Computer Networks
Fall 2013/14

Lab 2 – Building a Task Manager

Lab2 must be submitted using the course website by **09-1-2014.**
>>> No late submissions will be accepted.

Building a Task Manager

Goals:

- Building an SMTP client
- Building a task manager that provides the following services:
  - E-mail reminder
  - Add and assign task + reminder of due date
  - Poll a group

Major Milestones:

- Implement an SMTPClient class

- Modify config.ini file so that it includes the following values:
  - **SMTPName** – The SMTP server IP or DNS. **Use compnet.idc.ac.il**
  - **SMTPPort –** The SMTP server port. **Use port 25**
  - **ServerName –** Your server's IP of DNS. **Use localhost**
  - **SMTPUsername – Username for the SMTP server. Use tasker@cscidc.ac.il**
  - **SMTPPassword – Password for the SMTP server. Use "password"**
  - **SMTPIsAuthLogin – "TRUE"/"FALSE". Whether to use AUTH LOGIN during SMTP. Use TRUE.**
  - **reminderFilePath** – path of reminders database file. Use **.\reminder.data**
  - **taskFilePath** – path of tasks database file. **Use .\tasks.data**
  - **pollFilePath** – path of polls database file. **Use .\polls.data**

- Implement an index.html page in order to allow users to log-in with their email address.

- Implement an main.html page to be used as the main page of the application.

- For **reminders** implement the following pages:
  - reminders.html for managing E-Mail reminders
  - reminder_editor.html
  - submit_reminder.html

- For **tasks** implement the following pages:
  - tasks.html for managing tasks
  - task_editor.html

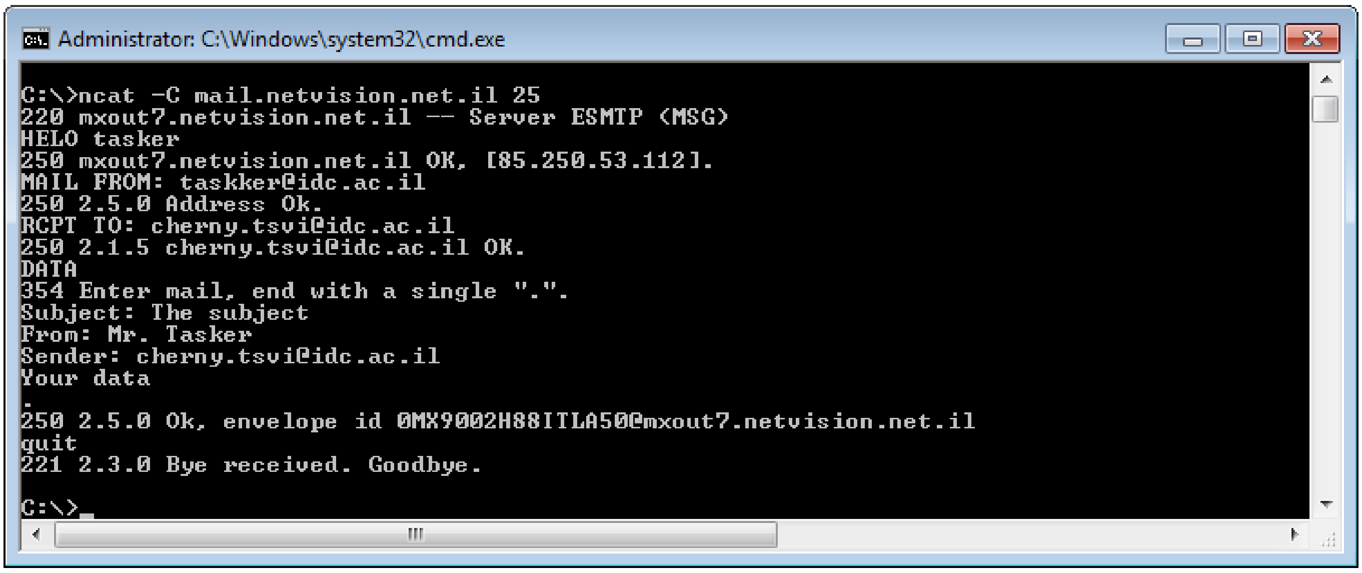
    - submit_task.html
    - task_reply.html

- For **polls** implement the following pages:
    - polls.html for managing polls
    - poll_editor.html
    - submit_poll.html
    - poll_reply.html

Goals in Detail:

1. Implement an SMTPClient class:

Implement an SMTP client class.
The class implements the SMTP protocol so that you will be able to use it in order to send e-mail messages using the **SMTPName** and **SMTPPort as** configured in the config.ini file.



