UNIT V

Device Discovery Management

Bits to learn:

Device Discovery Management:

Device Discovery capabilities – Registering a device, De-register a device.

Cloud Services for IoT:

Introduction to Cloud Storage models and communication APIs Web-Server. Web server for IoT.

Device Discovery

Definition:

Wireless Gateways support a feature known as Device Discovery or *Universal Plug 'n' Play* (UPnP), which allows UPnP-capable devices on your home network to <u>automatically discover each other's presence and begin sharing data</u>.

Registering a device

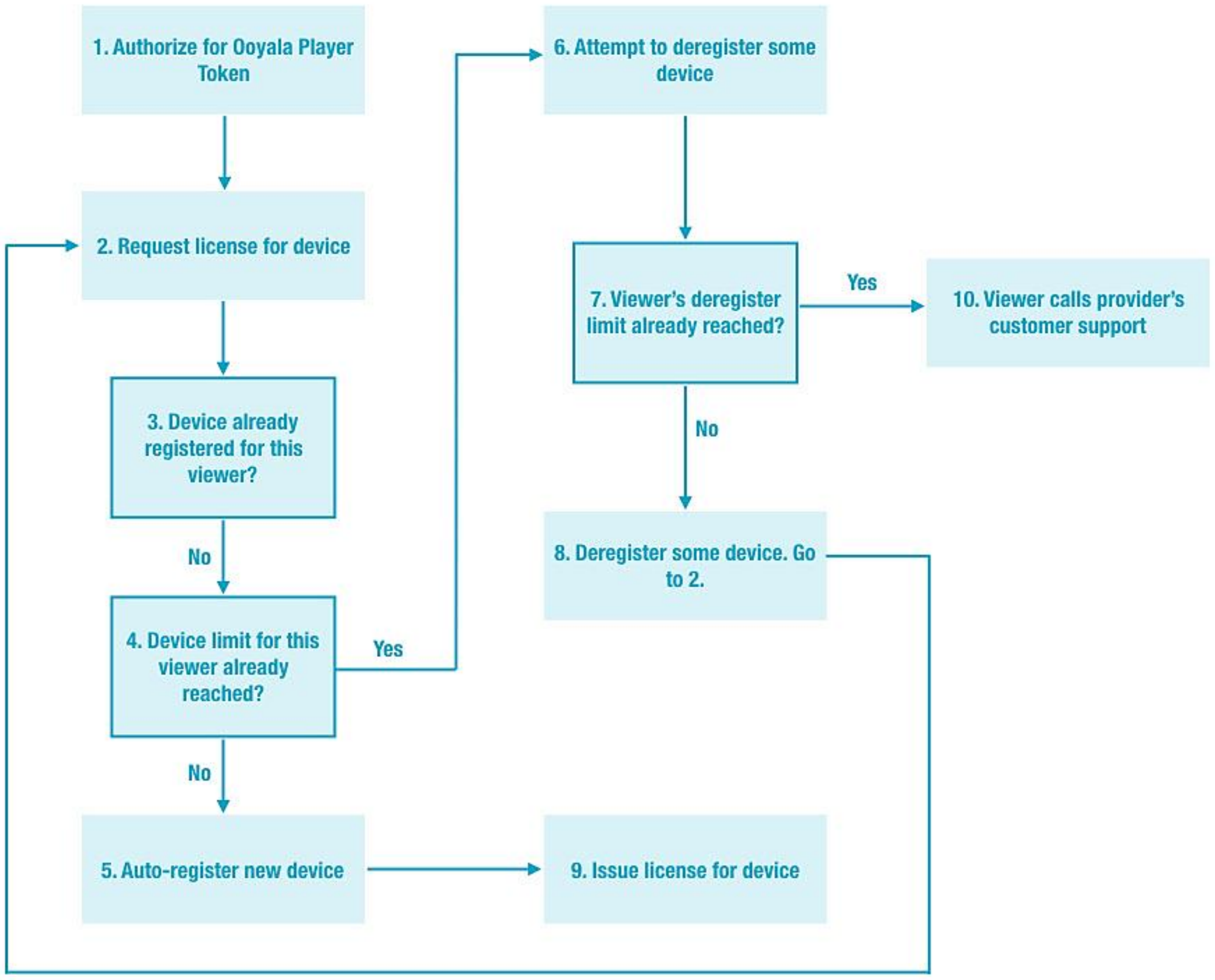
A device is registered during the first synchronization of the device. The Sync Server will allow users to register multiple devices with the same user ID and password depending on the value for the property of **Device,** in the control database.

Registering a device involves classifying the device as a device type, giving the device a name, and providing device information. Then you provide a *connection token or accept a token* that is generated by IoT Platform.

