ICT SBA Report

Shen Lik Hang 6E 28

November 2024

Sing Yin Secondary School

Contents

1 Introduction

1.1 Problem statement

You are a developer of the SY Task application, which allows students to add, list and manage tasks for their daily school life, such as doing homework and attending activities. The system should allow users to:

```
add tasks,
list all tasks,
list uncompleted tasks,
set tasks as completed,
update and delete tasks,
optionally include the due date and task category.
```

1.2 Indication of methodology

As I have no experience making a web application project before, I have asked Perplexity AI for suggestion on the procedure regarding the steps to make a project, and it gave me this response:

1. Requirements Finding
    - Collect detailed requirements from potential students to understand their needs for task management
2. System analysis
    - Analyze the gathered requirements to define the system's functionalities clearly
3. Design Phase
    - Design the architecture of the application based on the requirements and analysis
4. Development
    - Implement the application according to the design specifications
5. Testing
    - Validate that the application works as intended and meets user requirements
6. Deployment
    - Launch the application for use by students
7. Maintenance
    - Ensure ongoing support and improvements post-deployment

1.3 Main findings

1. Users always forget to complete tasks
2. Users want to organize different tasks efficiently
3. Users want to prioritize tasks
4. Users want to have a user-friendly interface instead of command-line interface
5. Users want to record how many tasks they have completed
6. Users want to collaborate with other users to manage different tasks
7. Users want to have a privacy protected task application
8. Users want to analyze their productivity over a time period
9. Users want to have an AI assistant which provides suggestions on task management
10. Users want the data can be automatically synchronized

1.4 Principal conclusion

An application with the following features is needed:

1. Support create, read, update, and delete (CRUD)
2. Notification for coming tasks
3. Filtering tasks
4. Collaboration between accounts
5. Make analysis of productivity
6. AI Chat-bot
7. Support synchronization when launching between different devices

2 Literature review

2.1 Background

I am a developer of the SY Task application. I am asked to make a task manager application for students. I renamed the application as "Achilles" who was known as being the greatest of all the Greek warriors. He represents the idea of invincibility and sheer power, which I want to use to inspire students to overcome overwhelming workloads without fear.

The reason is the students want to have some improvements on task management through using an application.

Therefore, the students would like to have an application with the features mentioned in **Section 1.4**

I have chosen the application to be web-based for the following reasons:

1. No installation required if web browser is installed
2. Web applications can be updated instantly, without requiring users to download and install new versions
3. Developers can push updates directly to the server, streamlining the maintenance process
4. Web applications typically require development for a single platform, reducing development and testing efforts

However, web application has the following limitations over phone application:

1. Offline functionality
2. Device-specific features, such as sensors

2.2 Purpose of the project

The main purpose of the project is

```
to design a system which allows students to add,
list and manage tasks for their daily school life
```

2.3 Requirement Analysis

2.3.1 Functional requirements

Task CRUD Basic requirements for a task manager

Task reminder Notify the users of coming tasks

Task classification Organize different task

Task prioritization Prioritize more important tasks, can be done by add ”Important” tag to tasks

2.3.2 Nonfunctional requirements

Performance Data should be processed fast and accurately

Usability User-friendly and accessible layout should be made

Security Data privacy should be protected

Synchronization Data should be kept consistently over different devices

2.3.3 Additional requirements

AI chat-bot Provide suggestions on the task management such as priority of tasks

Productivity analysis Analysis the productivity over a period of time by counting how much time used to complete tasks

3 Preparation

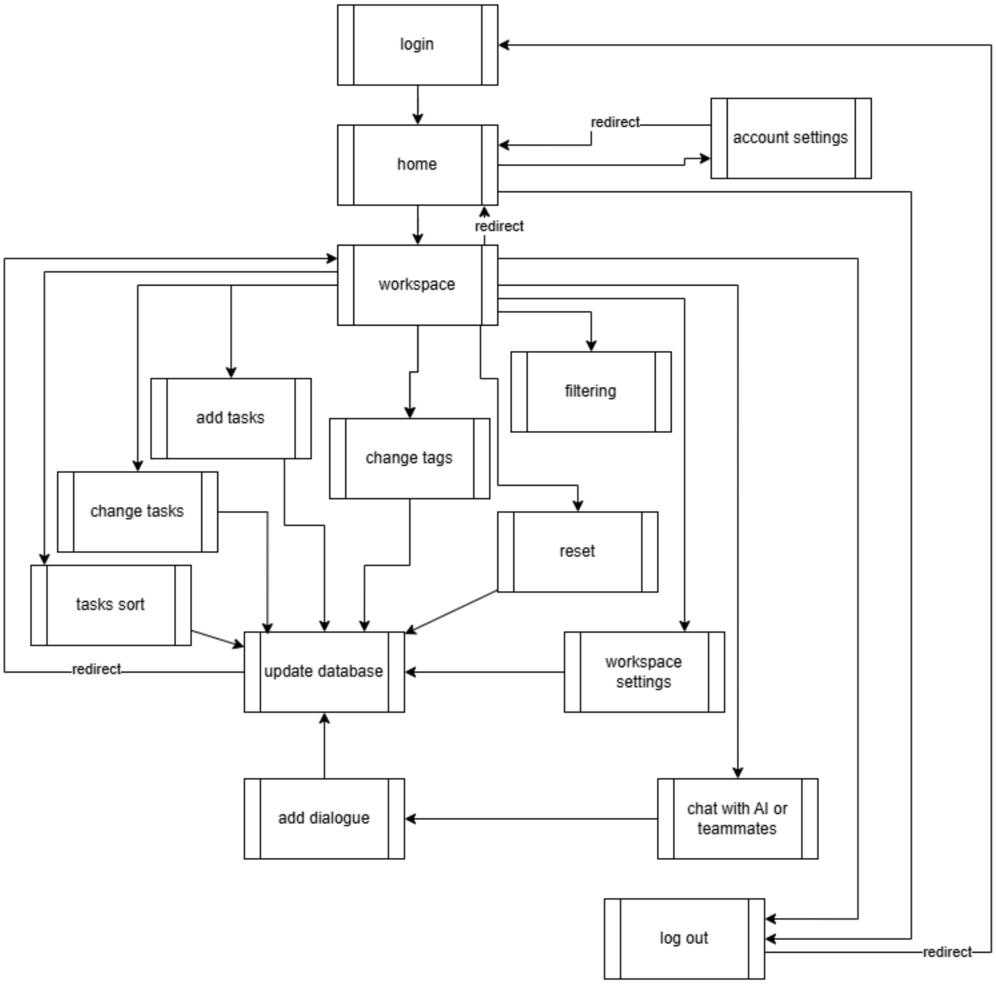
3.1 Project Plan

Pages included:

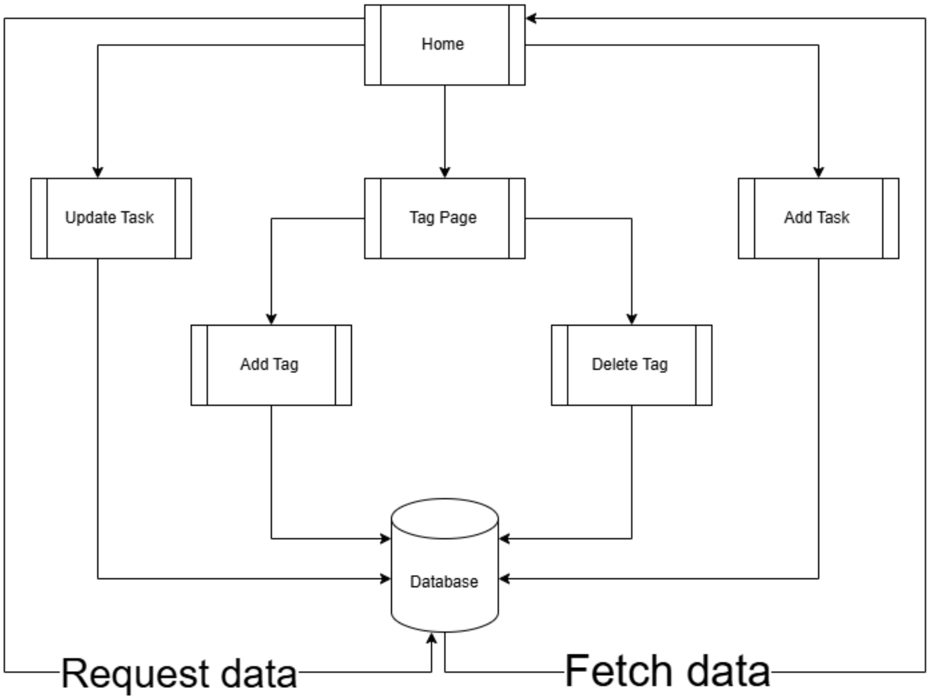
1. Index page
2. Update page
3. Error page
4. Tag page
5. Error report page (to be continued)
6. Feedback page (to be continued)

