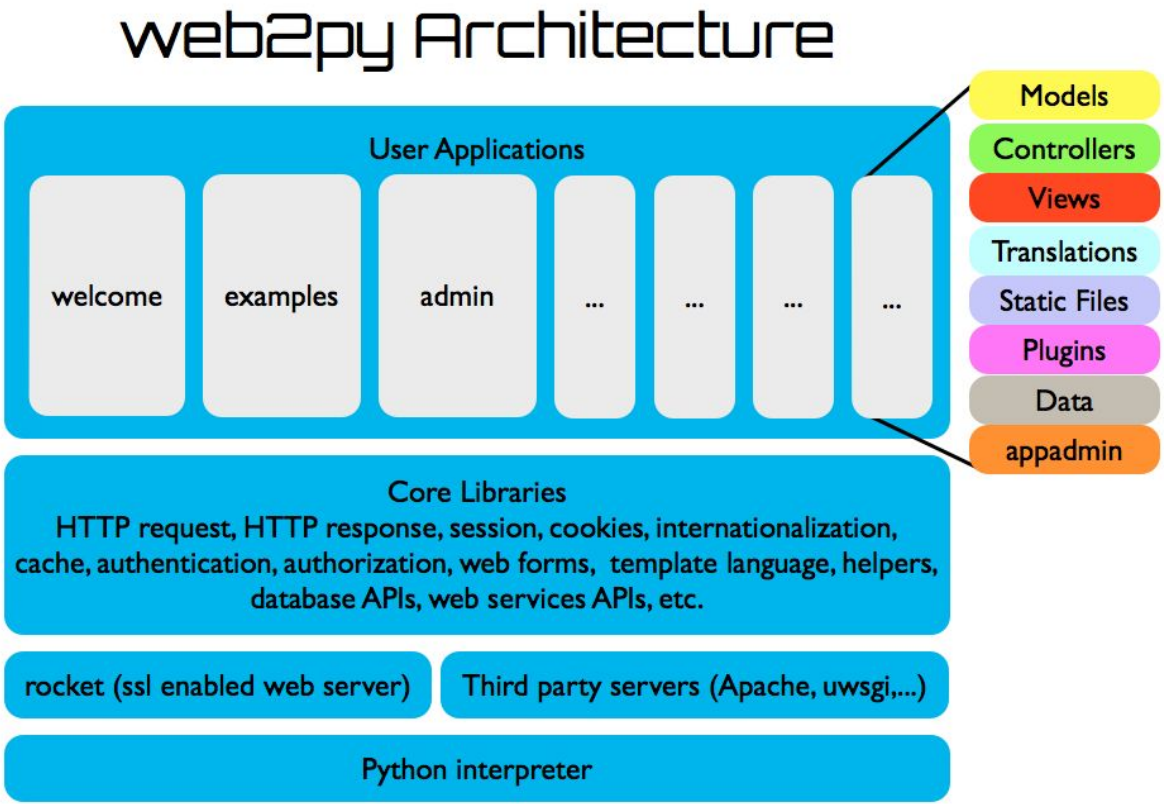
web2py

web2py structure



Core Libraries

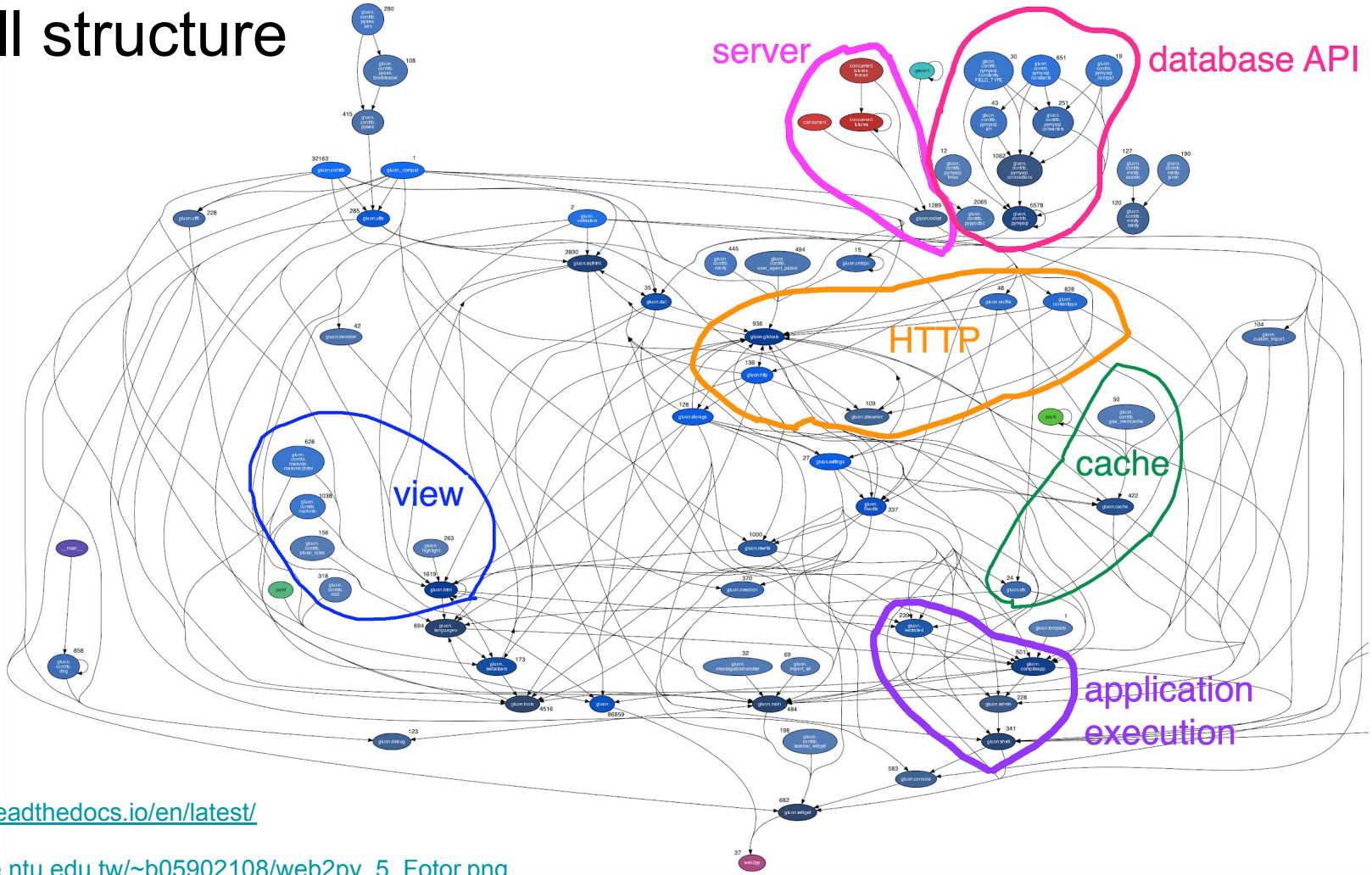
- **server**: main.py rocket.py fcgi.py
- **HTTP**: contenttype.py http.py global.py
- **session & cookies**: globals.py
- **internationalization**: languages.py
- **cache**: memcache.py cache.py cfs.py
- **authentication & authorization**: login_methods/*.py tools.py
- **web forms**: sqlhtml.py validators.py
- **view helper**: html.py highlight.py fpdf/*.py yatl(submodule)
- **helpers**: imageutils.py fileutils.py utils.py serializers.py
- **database APIs**: memdb.py sql.py
- **extensions & add-ons:** paymentech.py google_wallet.py
- **application execution:** compileapp.py restricted.py shell.py

detailed classification : https://hackmd.io/T-gPTCQwT4Wo9ishRSsnzQ?view

Folder structure

```
gluon/
    *.py
    packages/
        dal/
        yatl/
applications/
    app1/
        models/
        views/
        controller/
    app2
    admin
```

