ECM3408 - CA1: Skydrive

62007094

February 15, 2015

1 SQLite Database Creation

ew The following commands can be used to create a table called `documents`

```
$ sqlite3 skydrive.sqlite3
sqlite > CREATE TABLE documents
    ...> ( id INT PRIMARY KEY
    ...> , name VARCHAR
    ...> , message VARCHAR ); # sqlite does not impose length-limits
        on fields, so none is given.
.quit
```

1.1 Testing

Script output of creating a database as described above, storing an unencrypted message and retrieving it:

```
bash-3.2$ sqlite3 skydrive.sqlite3
SQLite version 3.8.5 2014-08-15 22:37:57
Enter ".help" for usage hints.
sqlite> CREATE TABLE documents
   ...> ( id INT PRIMARY KEY
   ...> , name VARCHAR
   ...> , message VARCHAR );
sqlite> INSERT INTO documents VALUES
   ...> (1, "Name", "Message");
sqlite> SELECT * FROM documents;
1|Name|Message
sqlite>
```

2 The Ruby Code

```
#!/usr/bin/env ruby -w
require "sqlite3"
require "webrick"
DATABASE = "skydrive.sqlite3"

# -- MODELS --

##
# Model for the Index: Gets a list of all documents and their IDs for the
    view
def model_index()
```

```
    db  = SQLite3::Database.new( DATABASE )
    qry = "SELECT id, name FROM documents;"
    hash = db.execute( qry )
    db.close
    return hash
end

##
# Model for showing an individual message
# Gets the message via id, and decrypts with the cipher given
def model_show(id, shift)
    db  = SQLite3::Database.new( DATABASE )
    qry = "SELECT message FROM documents "                          +
              "WHERE id = \"#{id}\";"
    val = db.get_first_value( qry )
    cipher = Caesar.new shift
    message = cipher.decrypt(val)
    db.close
    return message
end

##
# Model to update an entry
# Depending on whether an entry has just been updated, an entry has just been
# selected to be updated or no entry has been selected, yet, the appropriate
# data is processed

def model_update(show=false, update=false, id, message, shift)
    db  = SQLite3::Database.new( DATABASE )
    qry = "SELECT id, name FROM documents;"
    hash = db.execute( qry )

    cipher = Caesar.new shift.to_i

    ##
    # load an entry to be edited
    if show
      qry = "SELECT message FROM documents "                        +
      "WHERE id = \"#{id}\";"
      val = db.get_first_value( qry )
      message_dec = cipher.decrypt(val)
      db.close
      return false, hash, message_dec, id

    ##
    # update an entry with the text already entered
    elsif update
      message_enc = cipher.encrypt(message)
      puts message_enc
      puts shift
      puts id
```

```ruby
    qry= "UPDATE documents SET message=\"#{message_enc}\""      +
         "WHERE id=\"#{id}\";"
    db.execute( qry )
    return true, hash

  ##
  # just show the entries available for edit
  else
    return false, hash
  end
end

##
# Model for creating a new entry. Depending on whether data has already been
# entered or not, the appropriate data is processed

def model_new(process=false, message, name, shift)
  ##
  # if data has been entered to save
  if process
    cipher = Caesar.new shift.to_i
    message_enc = cipher.encrypt(message)

    db  = SQLite3::Database.new( DATABASE )
    qry = "SELECT id FROM documents ORDER BY id DESC LIMIT 1;"
    id = db.execute( qry ).join.to_i + 1
    qry = "INSERT INTO documents VALUES"                         +
        "(#{id}, \"#{name}\", \"#{message_enc}\");"
    db.execute( qry )
    db.close
    return true
  end
  return false
end


##
# Model for destroying an existing entry

def model_destroy(process=false, id)
  db  = SQLite3::Database.new( DATABASE )

  ##
  # if an entry has been selected to be destroyed
  if process
    qry = "DELETE FROM documents WHERE id=#{id};"
    db.execute (qry)
    qry = "SELECT id, name FROM documents;"
    hash = db.execute( qry )
    return true, hash
  end

