**from** pytz **import** timezone

timezone\_user = session.user\_timezone **or** timezone(**'UTC'**)
timezone\_is\_unknown = session.user\_timezone **is** None

*# -\*- coding: utf-8 -\*-

# -------------------------------------------------------------------------
# This scaffolding model makes your app work on Google App Engine too
# File is released under public domain and you can use without limitations
# -------------------------------------------------------------------------***if** request.global\_settings.web2py\_version < **"2.14.1"**:
 **raise** HTTP(500, **"Requires web2py 2.13.3 or newer"**)

*# -------------------------------------------------------------------------
# if SSL/HTTPS is properly configured and you want all HTTP requests to
# be redirected to HTTPS, uncomment the line below:
# -------------------------------------------------------------------------
# request.requires\_user\_timezonehttps()

# -------------------------------------------------------------------------
# app configuration made easy. Look inside private/appconfig.ini
# -------------------------------------------------------------------------***from** gluon.contrib.appconfig **import** AppConfig

*# -------------------------------------------------------------------------
# once in production, remove reload=True to gain full speed
# -------------------------------------------------------------------------*myconf = AppConfig(reload=True)

**if not** request.env.web2py\_runtime\_gae:
 *# ---------------------------------------------------------------------
 # if NOT running on Google App Engine use SQLite or other DB
 # ---------------------------------------------------------------------* db = DAL(myconf.get(**'db.uri'**),
 pool\_size=myconf.get(**'db.pool\_size'**),
 migrate\_enabled=myconf.get(**'db.migrate'**),
 check\_reserved=[**'all'**])
**else**:
 *# ---------------------------------------------------------------------
 # connect to Google BigTable (optional 'google:datastore://namespace')
 # ---------------------------------------------------------------------* db = DAL(**'google:datastore+ndb'**)
 *# ---------------------------------------------------------------------
 # store sessions and tickets there
 # ---------------------------------------------------------------------* session.connect(request, response, db=db)
 *# ---------------------------------------------------------------------
 # or store session in Memcache, Redis, etc.
 # from gluon.contrib.memdb import MEMDB
 # from google.appengine.api.memcache import Client
 # session.connect(request, response, db = MEMDB(Client()))
 # ---------------------------------------------------------------------

# -------------------------------------------------------------------------
# by default give a view/generic.extension to all actions from localhost
# none otherwise. a pattern can be 'controller/function.extension'
# -------------------------------------------------------------------------*response.generic\_patterns = [**'\*'**] **if** request.is\_local **else** []
*# -------------------------------------------------------------------------
# choose a style for forms
# -------------------------------------------------------------------------*response.formstyle = myconf.get(**'forms.formstyle'**) *# or 'bootstrap3\_stacked' or 'bootstrap2' or other*response.form\_label\_separator = myconf.get(**'forms.separator'**) **or ''***# -------------------------------------------------------------------------
# (optional) optimize handling of static files
# -------------------------------------------------------------------------
# response.optimize\_css = 'concat,minify,inline'
# response.optimize\_js = 'concat,minify,inline'

# -------------------------------------------------------------------------
# (optional) static assets folder versioning
# -------------------------------------------------------------------------
# response.static\_version = '0.0.0'

# -------------------------------------------------------------------------
# Here is sample code if you need for
# - email capabilities
# - authentication (registration, login, logout, ... )
# - authorization (role based authorization)
# - services (xml, csv, json, xmlrpc, jsonrpc, amf, rss)
# - old style crud actions
# (more options discussed in gluon/tools.py)
# -------------------------------------------------------------------------***from** gluon.tools **import** Auth, Service, PluginManager

*# host names must be a list of allowed host names (glob syntax allowed)*auth = Auth(db, host\_names=myconf.get(**'host.names'**))
service = Service()
plugins = PluginManager()

*# -------------------------------------------------------------------------
# create all tables needed by auth if not custom tables
# -------------------------------------------------------------------------*auth.define\_tables(username=False, signature=False)

*# -------------------------------------------------------------------------
# configure email
# -------------------------------------------------------------------------*mail = auth.settings.mailer
mail.settings.server = **'logging' if** request.is\_local **else** myconf.get(**'smtp.server'**)
mail.settings.sender = myconf.get(**'smtp.sender'**)
mail.settings.login = myconf.get(**'smtp.login'**)
mail.settings.tls = myconf.get(**'smtp.tls'**) **or** False
mail.settings.ssl = myconf.get(**'smtp.ssl'**) **or** False