Overall structure



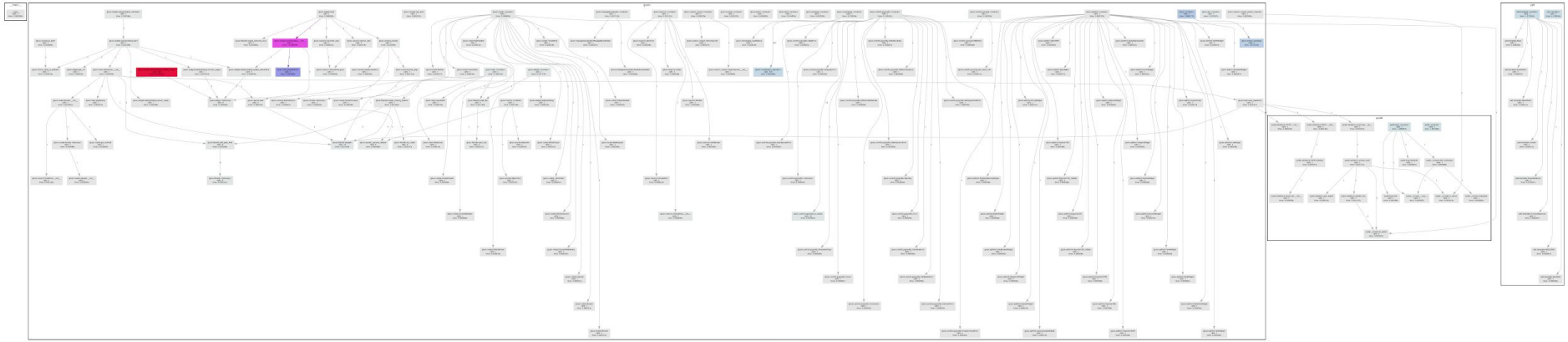
https://pydeps.readthedocs.io/en/latest/

https://www.csie.ntu.edu.tw/~b05902108/web2py_5_Fotor.png

runtime diagram

By using runtime analysis (pycallgraph --include "gluon.*" --include "pydal.*" --include "yatl.*" graphviz -- ./web2py.py)

operation used：start webserver, login as admin, add new app, interact with that app



By specifying the dependencies between codes, we can distribute our work according to:

1) lines of code
2) code dependency, just need to read the adjacent codes
3) personal preference

more detailed diagram：
https://www.csie.ntu.edu.tw/~b05902108/wide.png
pycallgraph：
https://github.com/gak/pycallgraph

code not in runtime diagram

gluon/
- authapi.py 808
- debug.py 123
- import_all.py 69
- sanitizer.py 1
- scheduler.py 1325
- sql.py 16
- tools.py 4516
- utf8.py 220

contrib/fpdf/ 3378
contrib/gateways/ 859
contrib/login_methods/ 2052
contrib/markdown/ 1388
contrib/memcache/ 981
contrib/pyrtf/ 1517
contrib/pysimplesoap/ 2704
contrib/pyuca/ 80