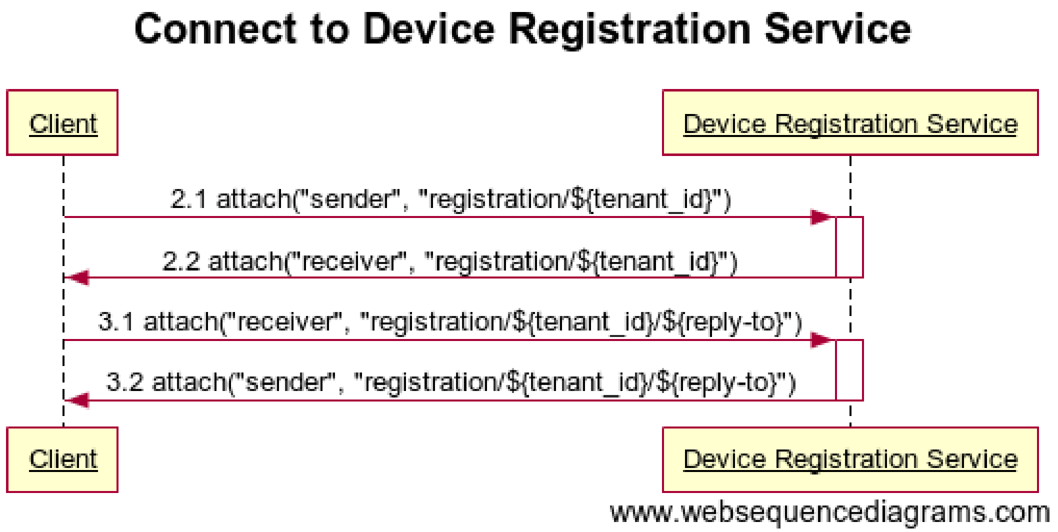
Example 1: Ooyala Player

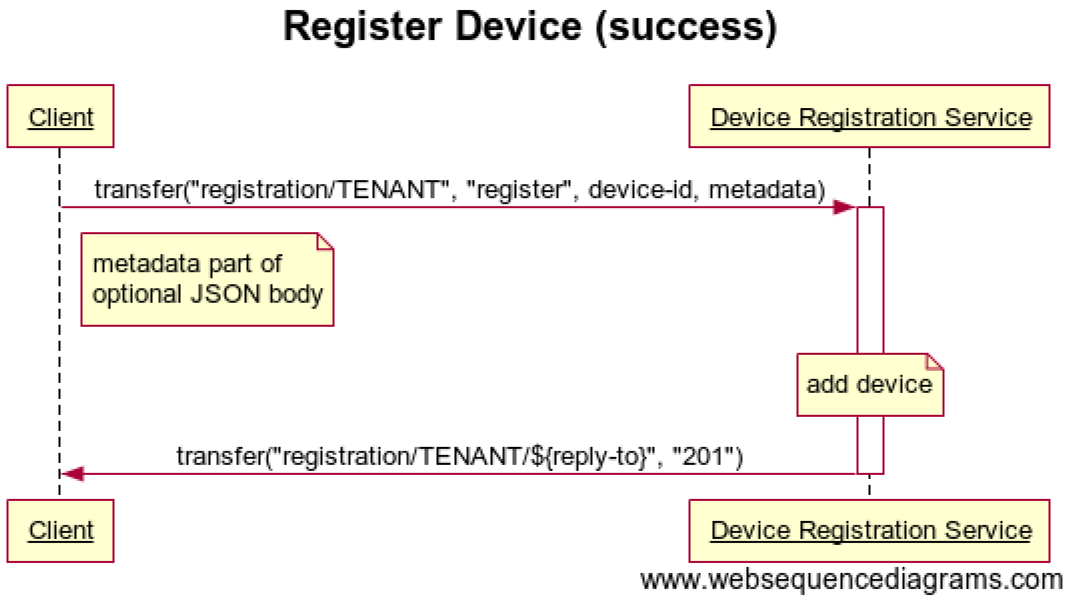
- **Ooyala Player** V4 is a next-generation video **player** that provides an engaging, personalized, responsive, content aware, and robust video playback experience that is consistent across devices and platforms.

- **Ooyala Player** V4 for HTML5 is a Web-based **player** that runs in popular browsers across desktop and mobile devices.


1. Authorize for Ooyala Player Token
2. Request license for device
3. Device already registered for this viewer?
No
4. Device limit for this viewer already reached?
No
Yes
5. Auto-register new device
6. Attempt to deregister some device
7. Viewer's deregister limit already reached?
Yes
No
8. Deregister some device. Go to 2.
9. Issue license for device
10. Viewer calls provider's customer support

Example 2: Ooyala Player


Connect to Device Registration Service
Client
Device Registration Service
2.1 attach("sender", "registration/${tenant_id}")
2.2 attach("receiver", "registration/${tenant_id}")
3.1 attach("receiver", "registration/${tenant_id}/${reply-to}")
3.2 attach("sender", "registration/${tenant_id}/${reply-to}")
Client
Device Registration Service
www.websequencediagrams.com


Register Device (success)
Client
Device Registration Service
transfer("registration/TENANT", "register", device-id, metadata)
metadata part of optional JSON body
add device
transfer("registration/TENANT/${reply-to}", "201")
Client
Device Registration Service
www.websequencediagrams.com

Advanced Message Queuing Protocol (**AMQP**) is an open source published standard for asynchronous messaging by wire. **AMQP** enables encrypted and interoperable messaging between organizations and **applications**.