Administrator: C:\Windows\system32\cmd.exe

```
C:\>ncat -C mail.netvision.net.il 25
220 mxout7.netvision.net.il -- Server ESMTP (MSG)
HELO tasker
250 mxout7.netvision.net.il OK, [85.250.53.112].
MAIL FROM: taskker@idc.ac.il
250 2.5.0 Address Ok.
RCPT TO: cherny.tsvi@idc.ac.il
250 2.1.5 cherny.tsvi@idc.ac.il OK.
DATA
354 Enter mail, end with a single ".".
Subject: The subject
From: Mr. Tasker
Sender: cherny.tsvi@idc.ac.il
Your data
.
250 2.5.0 Ok, envelope id 0MX9002H88ITLA50@mxout7.netvision.net.il
quit
221 2.3.0 Bye received. Goodbye.

C:\>_
```

In case SMTPIsAuthLogin is TRUE, use AUTH LOGIN to login the SMTP server before sending the mail (before MAIL FROM).
To use AUTH LOGIN first you will have to connect to the server using extended SMTP instead of regular SMTP, this is done by starting the dialogue with EHLO instead of HELO command. Once EHLO has been accepted send “AUTH LOGIN” to the server. Next, send the SMTPUsername in base64 encoding, and then send SMTPPassword in base64 encoding. Once you’ve done both, the server will accept or reject your credentials. After you have logged in successfully, continue as usual.

Important points:

- When sending EHLO in your client name, write: “FirstName1LastName1 FirstName2LastName2”.
  - For example for “AliceInWonderland” and “Bob Sponge” their EHLO would look like: “`EHLO AliceInWonderland BobSponge`”
- For Base64 implementation you can use the Java built-in classes or any code you can find on the net.
- The name of the sender must be “Mr. Tasker”

2. Implement index.html to allow users to log-in with their mail address

Once a client surfs to **any page** in the web application, the cookie in the http request is being checked. If there is no cookie header, or the cookie’s format is not of identical to the format “**usermail-[mail address]**”, then the response is an HTTP redirect (as we’ve seen in the recitation) to the default page “index.html”.

An http request to index.html checks if there is a valid cookie:

- If valid – HTTP redirects to “main.html”.
- If not valid:
  - The server shows a textbox with a submit button in the page index.html
  - If request to index.html contains a mail to log-in with, the server responds with set-cookie and HTTP redirect to “main.html”.

3. Implement main.html as main page for the application

The main.html page of the application contains:

- A log out button/link which sets the cookie to an empty string and then surfs the page index.html.
- A link to the “E-Mail reminders” page: reminders.html
- A link to the “Tasks” page: tasks.html
- A link to the “Polls” page: polls.html

4. Implement reminders

Reminders are e-mail messages being sent to a user at a certain time.
The reminders database file is the file **reminderFilePath** sets in the config.ini

Once the time to send a specific reminder arrives, the application needs to send the reminder’s owner a reminder via e-mail by using your SMTP client class.

Important: During application start-up, if there is a reminder that its time has expired (i.e. passed), this reminder is being sent immediately.

- reminders.html:
  - The page displays all the reminders of the logged-in user (**and no other reminders**)
  - A “new reminder” button/link that points to the reminder_editor.html page
  - A “home” button/link that points to the main.html page
  - Each reminder record is a line which is composed of:

        - A reminder title.
        - The date and time of creation.
        - The date and time of reminding
        - An “Edit” button/link that points to the reminder_editor.html page, which fills automatically the reminder’s data into the controls within that page.
        - A “Delete” button/link that deletes the reminder and surfs back to the reminders.html page (refreshes the page).