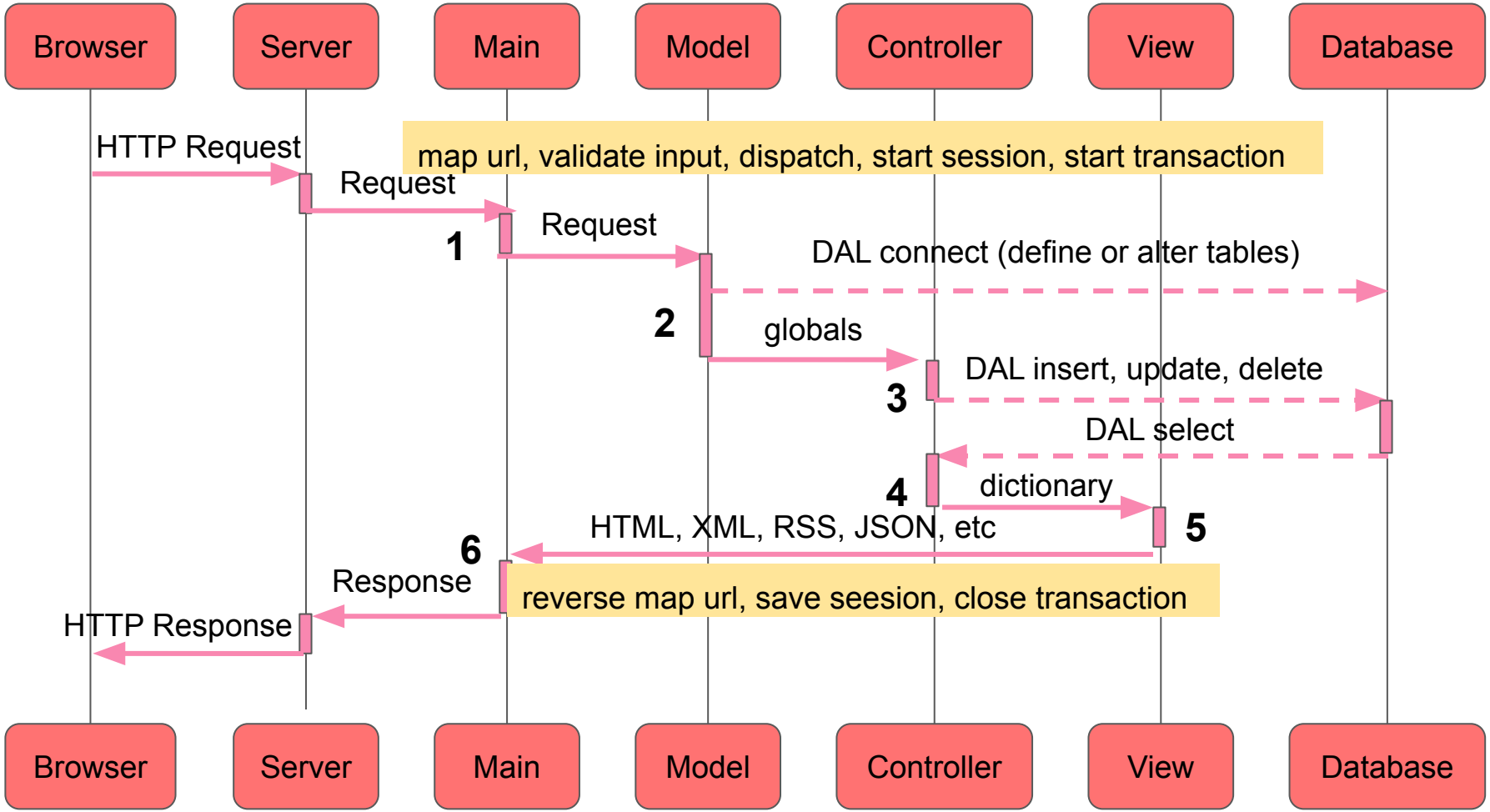
contrib/
- AuthorizeNet.py 193
- DowCommerce.py 192
- __init__.py 0
- appconfig.py 96
- autolinks.py 136
- dbg.py 858
- feedparser.py 3035
- gae_memcache.py 50
- gae_retry.py 34
- generics.py 60
- google_wallet.py 14
- heroku.py 19
- hypermedia.py 280
- imageutils.py 43
- memdb.py 563
- ordereddict.py 1
- pam.py 74
- paymentech.py 82
- pbkdf2.py 64
- pbkdf2_ctypes.py 139
- pdfinvoice.py 148
- populate.py 237

contrib/
- pyfpdf.py 3
- redis_cache.py 182
- redis_scheduler.py 572
- redis_session.py 162
- redis_utils.py 37
- shell.py 126
- simplejson.py 3
- simplejsonrpc.py 106
- sms_utils.py 109
- spreadsheet.py 504
- stripe.py 108
- taskbar_widget.py 198
- timecollect.py 71
- user_agent_parser.py 494
- webclient.py 150
- websocket_messaging.py 132

Imply that:

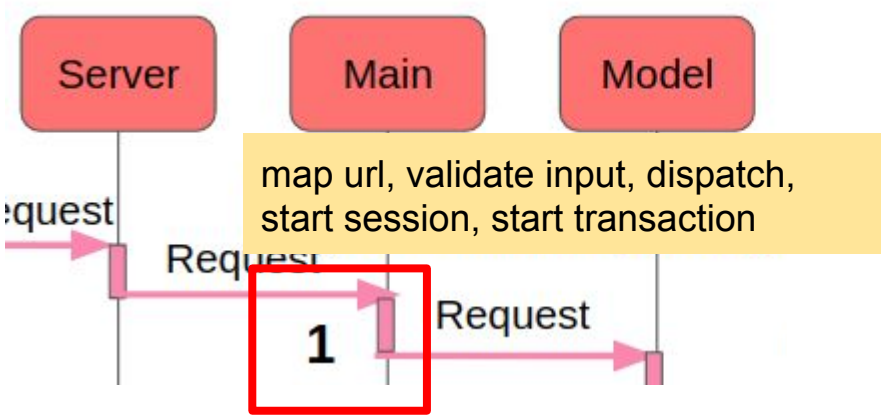
1) Lots of codes under contrib/ are not called.
2) Most of codes there are add-ons and have little thing to do with basic web operations.

Flow


Browser
Server
Main
Model
Controller
View
Database
HTTP Request
map url, validate input, dispatch, start session, start transaction
Request
Request
1
DAL connect (define or alter tables)
2
globals
DAL insert, update, delete
3
DAL select
4
dictionary
HTML, XML, RSS, JSON, etc
5
6
Response
reverse map url, save seesion, close transaction
HTTP Response
Browser
Server
Main
Model
Controller
View
Database

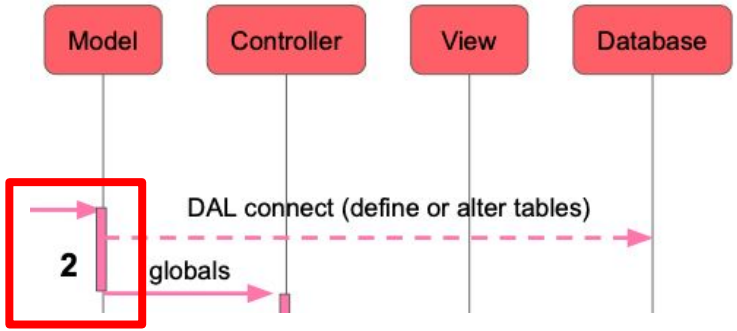
Dispatching

- URLs are mapped to Python modules and function calls
- Before calling the action, a few things happen:
  - Session object is created/retrieved
  - An execution environment for the request is created



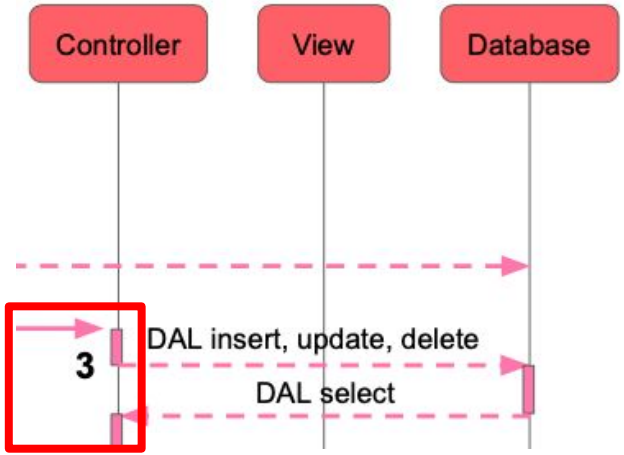
Migration / Glabal Variables Access

- Migration
  - Check whether or not the corresponding table exists
    - If it does not, it generates the SQL to create it and executes the SQL
    - If the table does exist but differs from the one being defined, it generates the SQL to alter the table and executes it
- Access all global variables in Model