3.2 Flowchart

Flowchart of desired application:

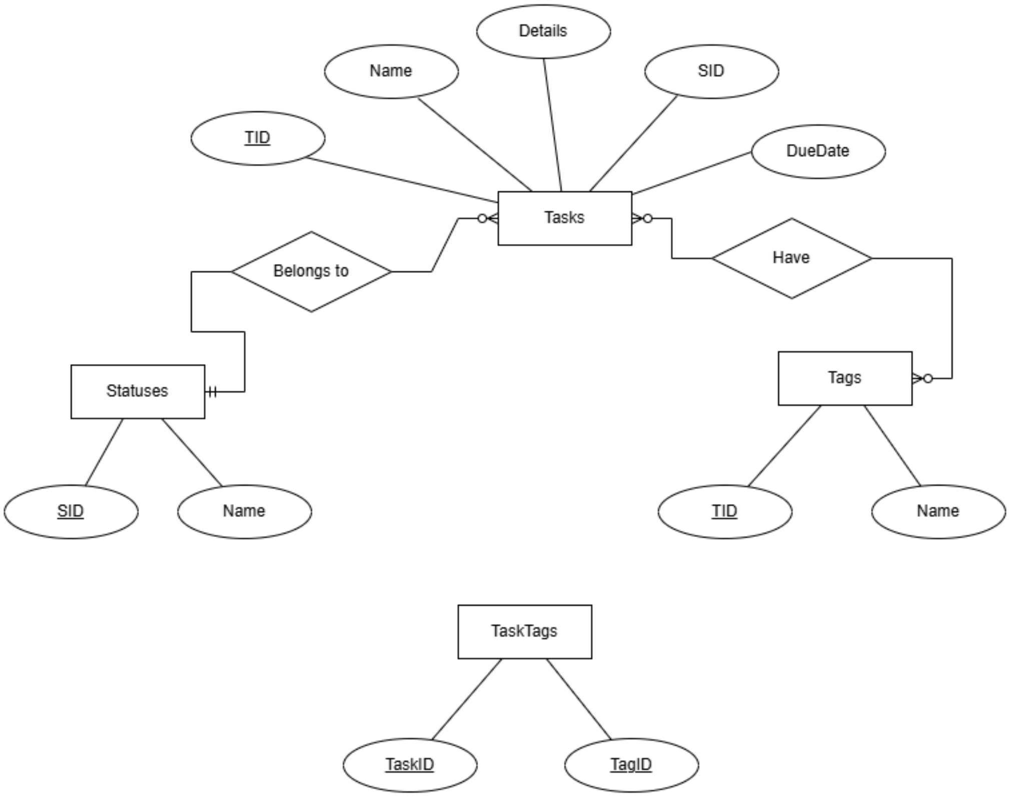


Flowchart of current application:


Home
Update Task
Tag Page
Add Task
Add Tag
Delete Tag
Database
Request data
Fetch data

3.3 ER Diagram

ER Diagram of current application:



Participation Constraint:

- A task must have a progress status but may or may not have a tag
- A status may or may not belong to a task
- A tag may or may not belong to a task

Cardinality of relationship:

- A task must have exactly one status and may have many tags, or none at all
- A status may belong to many tasks, or none at all
- A tag may belong to many tasks, or none at all

3.4 Database schema

Schema:

Tasks (TID, Name, Details, SID, DueDate)

```
Foreign Key: SID (from Statuses SID)
```

Tags (TID, Name)

```
Primary Key: TID or Name
```

Statuses (SID, Name)

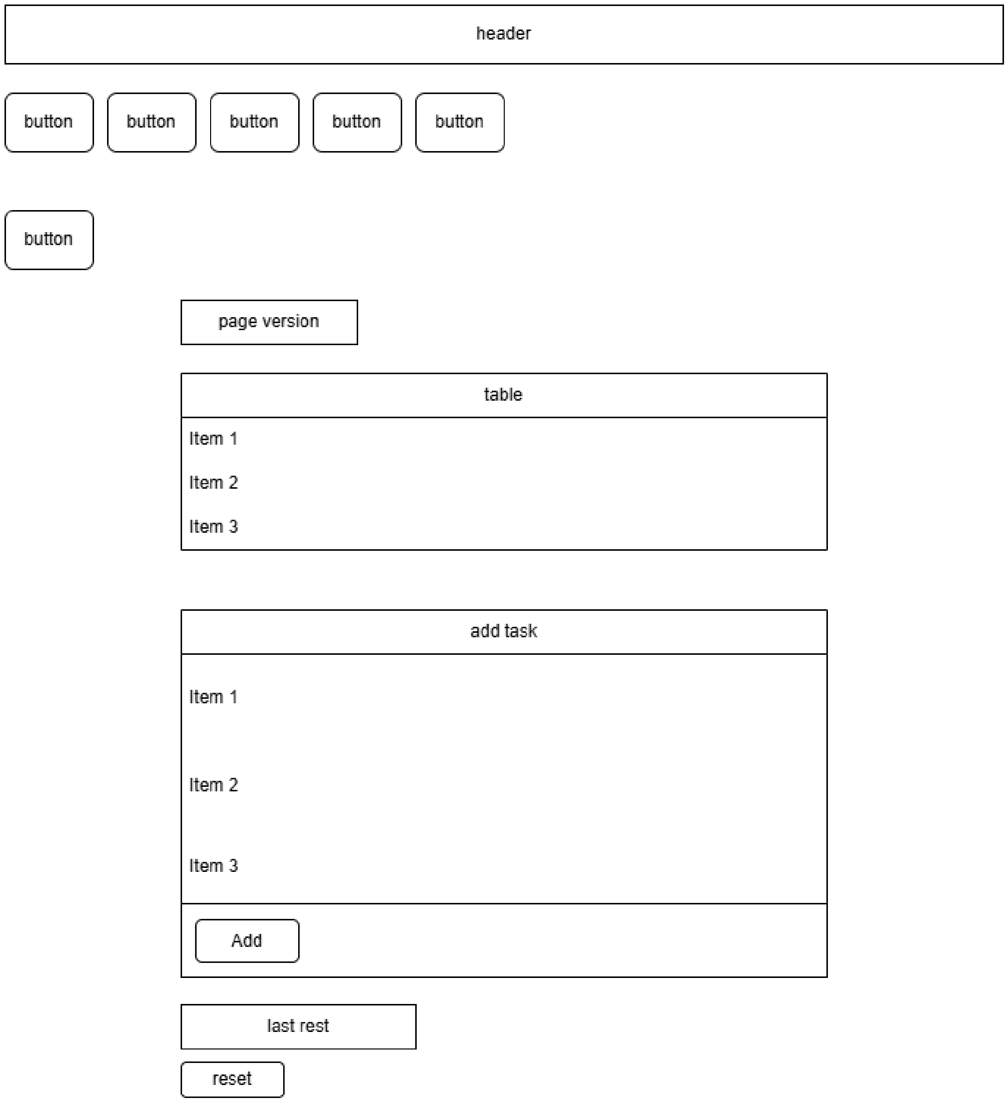
```
Primary Key: SID
```

TaskTags (TaskID, TagID)