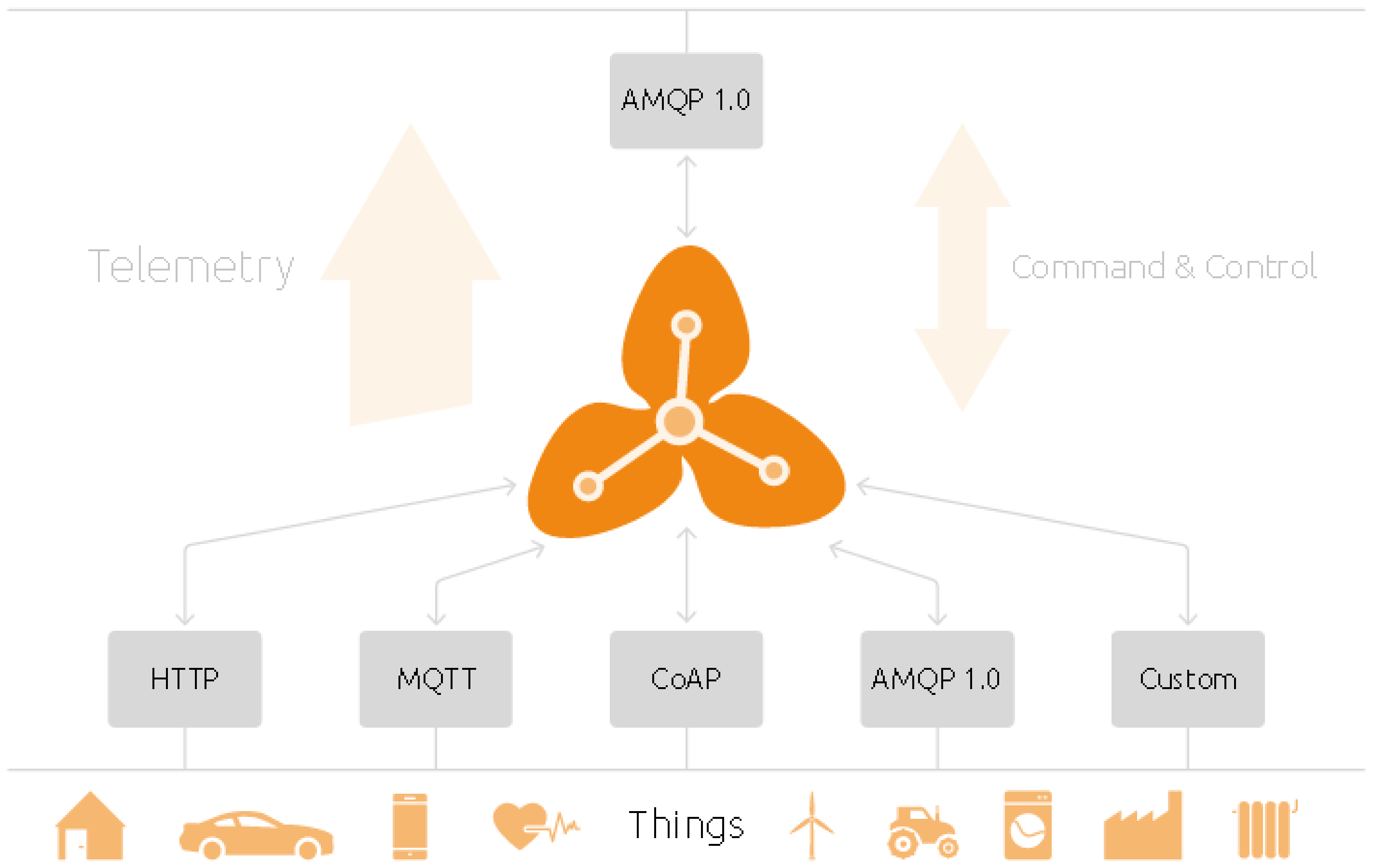
Eclipse Hono™ provides remote service interfaces for connecting large numbers of IoT devices to a back end and interacting with them in a uniform way regardless of the device communication protocol.

Preconditions

- The preconditions for performing any of the operations are as follows:
- Client has established an AMQP connection with Hono.
- Client has established an AMQP link in role *sender* with Hono using target address registration/${tenant_id}. This link is used by the client to send registration commands to Hono.
- Client has established an AMQP link in role *receiver* with Hono using source address registration/${tenant_id}/${reply-to} where *reply-to* may be any arbitrary string chosen by the client. This link is used by the client to receive responses to the registration requests it has sent to Hono. This link's source address is also referred to as the *reply-to* address for the request messages.

Register Device

- Clients use this command to initially *add* information about a new device to Hono. Each device is registered within a *tenant*, providing the scope of the device's identifier.
- This operation is *optional*, implementors of this API may provide other means for adding registration information, e.g. a RESTful API or a configuration file.



- Hono specifically supports scalable and secure ingestion of large volumes of sensor data by means of its Telemetry and Event APIs. Hono's Command & Control API allows for sending commands (request messages) to devices and receive a reply to such a command from a device in an asynchronous way.
- Finally, Hono provides APIs for integration with existing device and credentials management systems

WAMP - AutoBahn for IoT

- WAMP is an open standard WebSocket subprotocol that provides two application messaging patterns in one unified protocol:
  - routed **Remote Procedure Calls** and
  - **Publish & Subscribe**