- reminder_editor.html:
    - This page allows a user to create/modify a specific reminder
    - When editing an existing reminder, the reminder’s line is updated, otherwise, in the case of a new reminder, a new reminder line is created.
    - This page contains:
        - A textbox for subject.
        - A textarea for reminder’s content.
        - A textbox for the date of the reminder (format: DD/MM/YYYY).
        - A textbox for the time of the reminder (format: HH:MM).
        - A save button/link which creates/modifies a reminder record and then surfs to submit_reminder.html in order to display the result of the operation.
        - A cancel button/link which cancels the operation and surfs back to reminders.html.

- submit_reminder.html:
    - This page displays the success/failure of creating/modifying a reminder.
    - In case of a success, the http request of the page returns HTTP redirect response to the reminders.html page
    - In case of failure, the page clearly states that there was a failure and displays an error message
        - The page also contains a “go back” link to return to the page reminders.html

Important points to consider:

- What do you store in the database? For instance:
    - How do you identify a reminder?
    - How do you know which reminder belongs to which user?
- What method do you use in order store in the database? For instance:
    - XML based database (you can use java XML objects, or any other classes you find)
- Which data structure is used to store the database in memory?
- How do you know when to send a reminder (reminders can be changed or deleted)?

5. Implement tasks

A task is an e-mail message being sent from the logged-in user to a recipient with a task body and a due-date. The task e-mail message contains a link to the server that marks the task as completed, and lets the task recipient knows that the task has been closed.
If the due date arrives and the task is still not completed, an e-mail message is being sent to the task creator and the task recipient that time is due.

The tasks database file is the **taskFilePath** which is set in the config.ini file.

Important: During application start-up, if there is a task that its time is due, treat the task as if it is not completed.

- tasks.html:
    - This page displays all the tasks of the logged-in user (**and no other tasks**)
    - A "new task" button/link that points to task_editor.html
    - A "home" button/link that points to main.html
    - Each task is a line which is composed of:
        - Task title
        - Date and time of creation
        - Date and time of due date
        - Status text:
            - in progress…
            - completed
            - time is due
        - "Delete" button/link that deletes the task and surf back to the tasks.html page (refreshes the page).
            - Only "in progress…" tasks can be deleted

- task_editor.html:
    - This page allows a user to create a new task
    - This page contains:
        - Textbox for subject
        - Textarea for task content
        - Textbox for recipient
        - Textbox for date of task due date (format: DD/MM/YYYY)
        - Textbox for time of task due date (format: HH:MM)
        - Send button/link which creates a task and surfs to the submit_task.html page in order to display the result of the operation
        - Cancel button/link which cancels the operation and surfs back to tasks.html

- submit_task.html:
    - This page displays the success/failure of creating a task
    - In case of success, the http request of the page returns HTTP redirect response to tasks.html
        - Task status is "in progress…"
    - In case of failure, the page clearly states that there was a failure and displays an error message
        - The page also contains a "go back" link to return to the tasks.html page

- E-Mail at recipient:
    - The mail subject always begins with "Task:"
    - A link inside the email message which marks the task as completed is a link that points to the page "task_reply.html" at the server.

- In order to generate the full link you needs the server name which is written as **ServerName** in the config.ini file.

- Due date:
  - Send mail to the creator of the task and to the task recipient to let them know that the task time is due
  - Update the task status.

- task_reply.html:
  - Update the status of the task to "completed"
  - Send e-mail message to the task creator to let him/her know that the task has been completed

Important points to consider:

- Notice that the classes you build for the **reminder** part may be reused for the task part!

6. Implement polls

Poll is an e-mail message being sent to a **group** of recipients with a question and several possible answers. The recipients answer the question by using a link which appears in the e-mail message.

Every answer by a recipient causes the application to send an e-mail message to the poll initiator with the following summary:

- The poll question;
  The recipient answer.

The polls' database file is the **pollFilePath** file as is set in the config.ini file.

