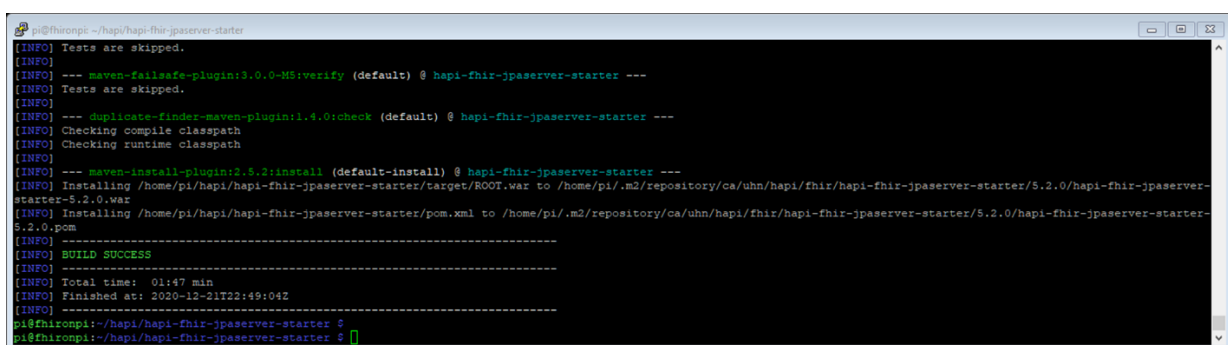


```
pi@thirompi: ~/hapi/hapi-fhir-jpaserver-starter
[INFO] Changes detected - Recompiling the module!
[INFO] Compiling 19 source files to /home/pi/hapi/hapi-fhir-jpaserver-starter/target/classes
[WARNING] Unable to autodetect 'javac' path, using 'javac' from the environment.
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ hapi-fhir-jpaserver-starter ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ hapi-fhir-jpaserver-starter ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 8 source files to /home/pi/hapi/hapi-fhir-jpaserver-starter/target/test-classes
[WARNING] Unable to autodetect 'javac' path, using 'javac' from the environment.
[INFO] --- maven-surefire-plugin:3.0.0-M3:test (default-test) @ hapi-fhir-jpaserver-starter ---
[INFO] Tests are skipped.
[INFO] --- maven-war-plugin:3.2.3:war (default-war) @ hapi-fhir-jpaserver-starter ---
[INFO] Packaging webapp
[INFO] Assembling webapp [hapi-fhir-jpaserver-starter] in [/home/pi/hapi/hapi-fhir-jpaserver-starter/target/ROOT]
[INFO] Processing war project
[INFO] Copying webapp resources [/home/pi/hapi/hapi-fhir-jpaserver-starter/src/main/webapp]
```

On my Raspberry Pi building the HAPI FHIR-server took between two and three minutes.

## **BUILD SUCCES**

To your relief you'll see a message of Maven when it finishes the build. Maven will also show the time used for this.



```
pi@thirompi: ~/hapi/hapi-fhir-jpaserver-starter
[INFO] Tests are skipped.
[INFO] --- maven-failsafe-plugin:3.0.0-M3:verify (default) @ hapi-fhir-jpaserver-starter ---
[INFO] Tests are skipped.
[INFO] --- duplicate-finder-maven-plugin:1.4.0:check (default) @ hapi-fhir-jpaserver-starter ---
[INFO] Checking compile classpath
[INFO] Checking runtime classpath
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ hapi-fhir-jpaserver-starter ---
[INFO] Installing /home/pi/hapi/hapi-fhir-jpaserver-starter/target/ROOT.war to /home/pi/.m2/repository/ca/uhn/hapi/fhir/hapi-fhir-jpaserver-starter/5.2.0/hapi-fhir-jpaserver-starter-5.2.0.war
[INFO] Installing /home/pi/hapi/hapi-fhir-jpaserver-starter/pom.xml to /home/pi/.m2/repository/ca/uhn/hapi/fhir/hapi-fhir-jpaserver-starter/5.2.0/hapi-fhir-jpaserver-starter-5.2.0.pom
[INFO] BUILD SUCCESS
[INFO]
[INFO] -----
[INFO] Total time: 01:47 min
[INFO] Finished at: 2020-12-21T22:49:04Z
[INFO] -----
pi@thirompi: ~/hapi/hapi-fhir-jpaserver-starter $
```

Now we've to do some preparations since the HAPI FHIR-server uses a small "H2" database to store its information in.