WAMP - session between client and router

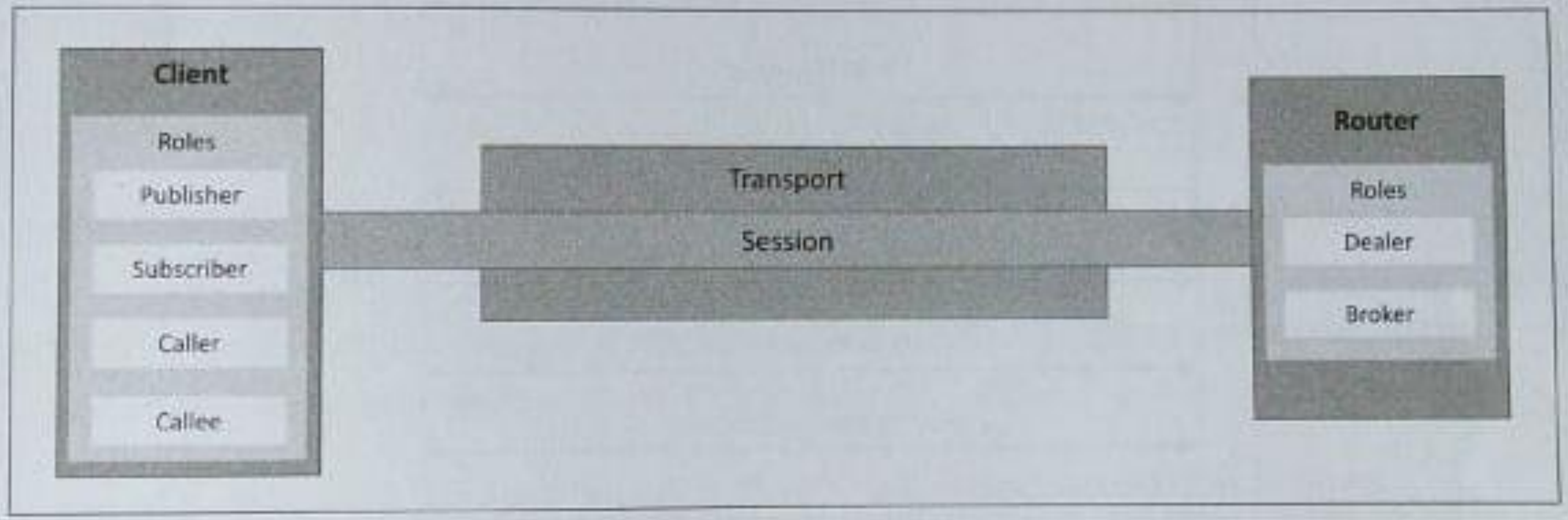


Figure 8.1: WAMP Session between Client and Router

WAMP protocol

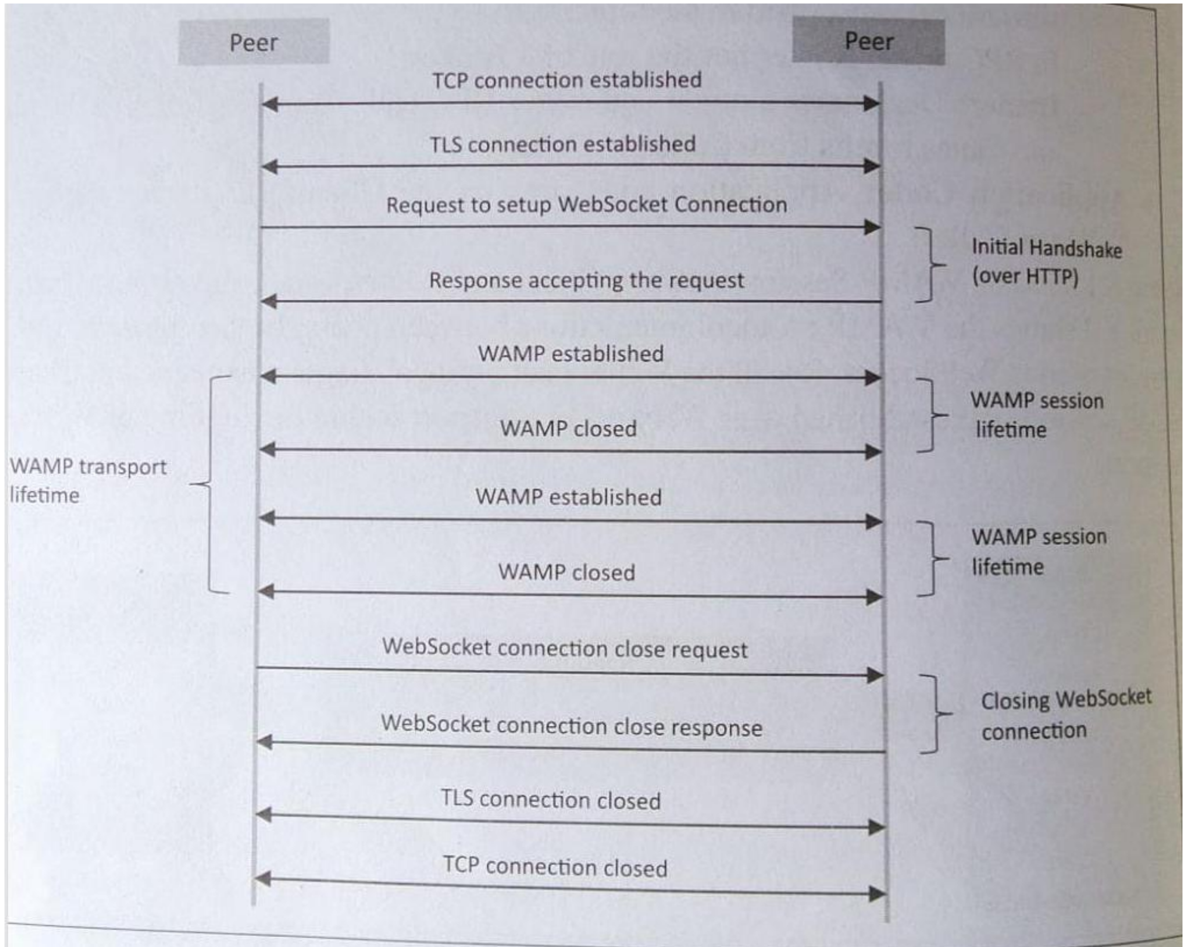
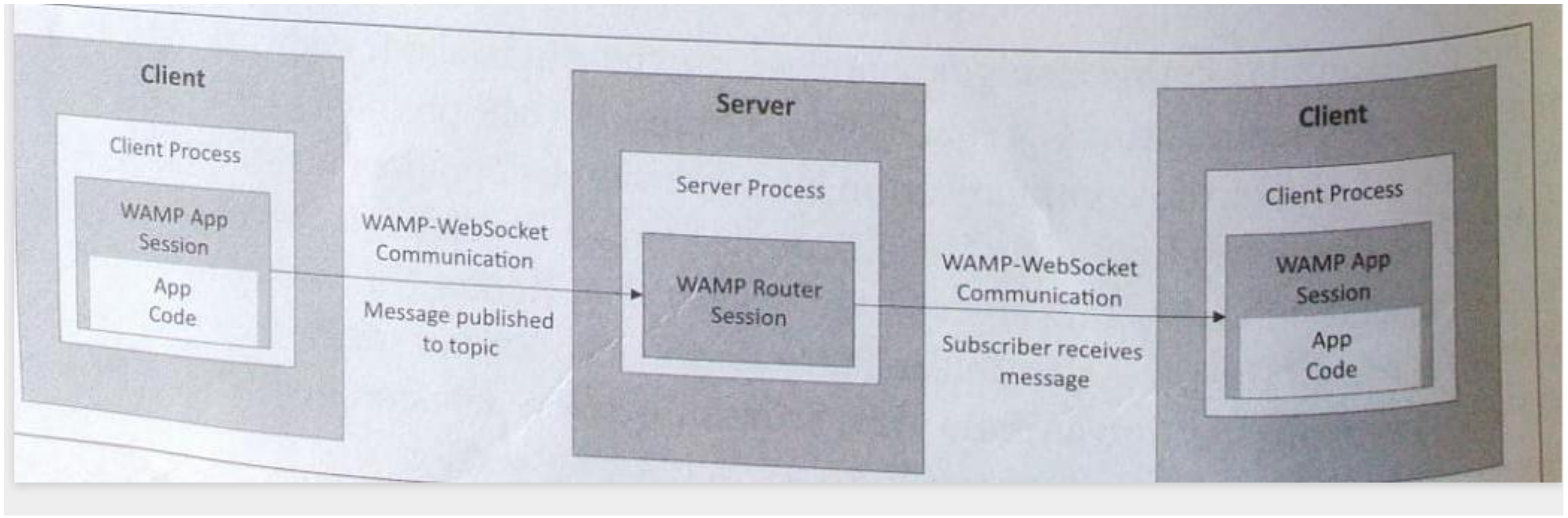