- polls.html:
  - The page displays all the polls of the logged-in user (**and no others**)
  - A "new poll" button/link that points to the poll_editor.html page
  - A "home" button/link that points to the main.html page
  - Each poll is a line which is composed of:
    - Poll title
    - Date and time of creation
    - Recipients replies
    - "Delete" button/link that deletes the task and surf back to the polls.html page (refreshes the page).

- poll_editor.html:
  - This page allows a user to create a new poll
  - The page contains:
    - Textbox for subject
    - Textarea for poll content
    - Textarea for recipients – delimited by CRLF
    - Textarea for answers – delimited by CRLF
    - Send button/link which creates the poll and surfs to the submit_poll.html page in order to display the result of the operation

        - Cancel button/link which cancels the operation and surfs back to polls.html

- submit_poll.html:
    - This page displays success/failure of creating a poll
    - In case of success, the request for the page returns HTTP redirect response to polls.html
    - In case of failure, the page clearly state that there was a failure and displays an error message
        - The page also contains a "go back" link in order to return to polls.html

- E-Mail at recipient:
    - The mail subject always begins with "Poll:"
    - Each answer is a link to the page "poll_reply.html" at the server. Surfing to the answer causes the task to be marked as completed.

- poll_reply.html:
    - Updates the status of the poll to: "completed"
    - Sends an e-mail message to the poll initiator that the recipients have answered the poll and summarize the current state
    - In case a recipient marks an already closed poll (because the due time has passed, or it is already marked as completed or deleted) the message "Poll has been closed" should be displayed to the user".

Important points to consider:

- Notice that the classes you have used for the tasks part and the reminders part can be used here!

Exception Handling:

The program must not crash!

Use exception handling to recover. If you cannot recover, print what happened to the console and close the application gracefully.

Notice that possibility of crashing will drop **many** points!

Traces (a.k.a. print to console):

The following traces are **required**:

- Listening port (on startup)
- The HTTP requests arriving to the server
- The HTTP response header returning to the browser
- **The SMTP dialogue**

You are **encouraged** to use additional traces in your code!
Traces will help you debug your application and perform a better QA.

Make sure your traces actually mean something that will help to understand better what is going on during runtime. **That can and will save you a lot of time!**

Bonus:

Feel free to add more functionality to the application as you see fit!
Good ideas will be **rewarded** with points and world-wide fame (score may be over a 100)!!!

Please write and explain all the implemented bonuses in a file named bonus.txt.
Notice that bonuses that will not be mentioned in bonus.txt will not be checked, that is, they will be ignored!

Important: We don't guarantee in advance that for extra-functionality, extra-points will be rewarded. Also, the idea is to ***add*** a new functionality ***not replace a requested functionality*** – that will lead to dropping of points! In other words, do not ignore things we require, and implement other things instead and call it a bonus! We cannot make it any clearer than that!

Discussion Board:

Please, use the forum of the course to ask questions when something is unclear.

You must make sure that you check the discussion board. If there are unclear matters about Lab 1 that were raised in the discussion board, **our answers are mandatory**, and apply to **everyone**!

You can register to receive e-mail notification from the forum.

A word from your checker:

- **Do not** create your own packages, all your classes **must be in the same default package**, and compile while all sources in the same directory.
- If you are coding using a mac, make sure you remove all hidden directories generated by your IDE.
- You can implement the lab using up to JDK1.6 .
- The checking of your code will be done from console, make sure your server works without IDE before submission!

Remember - A happy checker is a merciful checker!

Submission:

- You **must** submit a batch file called "compile.bat" - The batch file compiles your code successfully. If the batch fails to compile, **many** points will be dropped.
- You **must** submit a batch file called "run.bat" - The batch file starts your server. If the batch fails to start the server, **many** points will be dropped.

Submit the lab following the course submission rules.
The submitted zip should contain a folder named ID1_ID2 which contains:

- Sources (*.java)
- config.ini
- compile.bat
- run.bat
- Directory called "serverroot" containing your non-code files (your HTMLs + extra files/images etc.)
- bonus.txt with the bonuses you implemented – if applicable.
- **readme.txt** that explains what exactly you have implemented and the role of each class in your server.

Notice: not submitting one of the above requested files (exclude bonus.txt) will lead to dropping of points!

That's it ☺ !
Good Luck!