HAPI wants to put this database in a directory that doesn't exist yet, but is not able (under the user "tomcat") to create that directory. HAPI wants this

directory directly under the top directory / (namely in **/target**) but normal users in Unix environments are not allowed to create directories there.

This means that we've got to do that ourselves:

```
sudo mkdir /target
```

```
sudo mkdir /target/database
```

Now we've got to make sure that tomcat can access that directory:

```
sudo chown tomcat:tomcat /target
```

Ok!

Now we can copy the results of the "build" to the tomcat webapp environment:

```
sudo cp /home/pi/hapi/hapi-fhir-jpaserver-starter/target/ROOT.war  
/opt/tomcat/webapps/ROOT.war
```

Give tomcat access to this file:

```
sudo chown tomcat:tomcat /opt/tomcat/webapps/ROOT.war
```

Tomcat is going to unpack this "archive" in the directory:

```
/opt/tomcat/webapps/ROOT
```

but this directory already exists as a result of the default-installation of Tomcat. This means that we've firstly got to rename the existing ROOT directory:

```
sudo mv /opt/tomcat/webapps/ROOT /opt/tomcat/webapps/ROOT_OLD
```

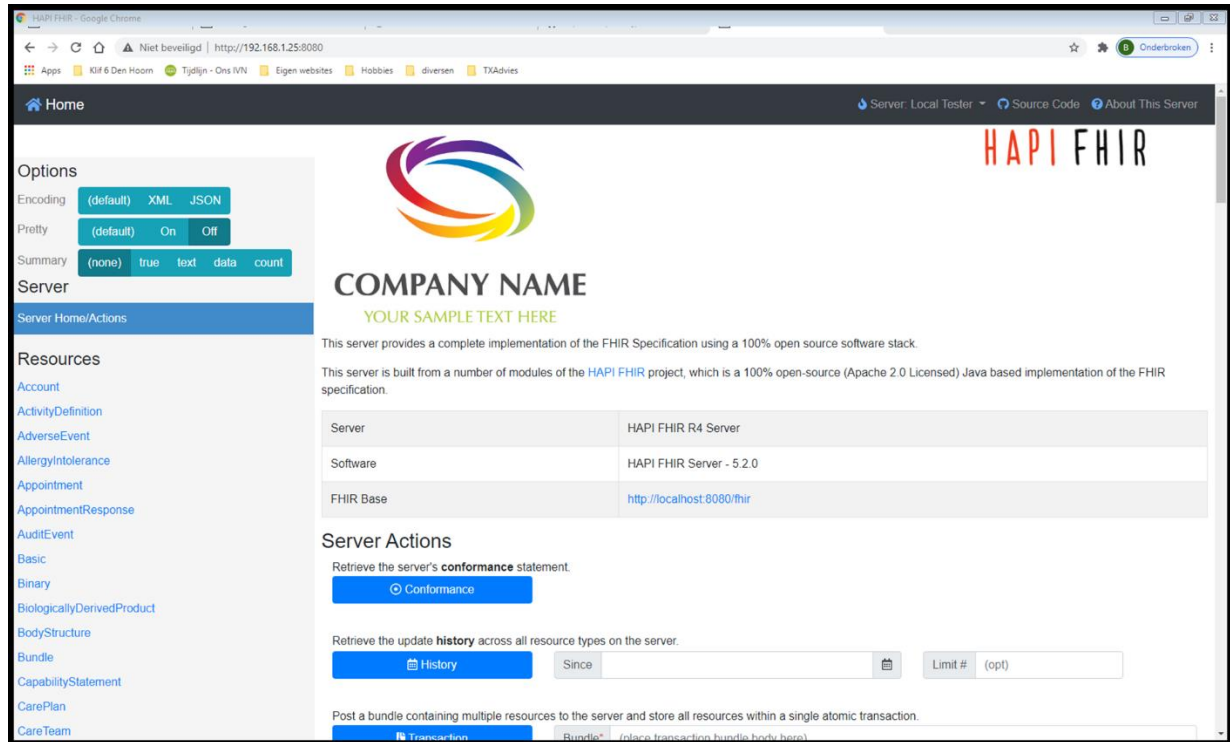
That was all – finally- . Now restart tomcat:

```
sudo systemctl start tomcat
```

It can take some time (a minute or more) for tomcat to deploy the HAPI Fhirserver. Start your webbrowser again and go to the address:

```
<IP-address of your Raspberry>:8080.
```

Earlier you saw the tomcat welcome-page on this address, but you moved it (to ROOT\_OLD) and now you'll see the welcome page of the test client for your HAPI FHIR-server instead. Look around on this page and it is ok to push buttons etc. There is nothing you can destroy here.