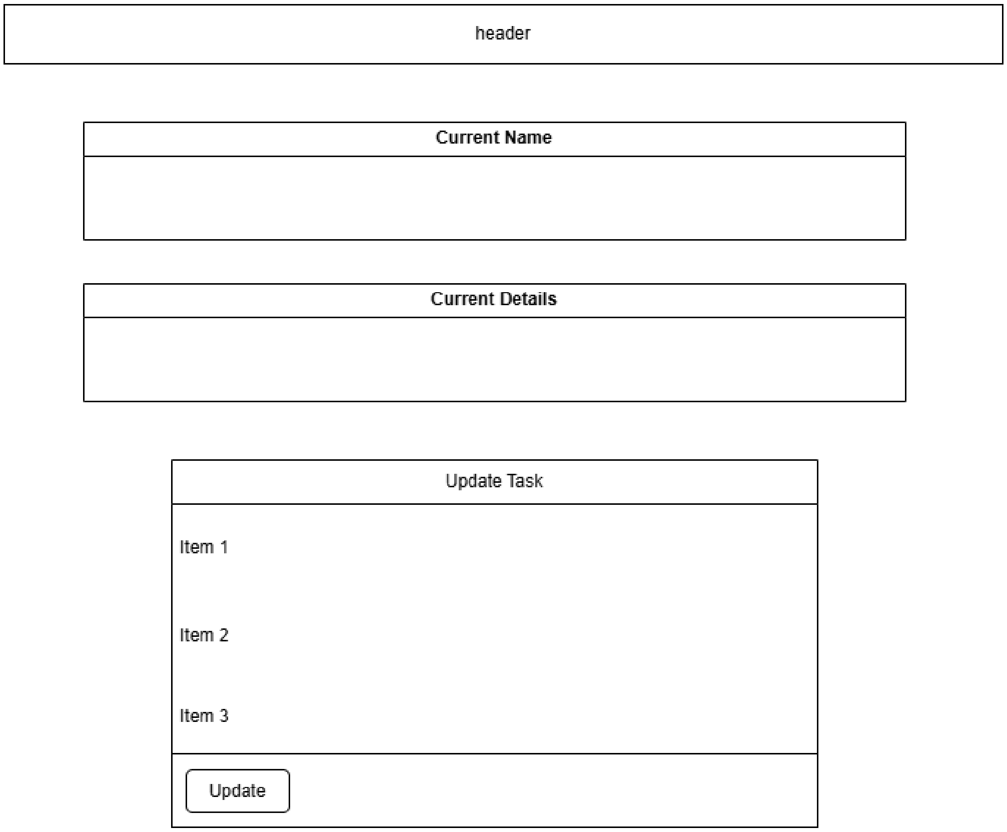
```
Primary Key: TaskID + TagID
Foreign Key: TaskId (from Tasks TID), TagID (from Tags TID)
```

3.5 Design

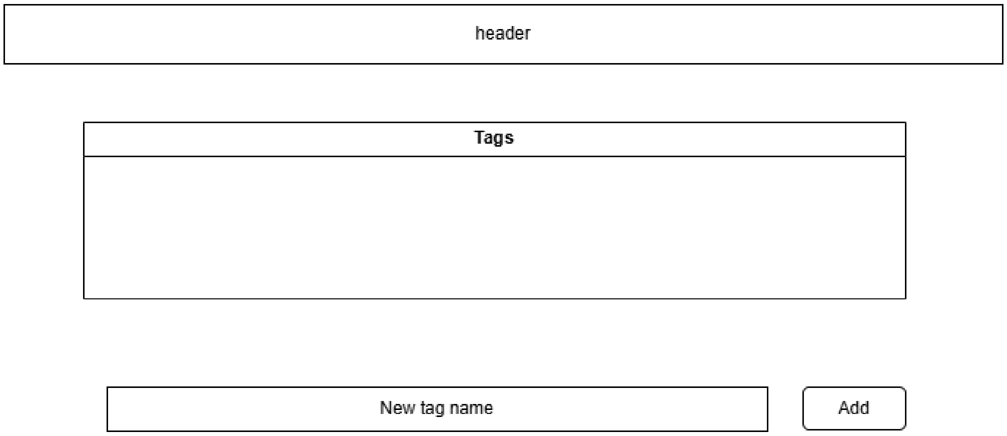
The layout design of index page:



The layout design of update page:



The layout design of tag page:


header
Tags
New tag name
Add

4 Methodology

4.1 Environment setup

There are a lot of frameworks which can be used to launch the application such as Flask and Django. After a few months of testing, Flask is chosen as its structure is more simplified than others.

Windows Version:

1. Install python3.10
2. Unzip the folder
3. Move to the unzipped folder directory
4. Type the commands:

```
python -m venv env
env\Scripts\activate
cd Achilles
pip install -r requirements.txt
python app.py
```

4.2 Database

4.2.1 relational database design

Schema:

Tasks (TID, Name, Details, SID, DueDate)

```
Foreign Key: SID (from Statuses SID)
```

Tags (TID, Name)

```
Primary Key: TID or Name
```

Statuses (SID, Name)

```
Primary Key: SID
```

TaskTags (TaskID, TagID)

```
Primary Key: TaskID + TagID
Foreign Key: TaskId (from Tasks TID), TagID (from Tags TID)
```

The Database is in **Third Normal Form**, reasons are:

1. Every attribute do not contain multiple values
2. There are not any repeating attributes
3. There are not any partial functional dependency between attributes
4. There are not any transitive functional dependency between attributes

Benefit of Normalization:

1. Higher flexibility of data management
2. Less storage space required
3. Prevent update, insertion and deletion anomaly

4.2.2 data redundancy

In each insertion, the primary key TID of Tasks will prevent repeating records from being inserted into Tasks.

In each update, the program first updates the record in Tasks with corresponding TID, then deletes all the records with the corresponding TID in TaskTags, finally inserts the records representing the relationship of tags and tasks into TaskTags. This can prevent data inconsistency.

In each deletion, the program deletes the records with the corresponding TID in Tasks and TaskTags. This can help maintain data consistency.

Therefore, update anomaly, insertion anomaly, deletion anomaly will not occur.

4.2.3 data integrity

SQL table creation statement:

```
"""
CREATE TABLE IF NOT EXISTS Tasks (
    TID INTEGER PRIMARY KEY AUTOINCREMENT,
    Name VARCHAR(255) NOT NULL DEFAULT 'no name',
    Details VARCHAR(255) DEFAULT NULL,
    SID INTEGER DEFAULT 0 NOT NULL,
    DueDate datetime,
    FOREIGN KEY (SID) REFERENCES Statuses(SID)
);

CREATE TABLE IF NOT EXISTS Tags (
    TID INTEGER PRIMARY KEY AUTOINCREMENT,
    Name VARCHAR(255) UNIQUE
);

CREATE TABLE IF NOT EXISTS Statuses (
    SID INTEGER PRIMARY KEY AUTOINCREMENT,
    Name VARCHAR(255) UNIQUE
);

CREATE TABLE IF NOT EXISTS TaskTags (
    TaskID INTEGER,
    TagID INTEGER,
    PRIMARY KEY (TaskID, TagID),
    FOREIGN KEY (TaskID) REFERENCES Tasks(TID),
    FOREIGN KEY (TagID) REFERENCES Tags(TID)
)
"""
```

Constraints of fields:

- Tasks

  TID An integer field that acts as the primary key, automatically incrementing for each new record.