```

```
    ##
    # load a list of entries for selection
    qry = "SELECT id, name FROM documents;"
    hash = db.execute( qry )

    db.close
    return false, hash
end

# -- VIEWS --

def view_show(val)
    "<html>"                                                    +
    "  <body>"                                                  +
    "    <p>" + val.to_s + "</p>"                               +
    "  </body>"                                                 +
    "</html>"
end

def view_index(vals)
    output = "<html>"                                           +
    "  <body>"                                                  +
    "    <form action=\"http://localhost:3000/show\""           +
    "    method=\"GET\">"                                       +
    "      <select name=\"id\">"

    ##
    # for each value, show name and have id as form-value
    vals.each do |key, value|
          output << "<option value=\"#{key}\">#{value}</option><br \>"
    end

    output << "</select>"                                       +
    "      <input name=\"shift\" value=\"Enter shift\"/>"       +
    "      <input type=\"Submit\"/>"                            +
    "    </form>"                                               +
    "  </body>"                                                 +
    "</html>"
end

def view_new(success=false)
    output = "<html>"                                           +
    "  <body>"

    if success
        output << "<p>Message encrypted and added</p>"
    end

    output << "    <form action=\"http://localhost:3000/new\""  +
    "    method=\"GET\">"                                       +
    "      <input name=\"shift\" value=\"Enter Shift\"/>"       +
    "      <input name=\"name\" value=\"Enter Name\"/>"         +
    "      <input name=\"message\" value=\"Enter Message\"/>"   +
```

```ruby
    "       <input type=\"Submit\"/>"                                    +
    "     </form>"                                                       +
    "   </body>"                                                         +
    "</html>"

    return output
end

def view_destroy(deleted=false, vals)
    output = "<html>"                                                    +
    "   <body>"

    if deleted
        output << "<p>Message deleted</p>"
    end

    output << "     <form action=\"http://localhost:3000/destroy\"" +
    "      method=\"GET\">"                                              +
    "        <select name=\"id\">"

    ##
    # for each value, show name and have id as form-value
    vals.each do |key, value|
      output << "<option value=\"#{key}\">#{value}</option><br \>"
    end

    output << "</select>"                                                +
    "       <input type=\"Submit\"/>"                                    +
    "     </form>"                                                       +
    "   </body>"                                                         +
    "</html>"

    return output
end

def view_update(updated, vals, msg, id, shift)
    output = "<html>"                                                    +
    "   <body>"

    if updated
        output << "<p>Message updated</p>"
    end

    ##
    # if the user requested to update an entry, the model will provide the
    # decrypted message to edit. This is checking if a message has been
    # transmitted and displays the edit form if that is the case
    if defined? msg
      output << "     <form action=\"http://localhost:3000/update\""+
      "      method=\"GET\">"                                            +
      "        <input type=\"hidden\" name=\"id\" value=\"#{id}\"/>" +
      "     <input type=\"hidden\" name=\"shift\" value=\"#{shift}\"/>"
         +
      "     <input name=\"message\" value=\"#{msg}\"/>"          +
      "     <input type=\"Submit\"/>"                            +
      "    </form>"
    end

    output << "    <form action=\"http://localhost:3000/update\"" +
    "     method=\"GET\">"                                         +
    "       <select name=\"id\">"

    ##
    # show all entries that can be edited
    vals.each do |key, value|
      output << "<option value=\"#{key}\">#{value}</option><br \>"
    end

    output << "</select>"                                          +
    "       <input name=\"shift\" value=\"Enter shift\"/>"         +
    "       <input type=\"Submit\"/>"                              +
    "     </form>"                                                 +
    "   </body>"                                                   +
    "</html>"

    return output
end


# -- CONTROLLER --

class Controller < WEBrick::HTTPServlet::AbstractServlet
    def do_GET ( req, rsp )
      ##
      # Decide on which MV by analysing the request
      case req.path
        ##
        # Index: Overview of all messaged
        when "/index"
          rsp.status = 200
          rsp.content_type = "text/html"
          rsp.body = view_index( model_index() )

        ##
        # Add new message
        when "/new"
          message = req.query[ "message" ] || ""
          name = req.query[ "name" ] || ""
          shift = req.query[ "shift" ] || ""

          ##
          # Check if something has been submitted for processing
          if message.length == 0 || name.length == 0 || shift.length == 0
            process = false
          else
            process = true
          end

          rsp.status = 200
          rsp.content_type = "text/html"
          rsp.body = view_new( model_new(process, message, name, shift) )

        ##
        # Showing an entry
        when "/show"
          id = req.query[ "id" ] || ""
          shift = req.query[ "shift" ] || ""
          rsp.status = 200
          rsp.content_type= "text/html"
          rsp.body = view_show( model_show(id, shift.to_i) )

        ##
        # Destroying an entry
        when "/destroy"
          id = req.query[ "id" ] || ""

          ##
          # Check if an entry has been submitted to be deleted
          if id.length == 0
            process = false
          else
            process = true
          end

          rsp.status = 200
          rsp.content_type = "text/html"
          success, vals = model_destroy(process, id)
          rsp.body = view_destroy(success, vals )

        ##
        # Update existing entry
        when "/update"
          message = req.query[ "message" ] || ""
          id = req.query[ "id" ] || ""
          shift = req.query[ "shift" ] || ""

          ##
          # If message already submitted to upgrade old one, call
              appropriately
          if message.length > 0
            update = true
            show = false
          elsif id.length > 0 && shift.length > 0
            update = false
            show = true
          end