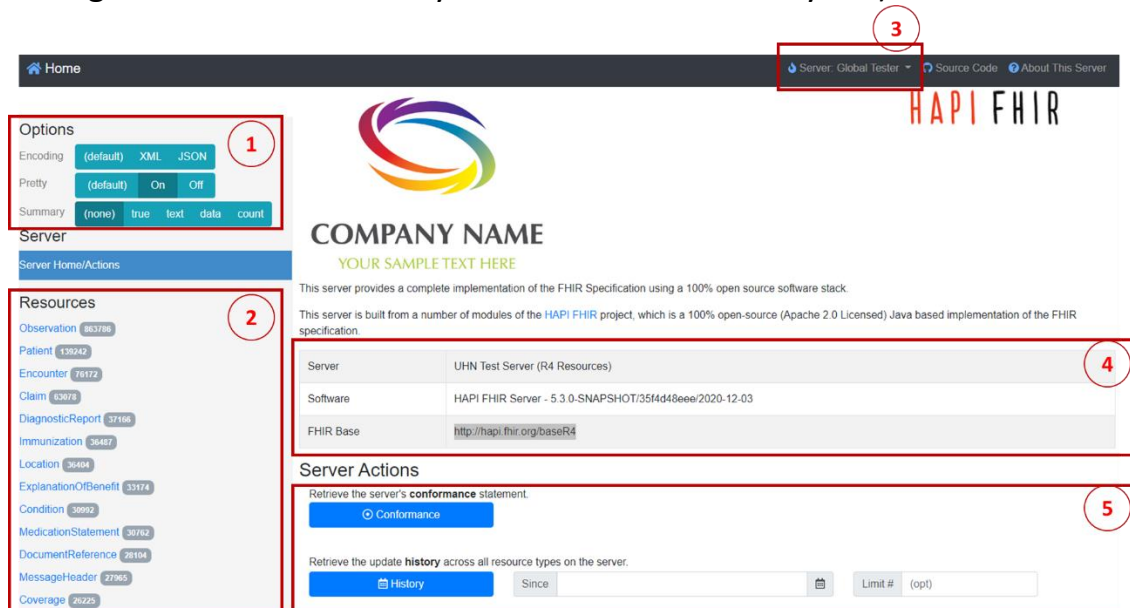
On the left side of the test page you'll see the FHIR-“resources” for which there is a place in your FHIR-server.

On top, on the right side of the grey bar you'll see the text “Server: Local Tester”. This means that the test client is communicating with the local FHIR-server, the server on your Raspberry Pi.

Click here and choose “Server: Global Tester”. Now you're looking into the open FHIR server <http://hapi.fhir.org/baseR4> . Suddenly you can access thousands of patients, , “encounters”, “observations” etc. This probably gives an impression about the strength of FHIR: using FHIR, healthcare information can be accessed from whatever which environment. FHIR has all what is needed to become the “lingua franco” in healthcare!

## The “Testclient”

Let’s have look at our newly installed “testclient” ( the numbered red frames in the figure below are not on your screen but drawn by me).



According to my discoveries the fields and buttons in the test client have the following purposes:

Frame nr.	Function
1	This sets the way you want to see the answers of the FHIR-server. You may choose between XML and JSON. Apart from that you can ask for a “prettified” output that is easier to read for us humans. I find “JSON”-output in a “pretty” way the easiest to understand, but if you are an XML-guru you’ll probably decide otherwise.
2	This is the (large) pane with all the FHIR-“resources” in the server you’re connected with. In your own FHIR-server there are a lot of resources present, however they are not filled with data. After connecting the “test client” with the global (test-) server you’ll see enormous numbers of data in the resources (the above screen-print demonstrates this). We’ll soon enter some data in your own FHIR-server too!
3	This is where you choose between your own server (Local Tester) and the test server at <a href="http://hapi.fhir.org">http://hapi.fhir.org</a> (Global Tester).
4	Gives information about the FHIR-server you’re connected with.
5	Here you can ask for the “conformance statement” of the FHIR-server you’re connected with. The “conformance statement” of a

Frame nr.	Function
	FHIR-server tells (in FHIR-format) which resources are known by this server and how you can access them. FHIR-servers are (mostly) capable of handling different FHIR-versions, like DSTU2, DSTU3 and R4. The “history”-button presents an oversight over the different versions over time supported by the server.