  Name A varchar(255) field that stores the task name. It's defined as NOT NULL with a default value of 'no name' to ensure every task has a name.

  Details A varchar(255) field that can store additional details about the task. It has a DEFAULT NULL value as it is optional.

SID An integer field that references the SID column in the Statuses table, establishing a foreign key relationship. This ensures that every task is associated with a valid status.

DueDate A datetime field that stores the due date of the task.

- Tags

  TID An integer field that acts as the primary key, automatically incrementing for each new record.

  Name A varchar(255) field that stores the tag name. It is defined as UNIQUE to ensure no two tags have the same name.

- Statuses

  SID An integer field that acts as the primary key, automatically incrementing for each new record.

  Name A varchar(255) field that stores the status name. It is defined as UNIQUE to ensure no two statuses have the same name.

- TaskTags

  TaskID An integer field that references the TID column in the Tasks table and acts as part of the composite key.

  TagID An integer field that references the TID column in the Tags table and acts as part of the composite key.

4.2.4 SQL implementation

Table creation statement: See **Section 4.2.3**

Record selection statement:

Tasks

```
SELECT * FROM Tasks WHERE SID = ?
SELECT * FROM Tasks WHERE TID = ?
SELECT MAX(TID) FROM Tasks
SELECT * FROM Tasks WHERE DueDate BETWEEN datetime('now', 'localtime') AND
datetime('now','+7 day','localtime')
```

Tags

```
SELECT * FROM Tags ORDER BY Name
```

Statuses

```
SELECT * FROM Statuses
```

TaskTags

```
SELECT * FROM TaskTags
SELECT * FROM TaskTags WHERE TagID = ?
```

Record insertion statement:

Tasks (with date)

```
INSERT INTO Tasks (Name, Details, SID, DueDate) VALUES (?, ?, ?, ?)
```

Tasks (without date)

```
INSERT INTO Tasks (Name, Details, SID) VALUES (?, ?, ?)
```

Statuses

```
INSERT INTO Statuses (Name) VALUES (?)
```

Tags

```
INSERT INTO Tags (Name) VALUES (?)
```

TaskTags

```
INSERT INTO TaskTags (TaskID, TagID) VALUES (?, ?)
```

Record update statement:

Tasks (with date)

```
UPDATE Tasks SET Name = ?, Details = ?, SID = ?, DueDate = ? WHERE TID = ?
```

Tasks (without date)

```
UPDATE Tasks SET Name = ?, Details = ?, SID = ? WHERE TID = ?
```

Tags

```
DELETE FROM Tags WHERE TID = ?
```

Record deletion statement:

Tasks

```
DELETE FROM Tasks WHERE TID = ?
```

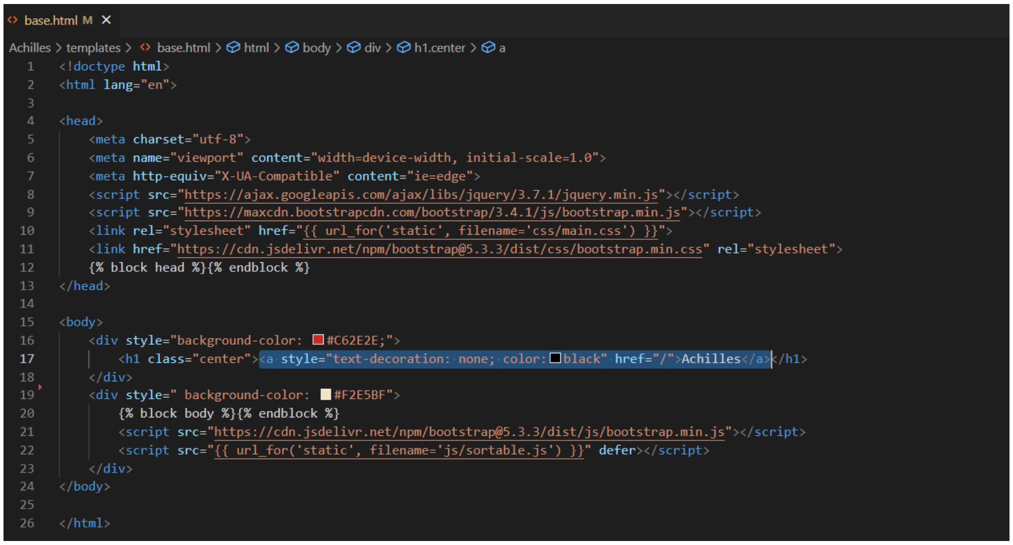
TaskTags

```
DELETE FROM TaskTags WHERE TID = ?
DELETE FROM TaskTags WHERE TaskID = ?
```

4.2.5 user friendliness

There are colored visual elements, and different clickable buttons to increase convenience and user friendliness

Users can return to the home index page by simply clicking the header ”Achilles” as there is a hyperlink set in **base.html**


base.html M
Achilles > templates > base.html > html > body > div > h1.center > a

```html
<!doctype html>
<html lang="en">

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/main.css') }}">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
    {% block head %}{% endblock %}
</head>

<body>
    <div style="background-color: #C62E2E;">
        <h1 class="center"><a style="text-decoration: none; color: black" href="/">Achilles</a></h1>
    </div>
    <div style=" background-color: #F2E5BF">
        {% block body %}{% endblock %}
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.min.js"></script>
        <script src="{{ url_for('static', filename='js/sortable.js') }}" defer></script>
    </div>
</body>

</html>
```

4.3 Program

4.3.1 Program structure

In **app.py**:

```
from flask import Flask, render_template, request, redirect
from db import *
import sqlite3
import mysql.connector
from werkzeug.exceptions import HTTPException
import json
import requests
from datetime import datetime
import pytz

app = Flask(__name__)

con = sqlite3.connect('info.db',
                             detect_types=sqlite3.PARSE_DECLTYPES |
                             sqlite3.PARSE_COLNAMES,
    check_same_thread=False)
cur = con.cursor()

# codes here ...

if __name__ == "__main__":
    app.run(debug=True)
```

In **db.py**:

```
import sqlite3
import mysql.connector

con = sqlite3.connect('info.db',
                             detect_types=sqlite3.PARSE_DECLTYPES |
                             sqlite3.PARSE_COLNAMES,
    check_same_thread=False)
cur = con.cursor()

# codes here ...
```

4.3.2 Selection of data types and data structures

Data types:

Integer

`storing ID of records, such as TID in Tasks and SID in Statuses`

String

`storing textual data, such as Details in Tasks`

Datetime

`storing Date and time information of the DueDate in Tasks`

Data structures:

List

`storing multiple constant values, such as tables in \textbf{db.py}`

Couple

`storing a sequence of values passing parameters to SQL queries`

Dictionary

```
storing corresponding keys and values of HTTP request.form in \textbf{app.py} or
corresponding names of tables and SQL table creation statement in \textbf{db.py}
```