Peer
Peer
TCP connection established
TLS connection established
Request to setup WebSocket Connection
Response accepting the request
Initial Handshake (over HTTP)
WAMP established
WAMP closed
WAMP session lifetime
WAMP transport lifetime
WAMP established
WAMP closed
WAMP session lifetime
WebSocket connection close request
WebSocket connection close response
Closing WebSocket connection
TLS connection closed
TCP connection closed

Figure 8.2: WAMP protocol

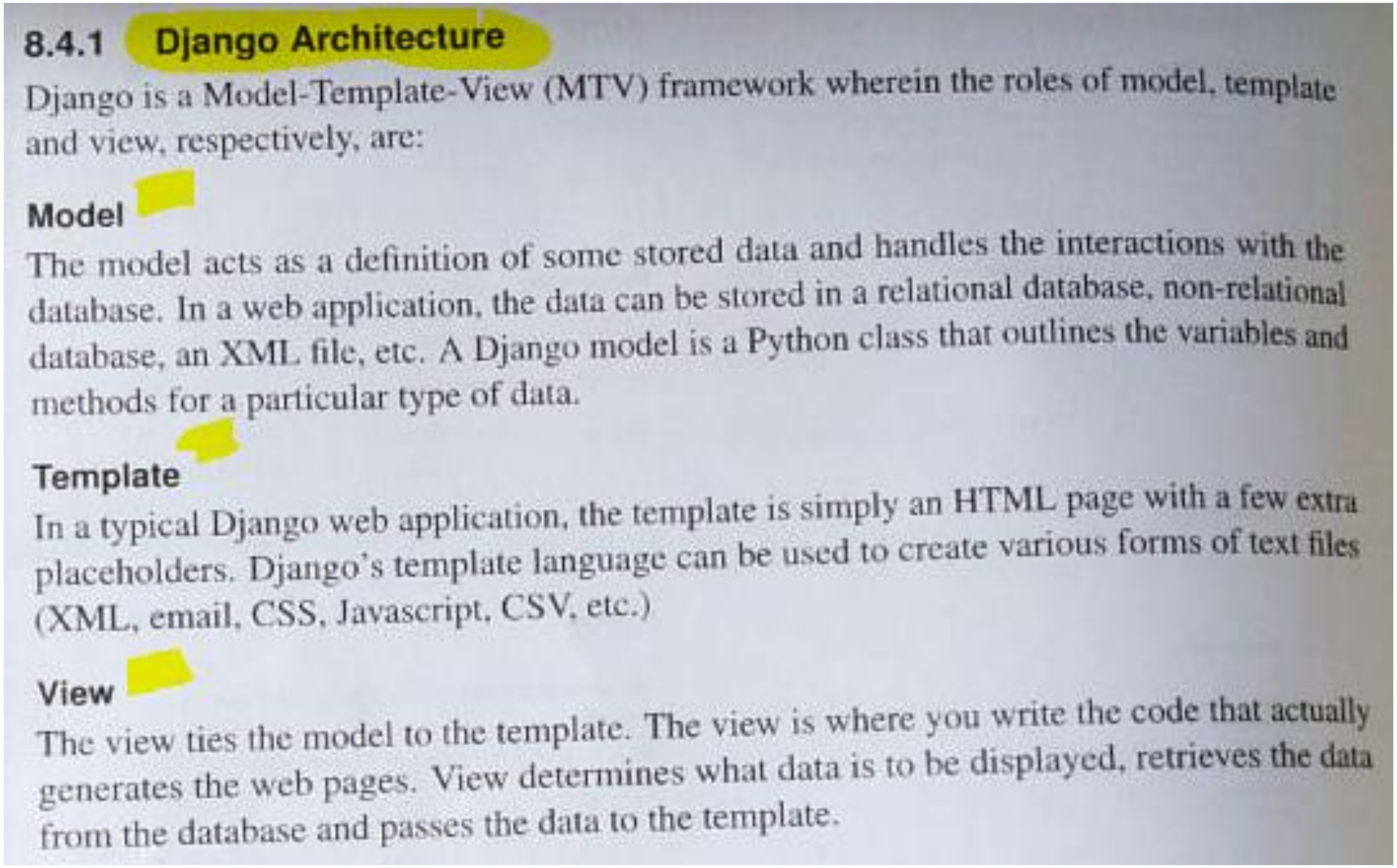
Publish – Subscribe messaging using WAMP - AutoBahn