## Adding patients

Finally we’re going to enter some information in our own FHIR-server. We’ll -of course- start with a patient. Since the FHIR-server doesn’t have a user interface (the server is meant to handle computer-computer communications) we need a “client” application to translate our input into something the server understands.

For this purpose we’re going to install the tool “Postman” on our PC. You can find the installation packages on <https://www.postman.com/downloads/> . Choose for a local installation on your PC, don’t try to use the Cloud-version because your Raspberry most likely is not accessible from the Internet.

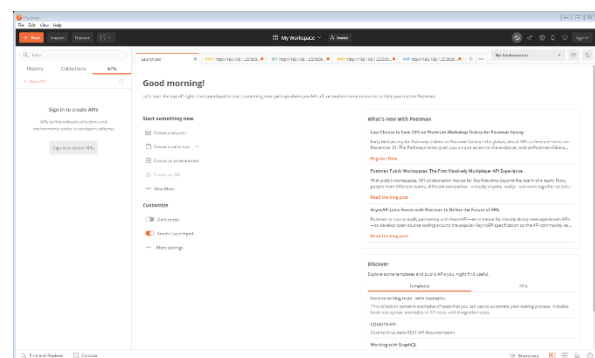
Postman is -simply said- a REST API client. REST stands for “Representational State Transfer” and is very standard on the Internet. If you want to know more about REST good starting points are: [https://en.wikipedia.org/wiki/Representational\\_state\\_transfer](https://en.wikipedia.org/wiki/Representational_state_transfer) and [https://en.wikipedia.org/wiki/Hypertext\\_Transfer\\_Protocol#Request\\_methods](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods).

After the installation of Postman, start the application and choose for “Create a request”.

In Postman choose for “POST” because we’re going to send (“Post”) a patient. In the *addressbar* of Postman you write the following:

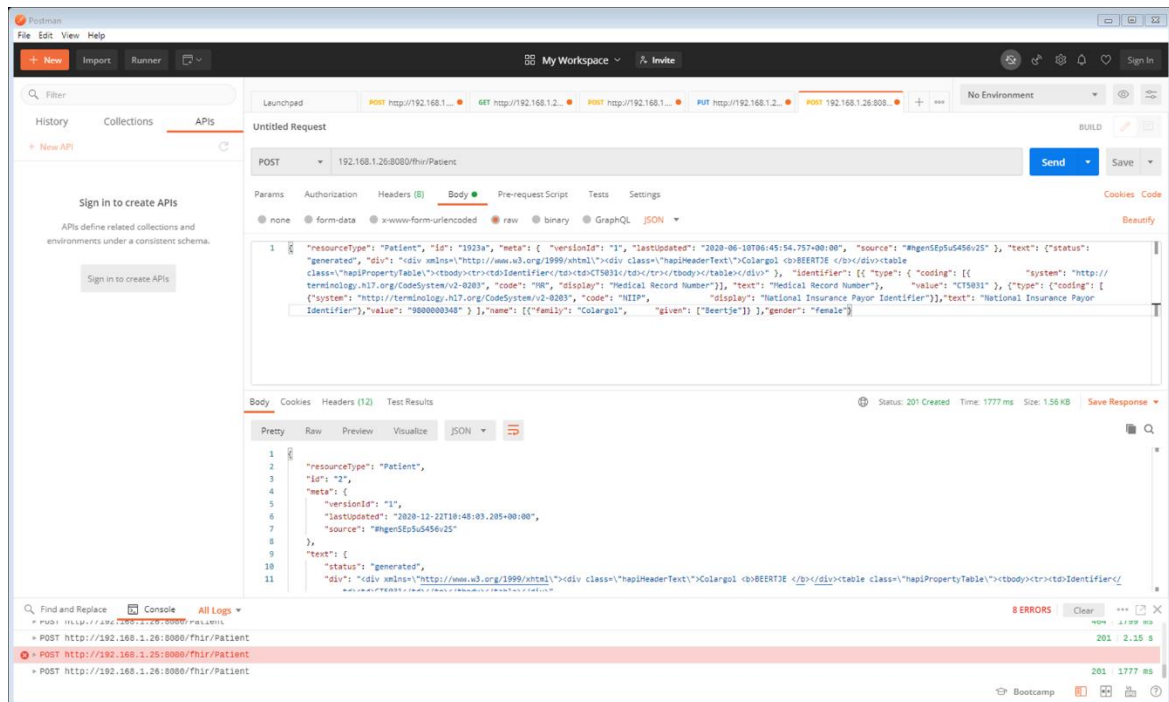
**<ip-addresss of your Pi>:8080/Patient**