4.3.3 Variable/Constant declaration and initialization

In **db.py**, SQL statements of initial table creation and names of tables or default names of elements are stored in constant variables as shown below:

```
tables = ["Tasks", "Tags", "Statuses", "TaskTags"]

tablemap = {
"Tasks":"""
CREATE TABLE IF NOT EXISTS Tasks (
    TID INTEGER PRIMARY KEY AUTOINCREMENT,
    Name VARCHAR(255) NOT NULL DEFAULT 'no name',
    Details VARCHAR(255) DEFAULT NULL,
    SID INTEGER DEFAULT 0 NOT NULL,
    DueDate datetime,
    FOREIGN KEY (SID) REFERENCES Statuses(SID)
)
""",

"Tags":"""
CREATE TABLE IF NOT EXISTS Tags (
    TID INTEGER PRIMARY KEY AUTOINCREMENT,
    Name VARCHAR(255) UNIQUE
)
""",

"Statuses":"""
CREATE TABLE IF NOT EXISTS Statuses (
    SID INTEGER PRIMARY KEY AUTOINCREMENT,
    Name VARCHAR(255) UNIQUE
)
""",

"TaskTags":"""
CREATE TABLE IF NOT EXISTS TaskTags (
    TaskID INTEGER,
    TagID INTEGER,
    PRIMARY KEY (TaskID, TagID),
    FOREIGN KEY (TaskID) REFERENCES Tasks(TID),
    FOREIGN KEY (TagID) REFERENCES Tags(TID)
)
""",
}

def setup():
    for i in tables:
        cur.execute(tablemap[i])
    con.commit()

def ins(a, b):
    q = f"SELECT 1 FROM {b} WHERE EXISTS (SELECT * FROM {b} WHERE Name = ?)"
    res = cur.execute(q, (a,))
    t = res.fetchall()
    if len(t)==0:
        q = f"INSERT INTO {b} (Name) VALUES (?)"
        cur.execute(q, (a,))
    con.commit()
    return ""

def init():
    lt = ["Not started", "On-going", "Completed", "I don\'t know"]
    for i in lt:
        ins(i, "Statuses")
    lt = ["School", "Home"]
    for i in lt:
        ins(i, "Tags")
```

4.3.4 Modular approach

Modules in **db.py**:

- init()
  - create the tables if they do not exist
  - initialize the preset values in the Tags and Statuses tables
- reset()
  - reset everything in the database by dropping all the tables and recreating them again
- raw(¡statement¿)
  - execute raw SQL statements without passing additional parameters
- raww(¡statement¿)
  - execute raw SQL queries without passing additional parameters and return the values fetched as a list

Modules in **app.py**:

- IndexGet()
  - return the **index.html** with fetched values from database for GET method
- IndexPost(¡response¿)
  - Inserts new records of Tasks and TaskTags and return the **index.html** with fetched values from database for GET method if no error occur
- UpdateGet()
  - return the **update.html** with fetched values from database for GET method
- UpdatePost(¡response¿)
  - update old records of Tasks and TaskTags and return the **index.html** with fetched values from database for GET method if no error occur
- TagGet()
  - return the **tags.html** with fetched values from database for GET method
- TagPost()
  - Inserts new records of Tag and return the **tags.html** with fetched values from database for GET method if no error occur

4.3.5 Scope of variables and parameters passing

The application does not use global variables to store trivial data. Instead, it use **data.json** file to store the trivial variables which cannot be reset by restarting the main program.

In **app.py**, the program passes parameters into different functions to achieve modular approach:

```
if request.method == "POST":
    return IndexPost(request.form)
```

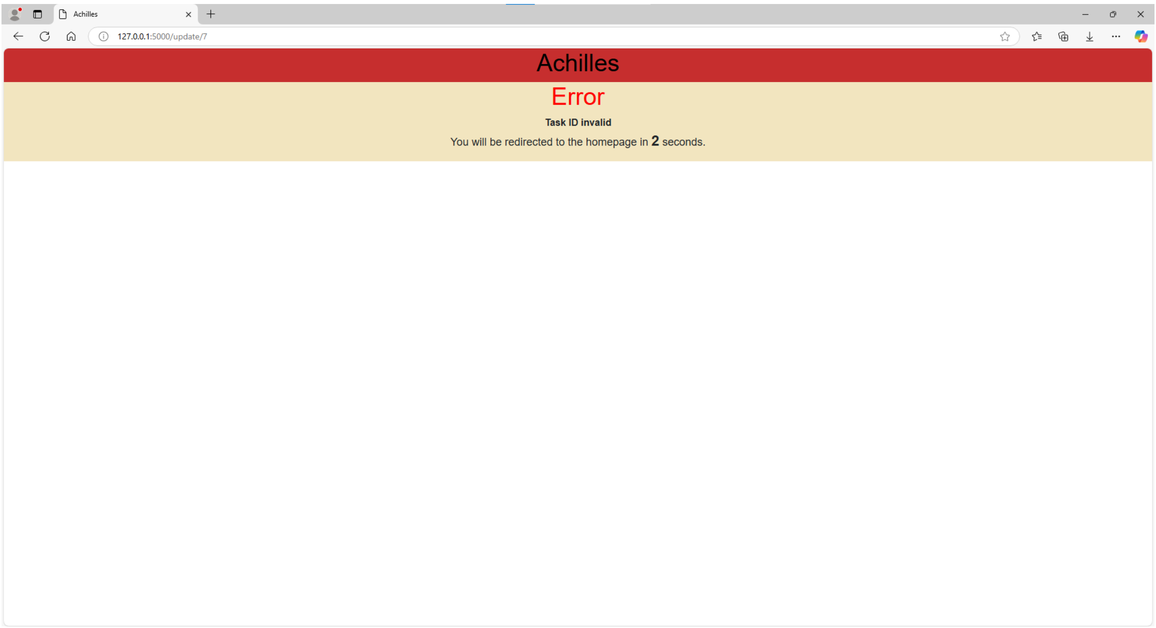
```
raw(query)
raww("SELECT * FROM Tags")
```

Also, the website url passes the TID of Tasks for deletion and update:

```
@app.route('/delete/<int:id>')
@app.route('/update/<int:id>', methods=['GET', 'POST'])
```

4.3.6 interface of program

Error Page:



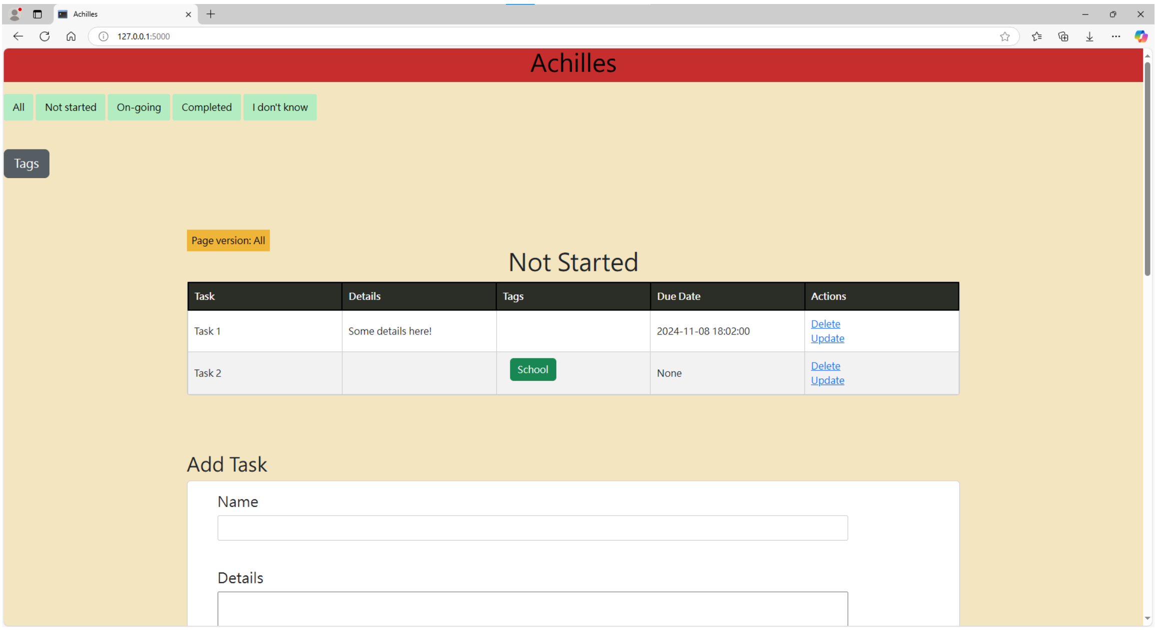
Achilles

Error

Task ID invalid

You will be redirected to the homepage in **2** seconds.