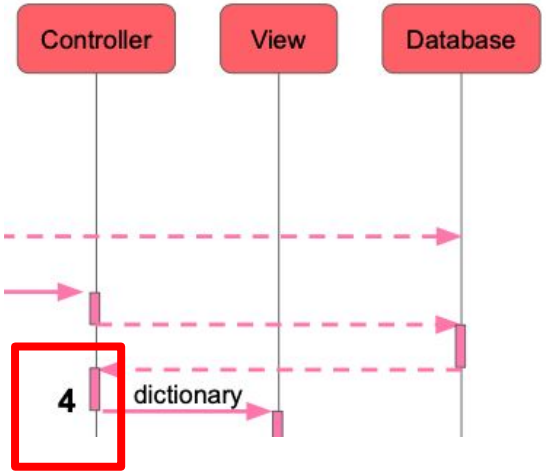
Database Query

- The dashed arrows represent communication with the database engine(s)
- The database queries can be written in
  - Raw SQL (discouraged)
  - web2py Database Abstraction Layer (recommended),
- web2py application code is not dependent on the specific database engine



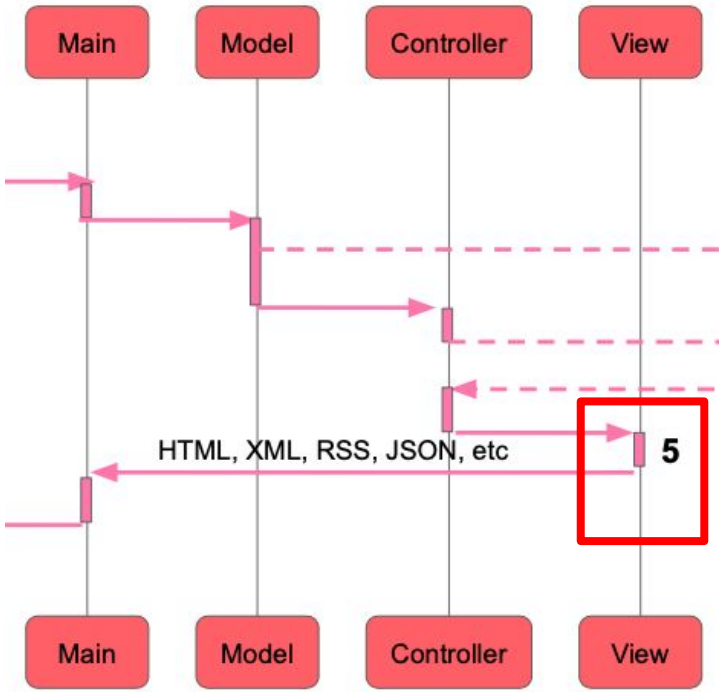
Controller Function Execution

- URLs are mapped to Python modules and function calls
- The controller may contain one or more functions (or “actions”)
- An **controller** actions is executed and may return a string, an iterable, a Python dictionary, etc.



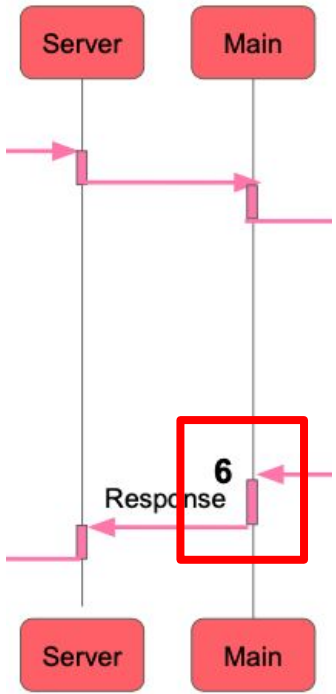
Rendering

- web2py tries to locate a **view** to the returned dictionary
  - The view must have the same name as action and the same extension as the requested page
  - Generates a generic view on failure
- The view sees global variables defined in model but not the global variables defined in controller



Committing Transaction

- All calls are wrapped into a transaction, and any uncaught exception causes the transaction to be rolled back. If the request succeeds, the transaction is committed
- web2py also handles sessions and session cookies automatically, and when a transaction is committed, the session is also stored, unless specified otherwise



PyDAL

PyDAL

- Introduction :
  - ➢ This package dynamically translates database languages to python in realtime.
  - ➢ It supports MySQL, MongoDB, Oracle and 12 other database languages.
- Example :