Don’t forget the :8080 because this will end up with a “connection refused” message. In my situation I entered: 192.168.1.25:8080/fhir/Patient



Choose “raw” for the “body” of your message and for “JSON”(see the picture below). After this, mouse-click inside the “body” field and enter the following text:

```
{ "resourceType": "Patient", "id": "1923a", "meta": { "versionId": "1", "lastUpdated": "2020-06-10T06:45:54.757+00:00", "source": "#hgenSEp5uS456v2S" }, "text": { "status": "generated", "div": "<div xmlns=\"http://www.w3.org/1999/xhtml\"><div class=\"hapiHeaderText\">Duck <b>DONALD </b></div><table class=\"hapiPropertyTable\"><tbody><tr><td>Identifier</td><td>CT5031</td></tr></tbody></table></div>" }, "identifier": [ { "type": { "coding": [ { "system": "http://terminology.hl7.org/CodeSystem/v2-0203", "code": "MR", "display": "Medical Record Number" } ], "text": "Medical Record Number", "value": "CT5031" }, { "type": { "coding": [ { "system": "http://terminology.hl7.org/CodeSystem/v2-0203", "code": "NIIP", "display": "National Insurance Payor Identifier" } ], "text": "National Insurance Payor Identifier", "value": "9800000348" } }, "name": [ { "family": "Duck", "given": ["Donald"] } ], "gender": "male" }
```



Push the blue “Send” button and soon you’ll see the response: Donald Duck now is a patient in your FHIR-server.

Now leave Postman, start your webbrowser, go to the address:

**<ip-address of your Pi>:8080**

Choose the “local tester”, click on the Patient-resource, and do a search on the name Donald or the family-name Duck.

The screenshot shows the 'Patient Resource' interface. On the left is a sidebar with a list of FHIR resource types. The main area is titled 'Response' and shows a 'HTTP 200' status. Below this, 'Response Headers' and 'Result Body' are displayed. The 'Result Body' is a JSON bundle (1684 bytes) containing one patient resource. The patient's ID is 'Patient/2\_history/1' and it was updated at '10:48:03'. Below the table, the 'Raw Message' shows the full JSON structure of the bundle, including metadata and the patient resource details like name, birth date, and identifiers.

ID	Updated
Patient/2_history/1	10:48:03

```
{
  "resourceType": "Bundle",
  "id": "d9208ac9-8062-464c-8367-30c0faf31785",
  "meta": {
    "lastUpdated": "2020-12-22T10:52:13.589+00:00"
  },
  "type": "searchset",
  "total": 1,
  "link": {
    "relation": "self",
    "url": "http://localhost:8080/fhir/Patient?pretty=true&name=Colargo1"
  },
  "entry": [
    {
      "fullUrl": "http://localhost:8080/fhir/Patient/2",
      "resource": {
        "resourceType": "Patient",
        "id": "2",
        "meta": {
          "versionId": "1",
          "lastUpdated": "2020-12-22T10:48:03.205+00:00",
          "source": "#hl7fhir-gtXPRESP"
        },
        "text": {
          "status": "generated",
          "div": "<div xmlns='http://www.w3.org/1999/xhtml'><div class='hapiHeader text'>Colargo1</div><table class='hapiPropertyTable'><tbody><tr><td>Identifier</td><td>CT5031</td></tr></tbody></table></div>"
        }
      },
      "identifier": [
        {
          "type": {
            "coding": [
              {
                "system": "http://terminology.hl7.org/CodeSystem/v2-0203",
                "code": "NM",
                "display": "Medical Record Number"
              }
            ],
            "text": "Medical Record Number"
          },
          "value": "CT5031"
        }
      ],
      "name": [
        {
          "type": {
            "coding": [
              {
                "system": "http://terminology.hl7.org/CodeSystem/v2-0203",
                "code": "NII",
                "display": "National Insurance Payer Identifier"
              }
            ]
          }
        }
      ]
    }
  ]
}
```

Have a look at the “body” of the patient you’ve sent to your FHIR-server. Change for example the given name and family name of Donald Duck into Gus Goose or so. Browse the Internet and look for ways to enter “Observations” in your FHIR server etc. Have fun!

## Finish

This is the end of this story about the installation of the HAPI FHIR-server on a Raspberry Pi. I hope you enjoyed it. Although this is not a simple project it is fun to see a small computer like the Pi handling a large innovation like FHIR. I also hope you’ll go on studying FHIR and experimenting with it. (Health-)care is important and FHIR can be a means to significantly enhance healthcare services!