Main Page:



Achilles

Not Started

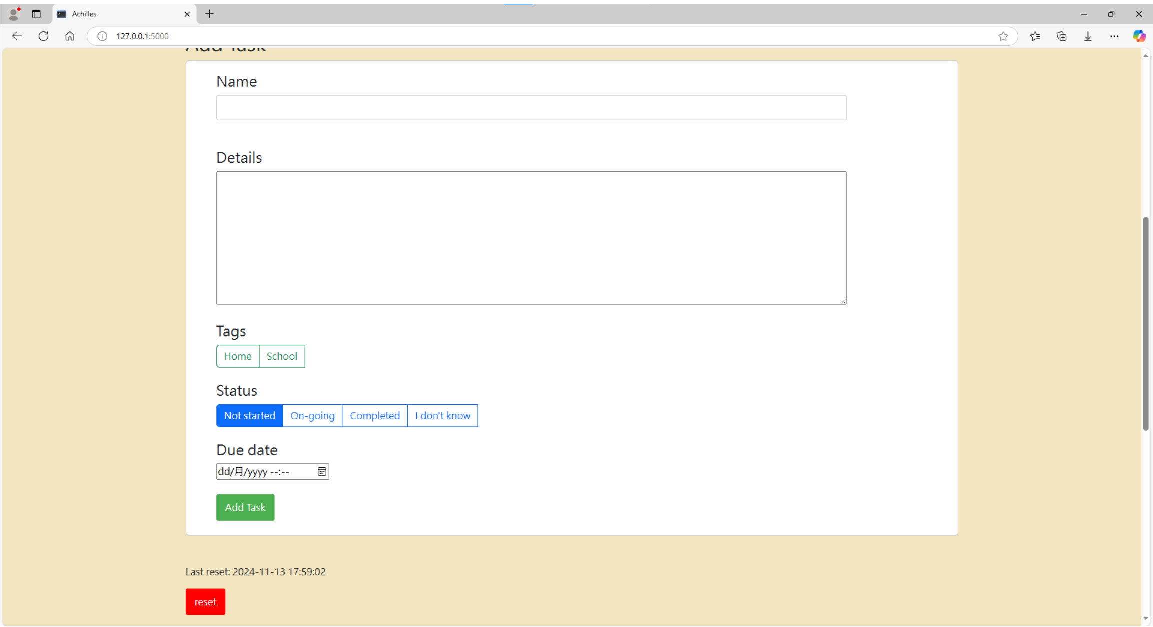
Task	Details	Tags	Due Date	Actions
Task 1	Some details here!		2024-11-08 18:02:00	Delete Update
Task 2		School	None	Delete Update

Add Task

Name

Details

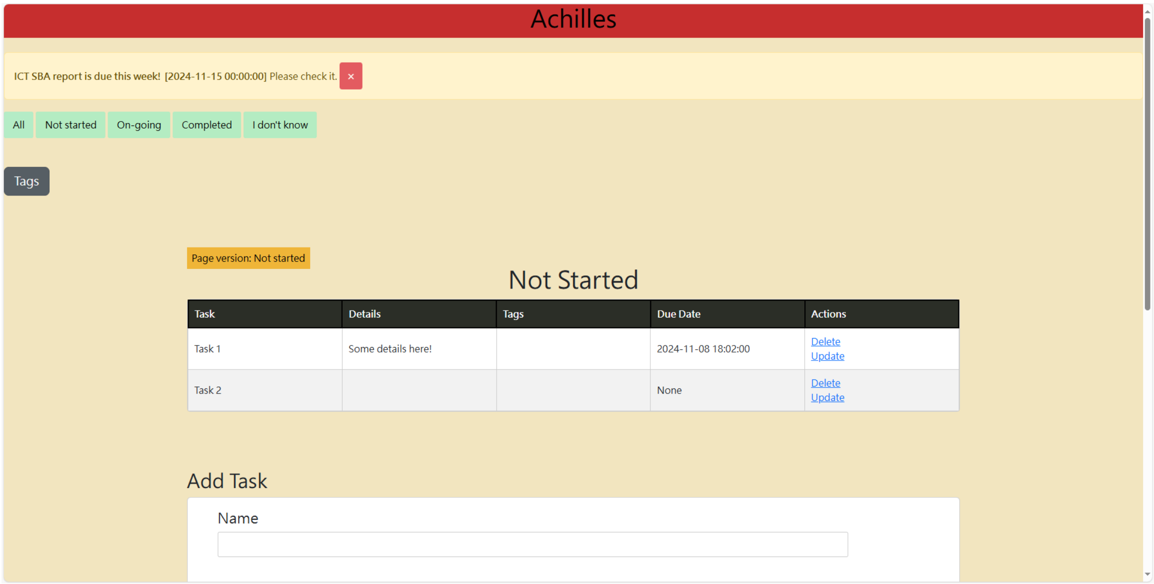
Main Page 2 – Add Task:



Last reset: 2024-11-13 17:59:02

Main Page 3 – Change Page Version:

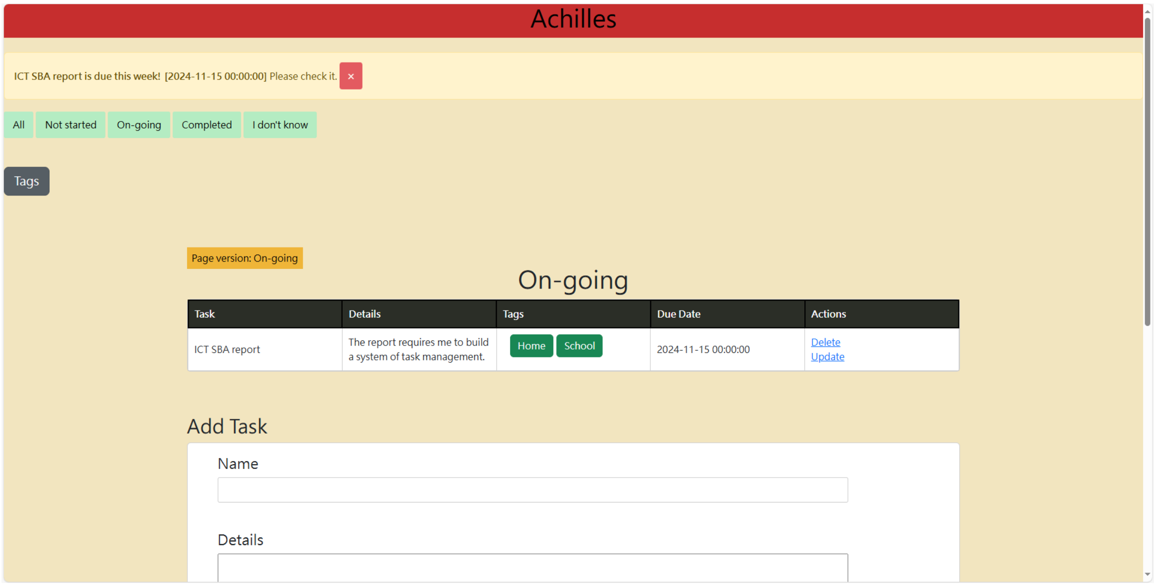
By clicking the top left buttons, the page will filter out the tasks of different statuses



ICT SBA report is due this week! [2024-11-15 00:00:00] Please check it.

Not Started

Task	Details	Tags	Due Date	Actions
Task 1	Some details here!		2024-11-08 18:02:00	Delete Update
Task 2			None	Delete Update



ICT SBA report is due this week! [2024-11-15 00:00:00] Please check it.