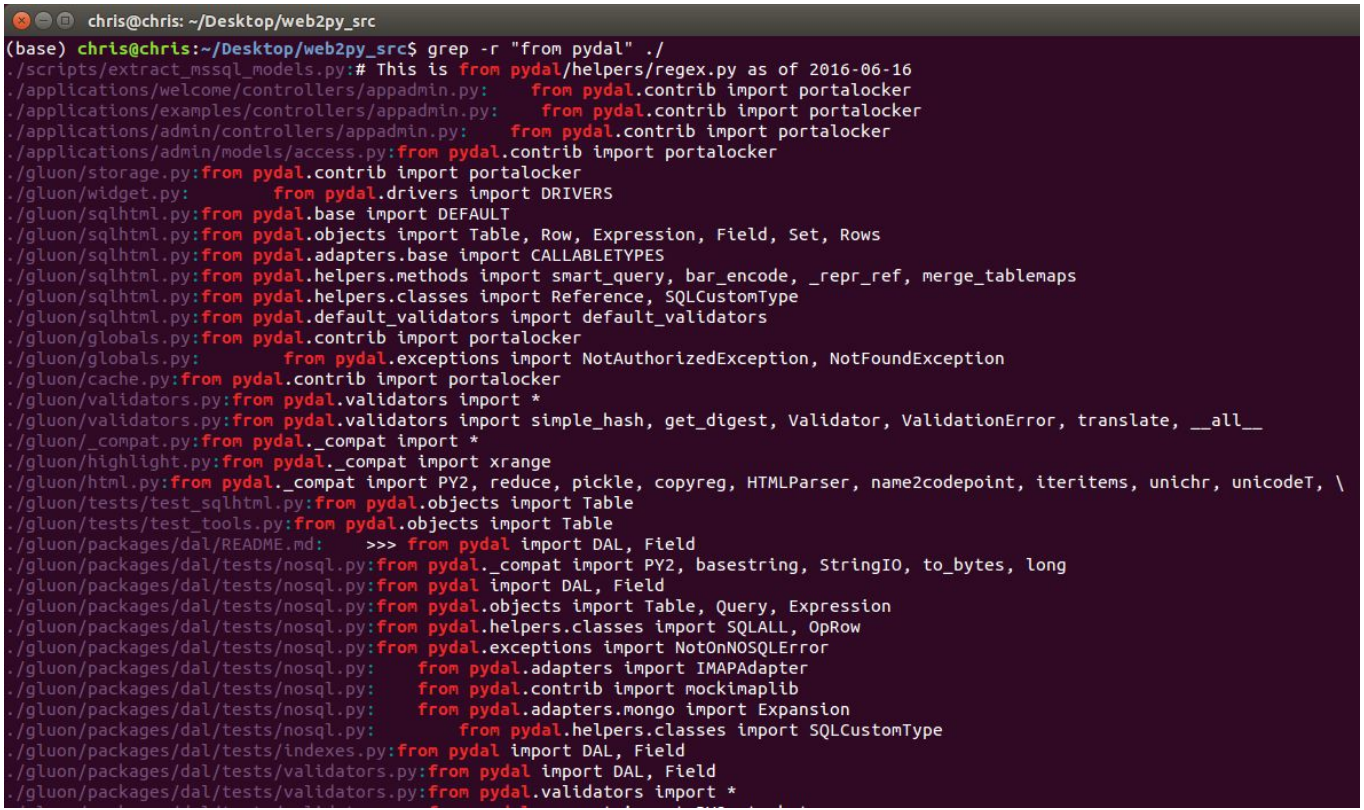
```
### create DAL connection (and create DB if it doesn't exist)
db = DAL(('sqlite://storage.sqlite','mysql://a:b@localhost/x'), folder=No

### define a table 'person' (create/alter as necessary)
person = db.define_table('person',Field('name','string'))

### insert a record
id = person.insert(name='James')

### retrieve it by arbitrary query
query = (person.name=='James') & (person.name.startswith('J'))
james = db(query).select(person.ALL)[0]
```

Files import pyDAL in web2py



- ✓ Incomprehensible
  - ✓ Many functions are massively used but hard to understand.
  - ✓ Ex: portalocker, Field, mockimaplib, etc.
- ✓ Proportion
  - ✓ # of python files ： 381
  - ✓ # of python files import pyDAL ： 33
  - ✓ Ratio ： 8.67%