```
import cgi
import datetime
import os
import sqlite3

from genshi.template import TemplateLoader
from genshi.template.base import Context

BASE_PATH = '/home/dsorber/bfcs'
DB_PATH = os.path.join(BASE_PATH, 'beer_temps.db')
TEMPLATE_PATH = os.path.join(BASE_PATH, 'templates')

# It accepts two arguments:
# environ points to a dictionary containing CGI like environment variables
# which is filled by the server for each received request from the client
# start_response is a callback function supplied by the server
# which will be used to send the HTTP status and headers to the server

def application(environ, start_response):

    loader = TemplateLoader(TEMPLATE_PATH, auto_reload=True)

    if not environ['PATH_INFO'] or environ['PATH_INFO'] == '/':
        # Load the index template
        template = loader.load('index.html')

        # Create the response args
        response_args = {'title': 'Beer Fermentation Control System'}

        # Open up the connection to the database, execute the query,
        # grab the results, then close the DB connection.
        db_conn = sqlite3.connect(DB_PATH)
        db = db_conn.cursor()
        db.execute("select strftime('%m/%d/%Y', date(ts, 'unixepoch')), "
                   "       count(ts), avg(temp1), max(temp1), min(temp1),"
                   "       avg(temp2), max(temp2), min(temp2), avg(temp3),"
                   "       max(temp3), min(temp3), avg(temp4), max(temp4),"
                   "       min(temp4)"
                   "from temp_data "
                   "group by date(ts, 'unixepoch')")
        response_args['results'] = db.fetchall()
        db_conn.close()

        # Generate the stream and then render it to HTML
        stream = template.generate(**response_args)
        response_body = stream.render('html', doctype='html', encoding='utf-8')

        # Set the appropriate response headers
        response_headers = [('Content-Type', 'text/html; charset=utf-8'),
                            ('Content-Length', str(len(response_body)))]

    elif environ['PATH_INFO'].startswith('/day/') and environ['PATH_INFO'].endswith('/all'):
        # Load the all template
        template = loader.load('all.html')

        # Parse the date out of the URL (TODO: validate all this)
        day = environ['PATH_INFO'][5:7]
        month = environ['PATH_INFO'][7:9]
        year = environ['PATH_INFO'][9:13]
        date = '%s/%s/%s' % (month, day, year)

        # Create the response args
        response_args = {'title': 'Sensor Readings for %s' % date}

        # Open up the connection to the database, execute the query
        # (limit to today and the past 5 minutes),
        # grab the results, then close the DB connection.
        db_conn = sqlite3.connect(DB_PATH)
        db = db_conn.cursor()
        db.execute("select strftime('%%H:%%M:%%S', ts, 'unixepoch'), "
                   "       temp1, temp2, temp3, temp4, relay_status "
                   "from temp_data "
                   "where date(ts, 'unixepoch') is date('%s-%s-%s')"
                   "order by ts desc" % (year, month, day))
        response_args['results'] = db.fetchall()

        db_conn.close()

        # Generate the stream and then render it to HTML
```

```
        stream = template.generate(**response_args)
        response_body = stream.render('html', doctype='html', encoding='utf-8')

        # Set the appropriate response headers
        response_headers = [('Content-Type', 'text/html; charset=utf-8'),
                            ('Content-Length', str(len(response_body)))]

    elif environ['PATH_INFO'].startswith('/day/'):
        # Load the day template
        template = loader.load('day.html')

        # Parse the date out of the URL (TODO: validate all this)
        day = environ['PATH_INFO'][5:7]
        month = environ['PATH_INFO'][7:9]
        year = environ['PATH_INFO'][9:13]
        date = '%s/%s/%s' % (month, day, year)

        # Create the response args
        response_args = {'title': 'Sensor Readings for %s' % date,
                         'url_date': environ['PATH_INFO'][5:13]}