```

```
            rsp.status = 200
            rsp.content_type = "text/html"

            updated, vals, msg, id = model_update(show, update,
                                                  id, message, shift)

            rsp.body = view_update(updated, vals, msg, id, shift)
        end
    end
end

# -- SUPPORT CLASSES --

##
# Caesar: Providing the encryption scheme and functions to encrypt
class Caesar

  ##
  # Constructor - takes amount of shift as arguemtn, and uses an alphabet
      of all
  # letter (capital or not), numbers and whitespace by default
  def initialize(shift, alphabet = (('a'..'z').to_a + ('A'..'Z')
                                    .to_a + ('0'..'9').to_a + [' '])
                                    .join)
    ##
    # Put alphabet in array and rotate array by amount of shift if de-/
        encrypt
    # requested
    chars = alphabet.chars.to_a
    @encrypter = Hash[chars.zip(chars.rotate(shift))]
    @decrypter = Hash[chars.zip(chars.rotate(-shift))]
  end

  def encrypt(string)
    @encrypter.values_at(*string.chars).join
  end

  def decrypt(string)
    @decrypter.values_at(*string.chars).join
  end
end

# -- RUNTIME --

server = WEBrick::HTTPServer.new( :Port => 3000 )
   server.mount( "/", Controller )
   server.start
```

3 Testing

Unit	Process	Expected Outcome	Actual Outcome
Index	Call /index	List of all saved messages	As Expected
Index/Show	Select a message on /index, enter the shift and submit	Opens /show with decrypted message. Shows only scrambled message with the wrong key provided	As Expected
New	Call /new	Option to enter new message with name, shift and message body	As Expected
Add New	Submit new entry and have it written to the database	Submit form at /new with filled out forms andante confirmation of entry been written, and have it display on /index and read it on /show with the shift originally provided. Scrambled output if providing any other shift key	As expected
Destroy	Open /destroy and delete an entry	Get presented with a list of entries, select one for deletion, get confirmation, and not be able to access it anymore on /index	As Expected
Update	Open /update, select an entry to be updated, enter the shift key, edit and save	As process and then check on /index and subsequently on /show whether the change was successful	As Expected

All components tested for functionality and purposes succesfully.