Django

- Django is a Python-based free and open-source web framework
- **Django** is a free and open source web application framework written in Python. A framework is nothing more than a collection of modules that make development easier. They are grouped together, and allow you to create applications or websites from an existing source, instead of from scratch.

Django Architecture



8.4.1 Django Architecture

Django is a Model-Template-View (MTV) framework wherein the roles of model, template and view, respectively, are:

Model

The model acts as a definition of some stored data and handles the interactions with the database. In a web application, the data can be stored in a relational database, non-relational database, an XML file, etc. A Django model is a Python class that outlines the variables and methods for a particular type of data.

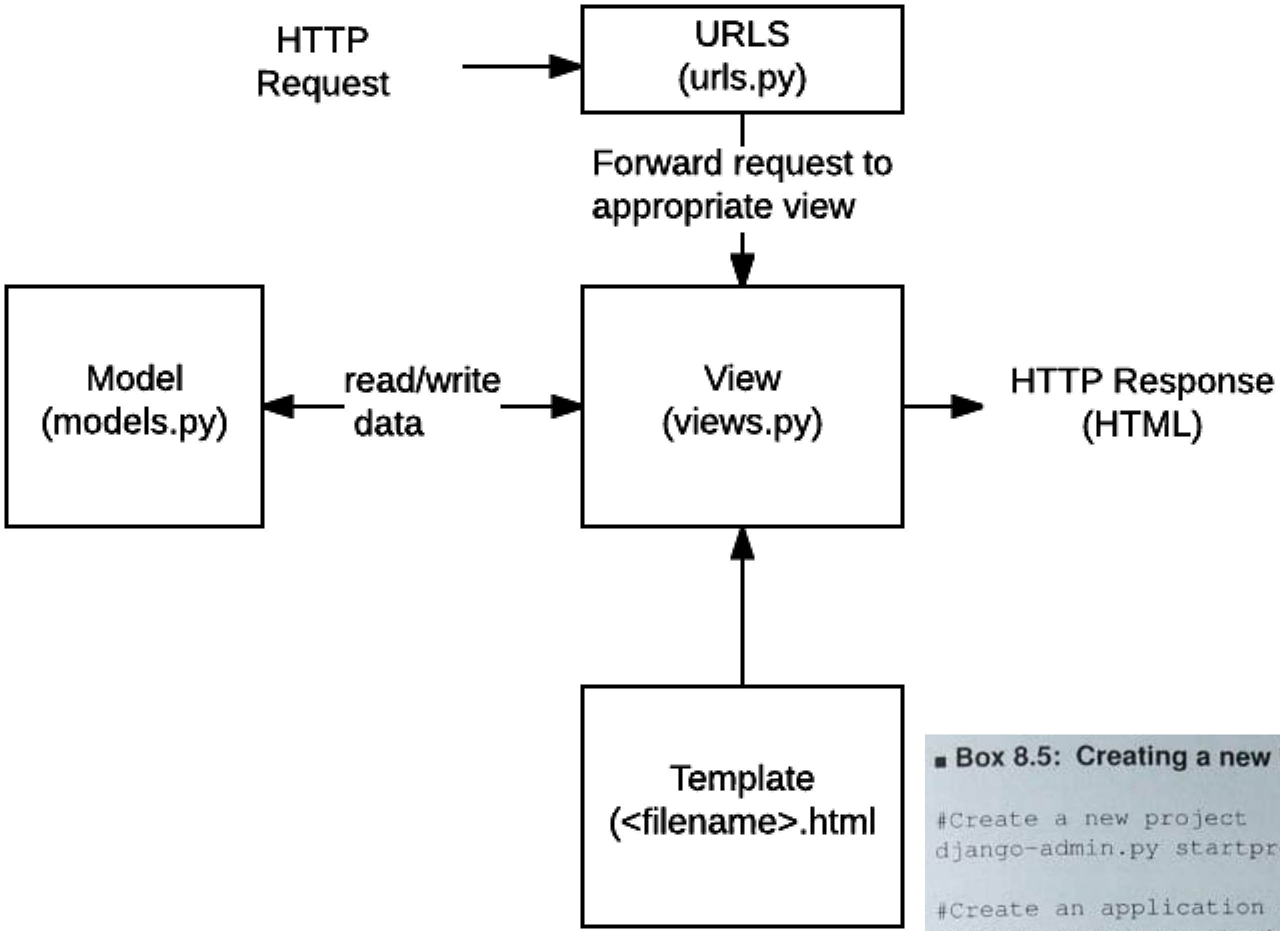
Template

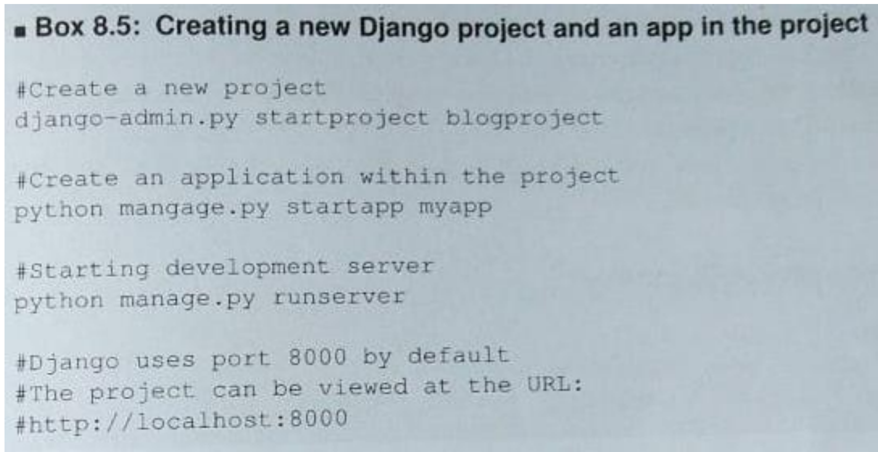
In a typical Django web application, the template is simply an HTML page with a few extra placeholders. Django's template language can be used to create various forms of text files (XML, email, CSS, Javascript, CSV, etc.)

View

The view ties the model to the template. The view is where you write the code that actually generates the web pages. View determines what data is to be displayed, retrieves the data from the database and passes the data to the template.

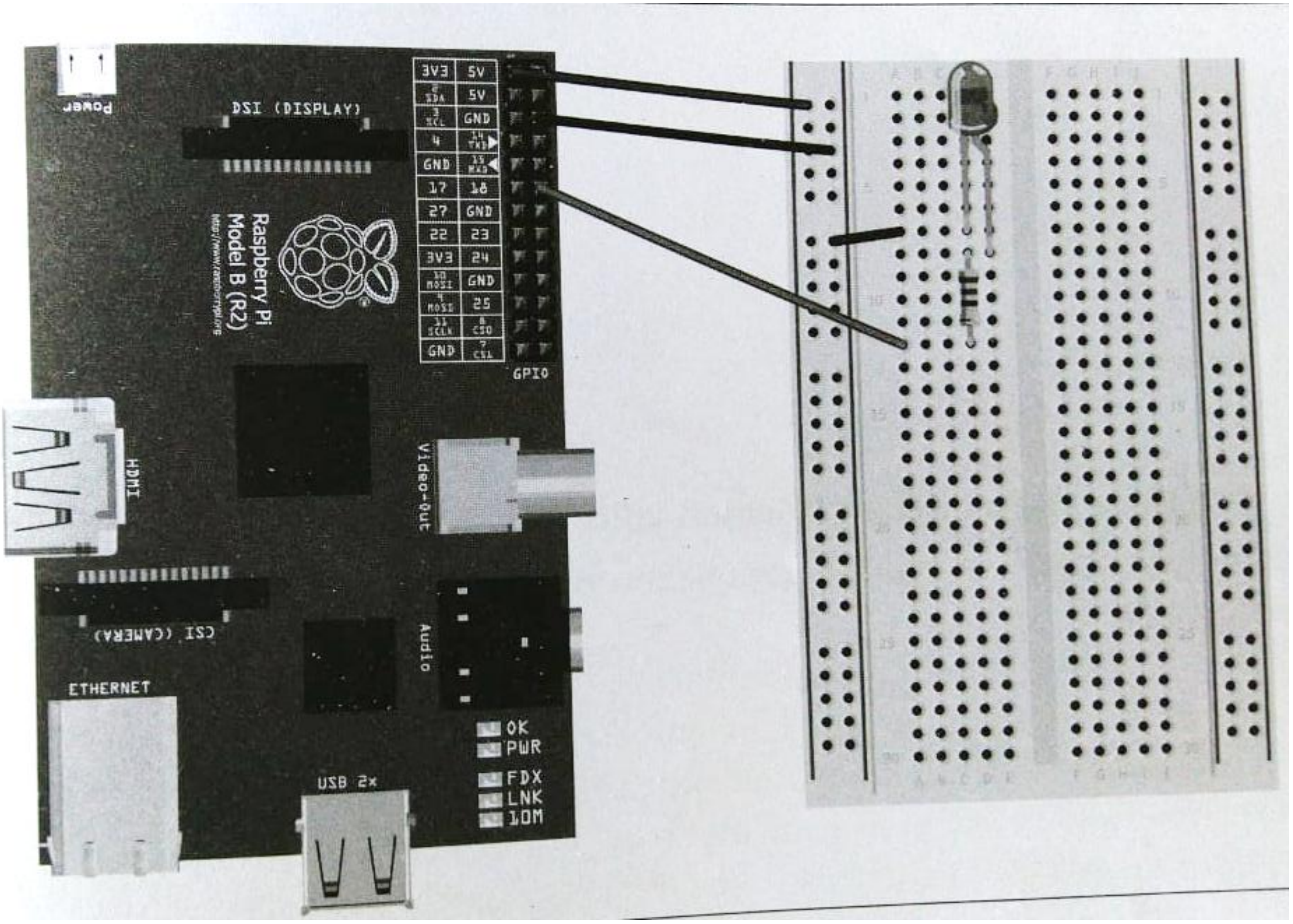
Creating a Django Project and App:

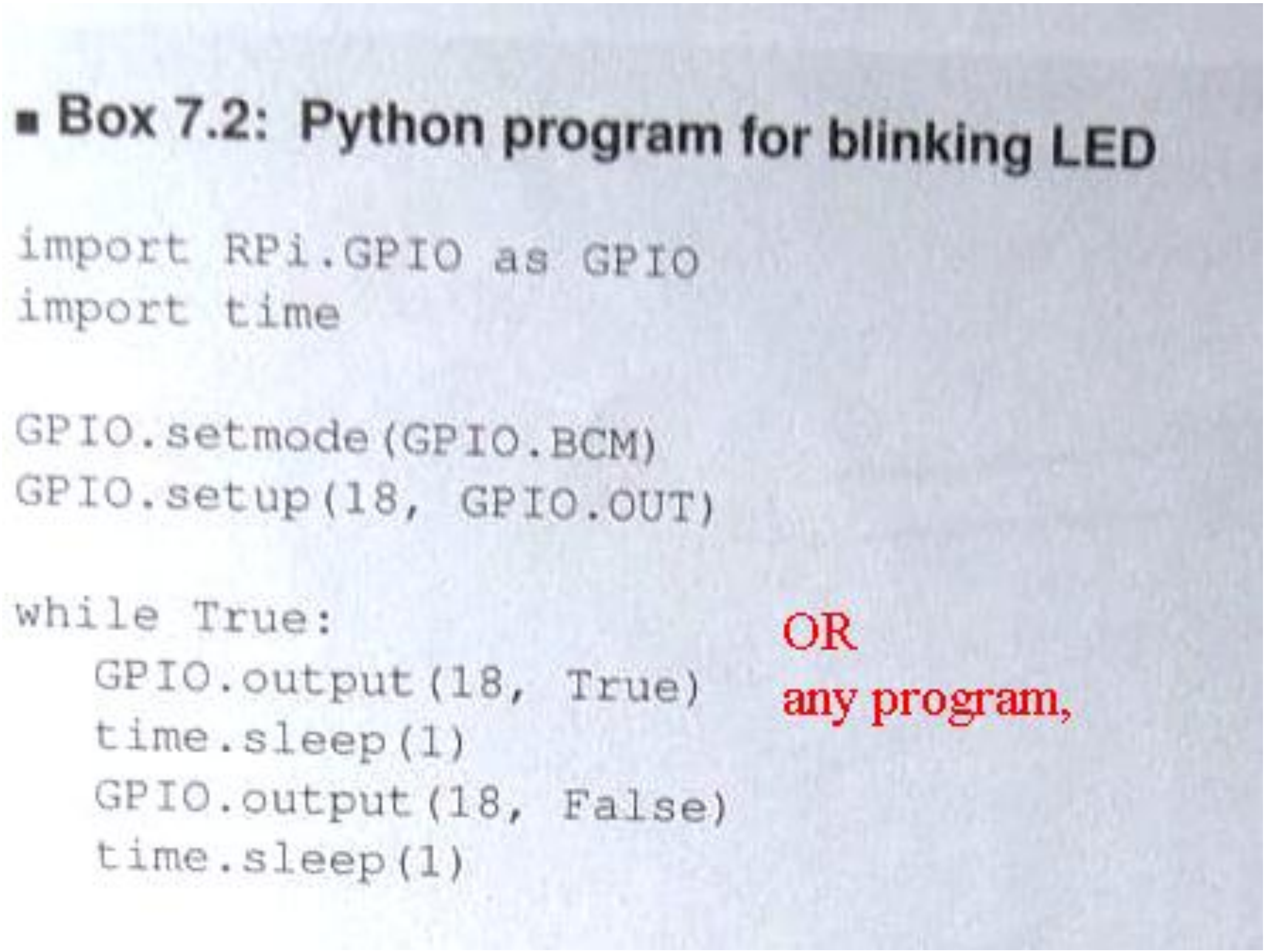



■ **Box 8.5: Creating a new Django project and an app in the project**

```
#Create a new project
django-admin.py startproject blogproject

#Create an application within the project
python mangage.py startapp myapp

#Starting development server
python manage.py runserver

#Django uses port 8000 by default
#The project can be viewed at the URL:
#http://localhost:8000
```

Controlling LED with Raspberry Pi





■ **Box 7.2: Python program for blinking LED**

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)

while True:
    GPIO.output(18, True)
    time.sleep(1)
    GPIO.output(18, False)
    time.sleep(1)
```

OR
any program,

Unit end questions:

- Describe Device Registration logical flow on a user self-service portal.
- Illustrate Device Deregister process, using request-response formats.
- Explain WAMP ? AutoBahn for IoT, with session between Client and Router.(text 1:Bahga & Madishetti, Chapter 8.2, page# 199)
- Demonstrate Django Architecture.(text 1:Bahga & Madishetti, Chapter 8.4, page# 206)
- Develop and explain with sketch & schematic to put the LEDs ON & OFF for Texas Instruments IOT device CC3200, the following LEDs, with delay in brackets: RED (1s),GREEN (0.5s), RED(1s), YELLOW (1s), GREEN (0.5s),YELLOW (1s).
- Develop and explain with python code and circuit diagram LED Blinking with Raspberry Pi and Python Program.(text 1:Bahga & Madishetti, Chapter 7.6.1, page# 186)

End