        # Open up the connection to the database, execute the query
        # (limit to today and the past 5 minutes),
        # grab the results, then close the DB connection.
        db_conn = sqlite3.connect(DB_PATH)
        db = db_conn.cursor()
        db.execute("select strftime('%%H:%%M:%%S', ts, 'unixepoch'), "
                   "       temp1, temp2, temp3, temp4, relay_status "
                   "from temp_data "
                   "where date(ts, 'unixepoch') is date('%s-%s-%s')"
                   "order by ts desc limit 300" % (year, month, day))
        response_args['results'] = db.fetchall()

        # Chart query
        #
        db.execute("select strftime('%%H:%%M:%%S', ts, 'unixepoch'), ts, temp1, "
                   "       temp2, temp3, temp4 from ( "
                   "select ts, temp1, temp2, temp3, temp4 "
                   "from temp_data "
                   "where date(ts, 'unixepoch') is date('%s-%s-%s')"
                   "order by ts desc limit 60) order by ts"
                   % (year, month, day))
        chart_results = db.fetchall()

        # Process chart query results
        sensor1_data = []
        sensor2_data = []
        sensor3_data = []
        sensor4_data = []
        for row in chart_results:
            sensor1_data.append([int(row[1]) * 1000, row[2]])
            sensor2_data.append([int(row[1]) * 1000, row[3]])
            sensor3_data.append([int(row[1]) * 1000, row[4]])
            sensor4_data.append([int(row[1]) * 1000, row[5]])
        response_args['sensor1_data'] = str(sensor1_data)
        response_args['sensor2_data'] = str(sensor2_data)
        response_args['sensor3_data'] = str(sensor3_data)
        response_args['sensor4_data'] = str(sensor4_data)

        # This is a bit of hack to get around a javascript issue in
        # the template
        # response_args['chart_results'] = chart_results[:-1]
        # response_args['chart_last_row'] = chart_results[-1]

        db_conn.close()

        # Generate the stream and then render it to HTML
        stream = template.generate(**response_args)
        response_body = stream.render('html', doctype='html', encoding='utf-8')

        # Set the appropriate response headers
        response_headers = [('Content-Type', 'text/html; charset=utf-8'),
                            ('Content-Length', str(len(response_body)))]

    elif environ['PATH_INFO'] == ('/chart_update'):
        # Update the self updating chart
        response_body = ''

        if environ['QUERY_STRING']:
```

```
        # Parse the query string to get the last timestamp from the client
        get_args = cgi.parse_qs(environ['QUERY_STRING'])
        last_ts = int(get_args['last_ts'][0])
        date_from_ts = datetime.datetime.fromtimestamp(last_ts)

        # Reconstruct portions of the date from the ts
        year = date_from_ts.year
        month = '%02d' % date_from_ts.month
        day = '%02d' % date_from_ts.day

        # Grab any entries from the supplied date that are greater than
        # the last timestamp sent from the client javascript
        db_conn = sqlite3.connect(DB_PATH)
        db = db_conn.cursor()
        db.execute("select strftime('%%H:%%M:%%S', ts, 'unixepoch'), ts, "
                   "       temp1, temp2, temp3, temp4, relay_status "
                   "from temp_data "
                   "where date(ts, 'unixepoch') is date('%s-%s-%s') and "
                   "      ts > %s "
                   "order by ts asc limit 60" % (year, month, day, last_ts))
        db_rows = db.fetchall()
        db_conn.close()

        # Iterate over the rows fetched by the query and pretty them up
        # in preparation of being sent back to the client javascript for
        # display
        result_rows = []
        for row in db_rows:
            result_rows.append('%s %s %s %s %s %s %s' %
                               (row[0], (int(row[1]) * 1000),
                               row[2], row[3], row[4], row[5],
                               int(row[6]) and 'on' or 'off'))

        response_body = ','.join(result_rows)
        response_body = response_body.encode('utf-8')


    response_headers = [('Content-Type', 'text/plain; charset=utf-8'),
                        ('Content-Length', str(len(response_body)))]

else:
    # User supplied an invalid URL
    response_body = 'Invalid URL "%s"' % environ['PATH_INFO']
    response_headers = [('Content-Type', 'text/plain'),
                        ('Content-Length', str(len(response_body)))]

# HTTP response code and message
status = '200 OK'

# Send the status and response headers to the server
start_response(status, response_headers)

# Return the response body. (Notice it is wrapped in a list although it
# could be any iterable.)
return [response_body]
```