*# -------------------------------------------------------------------------
# configure auth policy
# -------------------------------------------------------------------------*auth.settings.registration\_requires\_verification = False
auth.settings.registration\_requires\_approval = False
auth.settings.reset\_password\_requires\_verification = True

*# -------------------------------------------------------------------------
# Define your tables below (or better in another model file) for example
#
# >>> db.define\_table('mytable', Field('myfield', 'string'))
#
# Fields can be 'string','text','password','integer','double','boolean'
# 'date','time','datetime','blob','upload', 'reference TABLENAME'
# There is an implicit 'id integer autoincrement' field
# Consult manual for more options, validators, etc.
#
# More API examples for controllers:
#
# >>> db.mytable.insert(myfield='value')
# >>> rows = db(db.mytable.myfield == 'value').select(db.mytable.ALL)
# >>> for row in rows: print row.id, row.myfield
# -------------------------------------------------------------------------

# -------------------------------------------------------------------------
# after defining tables, uncomment below to enable auditing
# -------------------------------------------------------------------------
# auth.enable\_record\_versioning(db)*db.define\_table(**'medicines'**,
 Field(**'medicine\_name'**, **'string'**, length=40, required=True),
 Field(**'dosage'**, **'string'**, length=40, required=True),
 format = **'%(medicine\_name)s'**)

db.define\_table(**'bristol\_scales'**,
 Field(**'b\_scale'**, **'string'**, length=60, required=True),
 format=**'%(b\_scale)s'**,
 )

db.define\_table(**'event\_types'**,
 Field(**'e\_type'**, **'string'**, length=40, required=True),
 format=**'%(e\_type)s'**,
 )

db.define\_table(**'event\_levels'**,
 Field(**'e\_level'**, **'string'**, length=40, required=True),
 format=**'%(e\_level)s'**,
 )

db.define\_table(**'durations'**,
 Field(**'duration\_time'**, **'string'**, length=10, required=True),
 format=**'%(duration\_time)s'**,
 )

db.define\_table(**'events'**,
 Field(**'event\_time'**, **'datetime'**, default = request.now, update = request.now,
 requires=IS\_DATETIME(format=(**'%m-%d-%Y %H:%M'**), timezone = pytz.timezone(timezone\_user))),
 Field(**'e\_type'**, **'reference event\_types'**,
 requires=IS\_EMPTY\_OR(IS\_IN\_DB(db, db.event\_types.id, **'%(e\_type)s'**)),
 represent=**lambda** v, r: **'' if** v **is** None **else** v.e\_type),
 Field(**'e\_level'**, **'reference event\_levels'**,
 requires=IS\_EMPTY\_OR(IS\_IN\_DB(db, db.event\_levels.id, **'%(e\_level)s'**)),
 represent=**lambda** v, r: **'' if** v **is** None **else** v.e\_level),
 Field(**'b\_scale'**, **'reference bristol\_scales'**,
 requires=IS\_EMPTY\_OR(IS\_IN\_DB(db, db.bristol\_scales.id, **'%(b\_scale)s'**)),
 represent=**lambda** v, r: **'' if** v **is** None **else** v.b\_scale),
 Field(**'systolic'**, **'integer'**, length=3, required=False, represent=**lambda** v, r: **'' if** v **is** 0 **else** v),
 Field(**'diastolic'**, **'integer'**, length=3, required=False, represent=**lambda** v, r: **'' if** v **is** 0 **else** v),
 Field(**'pulse'**, **'integer'**, length=3, required=False, represent=**lambda** v, r: **'' if** v **is** 0 **else** v),
 Field(**'medicine\_name'**, **'reference medicines'**, requires=IS\_EMPTY\_OR(IS\_IN\_DB(db, db.medicines.id, **'%(medicine\_name)s'**)),
 represent=**lambda** v, r: **'' if** v **is** None **else** v.medicine\_name),
 Field(**'lbs'**, **'double'**, required=False, represent=**lambda** v, r: **'' if** v **is** 0.00 **else** v),
 Field(**'duration\_time'**, **'reference durations'**, requires=IS\_EMPTY\_OR(IS\_IN\_DB(db, db.durations.id, **'%(duration\_time)s'**)),
 represent=**lambda** v, r: **'' if** v **is** None **else** v.duration\_time),
 Field(**'note'**, **'text'**, required=False, default=**''**, represent=**lambda** v, r: **'' if** v **is** None **else** v)
 )

db.medicines.id.readable = False
db.bristol\_scales.id.readable = False
db.event\_types.id.readable = False
db.event\_levels.id.readable = False
db.durations.id.readable = False
db.events.id.readable = False