Page version: On-going

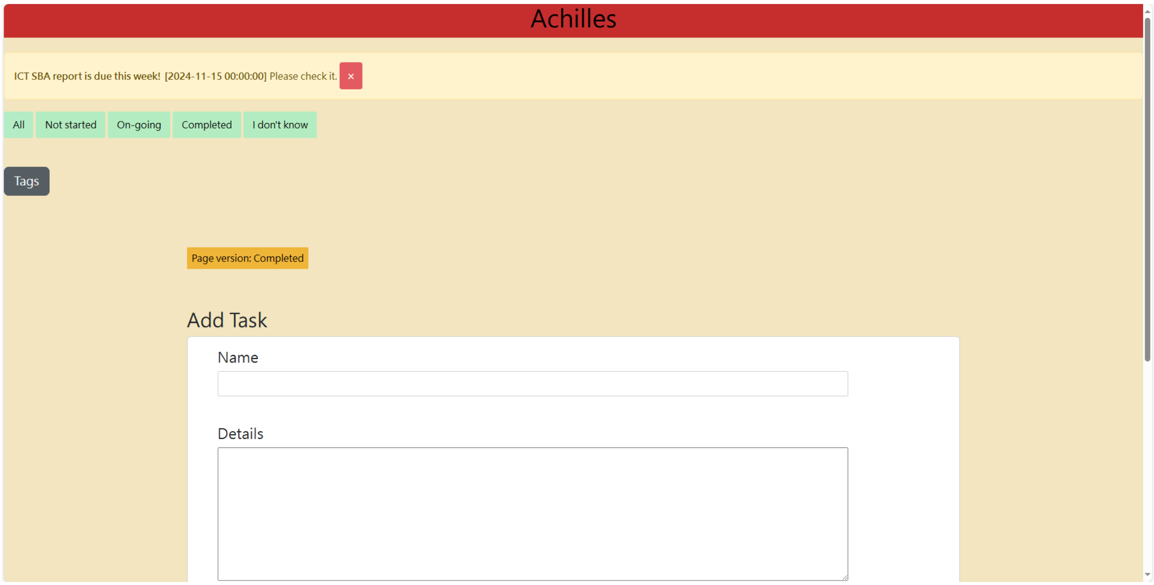
On-going

Task	Details	Tags	Due Date	Actions
ICT SBA report	The report requires me to build a system of task management.	Home School	2024-11-15 00:00:00	Delete Update

Add Task

Name

Details



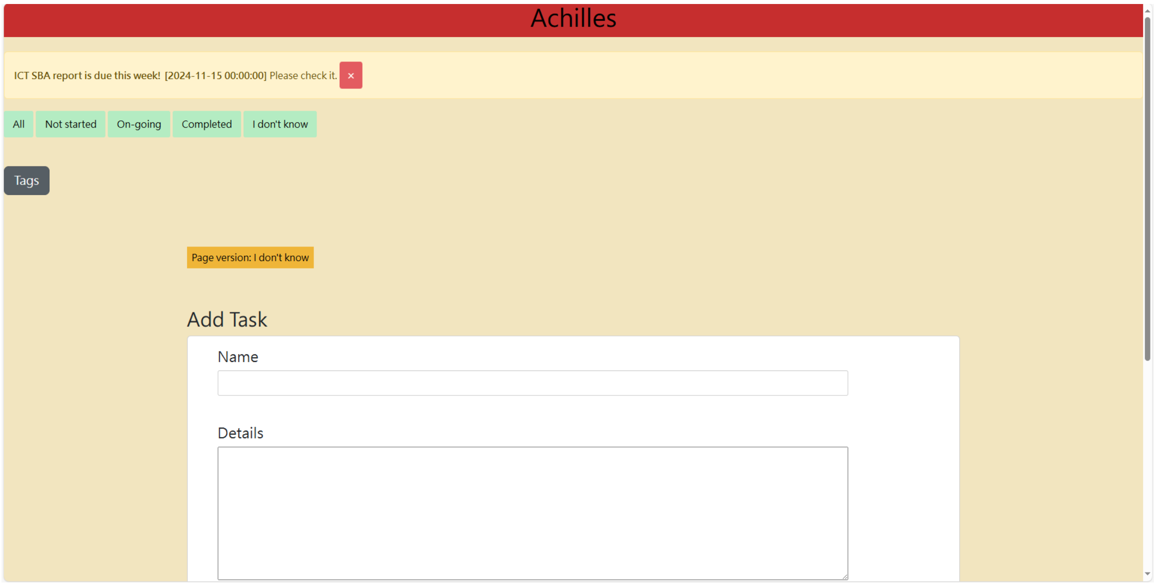
ICT SBA report is due this week! [2024-11-15 00:00:00] Please check it.

Page version: Completed

Add Task

Name

Details

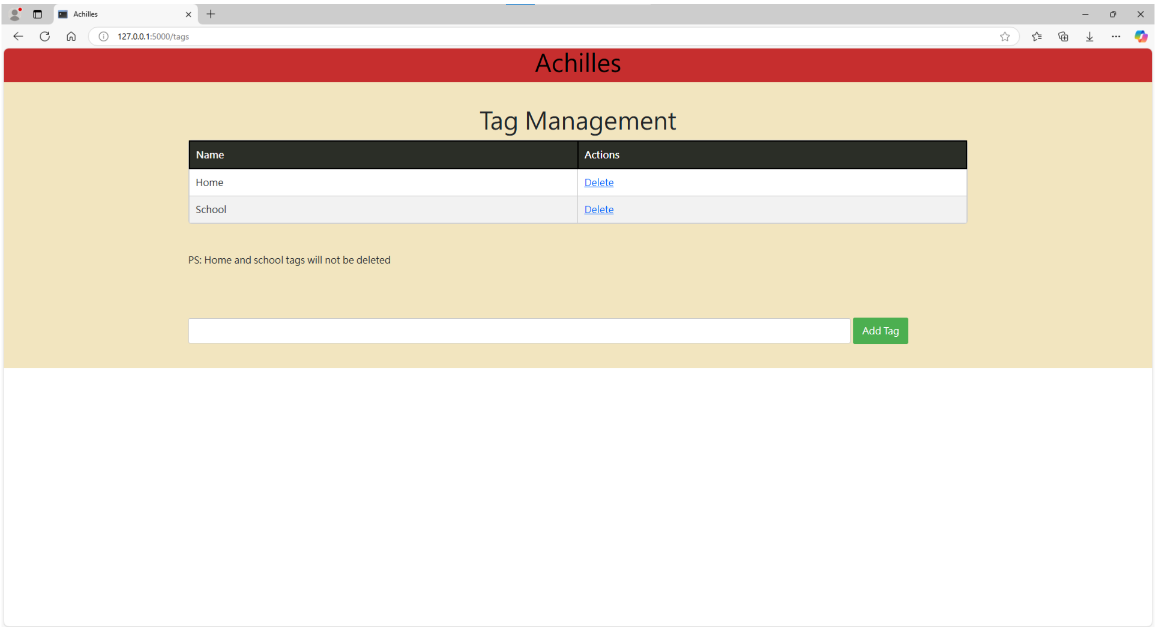


ICT SBA report is due this week! [2024-11-15 00:00:00] Please check it.

Page version: I don't know

Add Task

Tag Page:



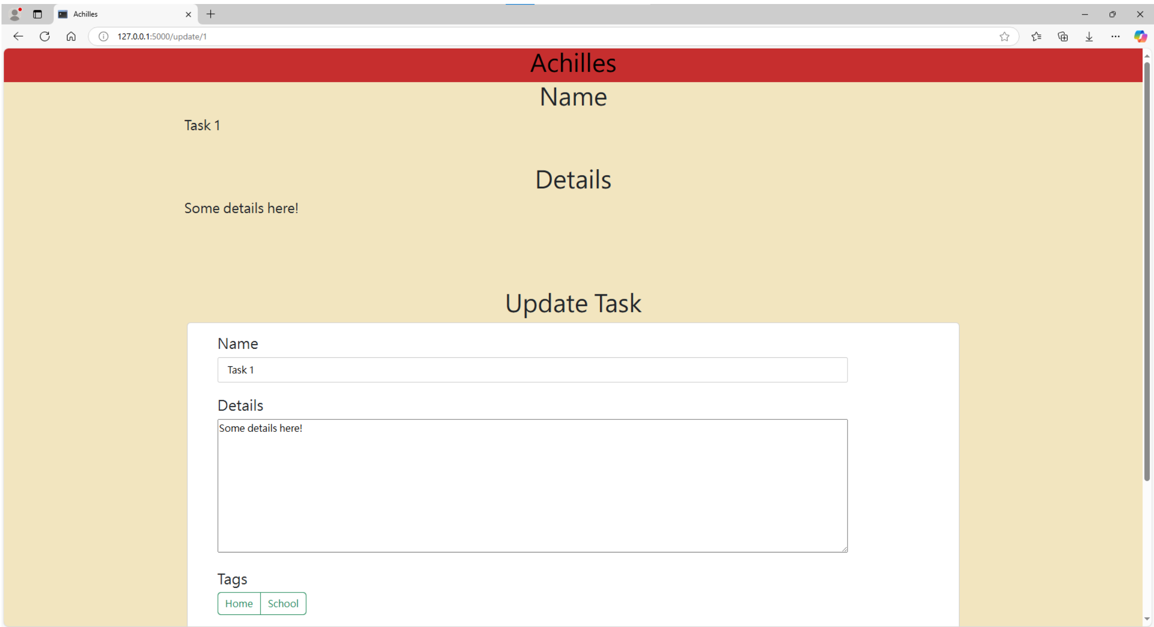
Achilles

Tag Management

Name	Actions
Home	Delete
School	Delete

PS: Home and school tags will not be deleted

Update Page:



Achilles

Name

Task 1

Details

Some details here!

Update Task

Name

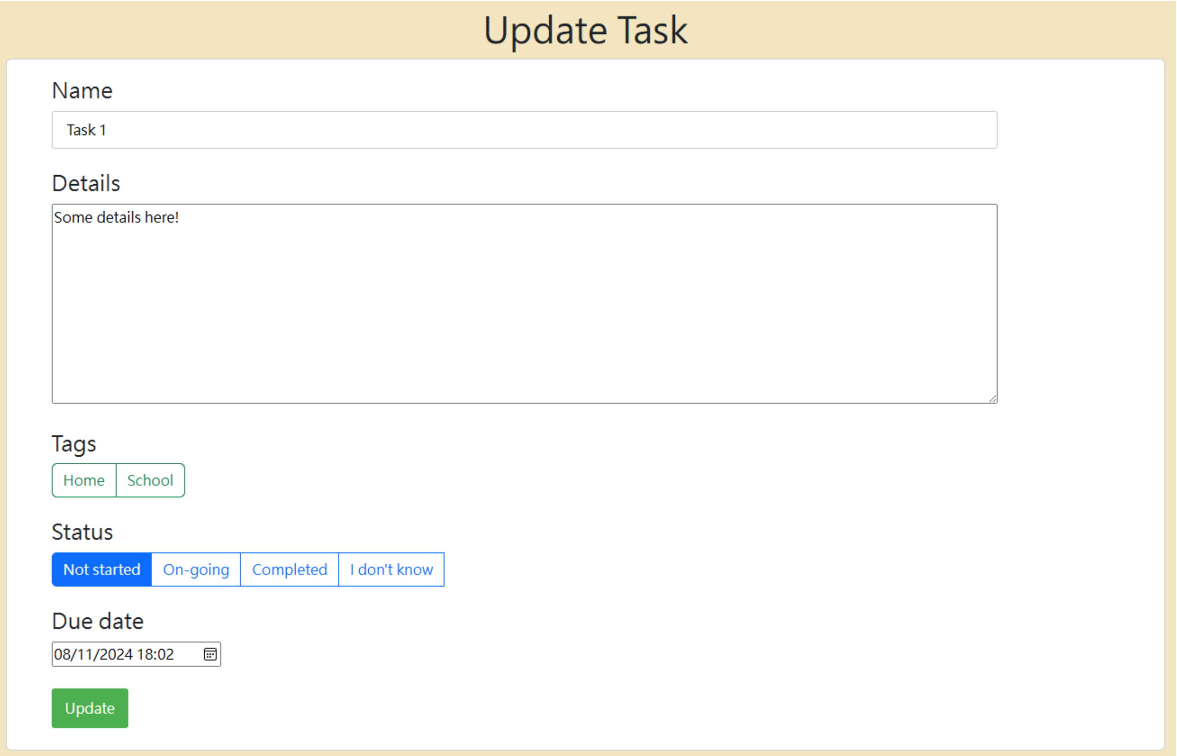
Task 1

Details

Some details here!

Tags

Update Page 2:



Update Task

Name

Task 1

Details

Some details here!

Tags

Home School

Status

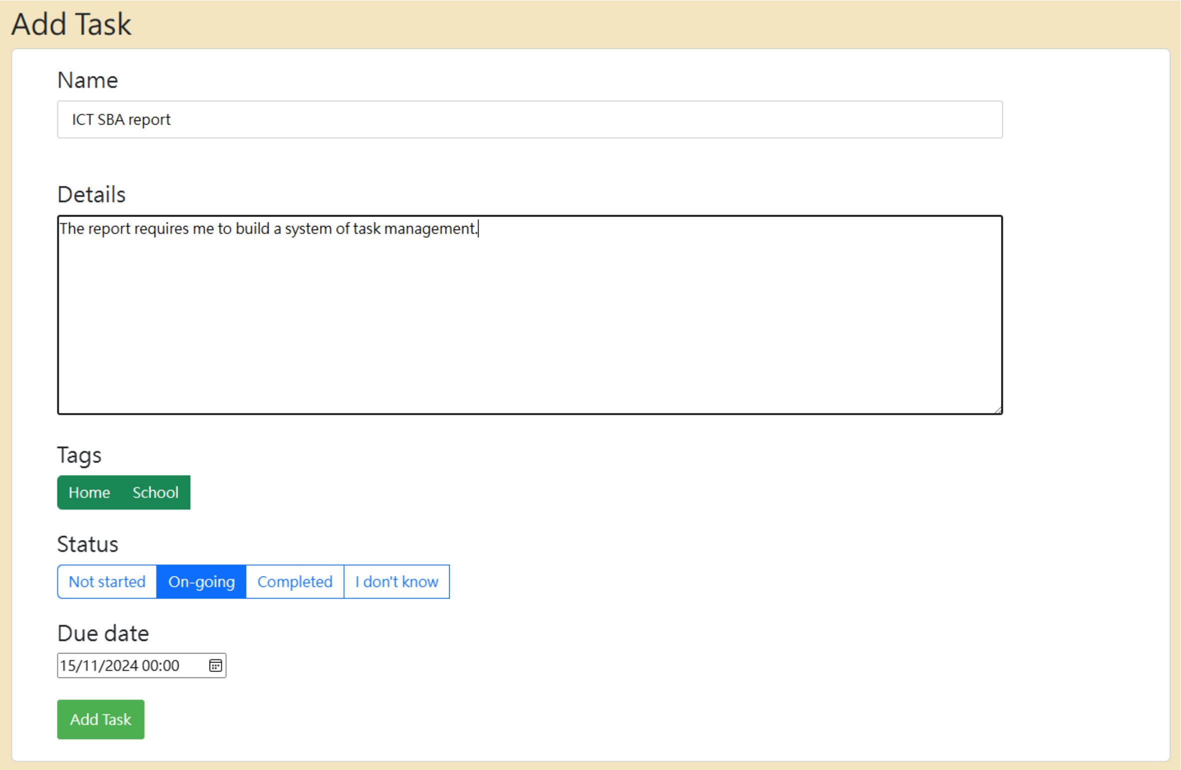
[✓ Not started] On-going Completed I don't know

Due date

08/11/2024 18:02

4.3.7 data collection, input and validation

Users can input the info of the coming tasks into the text boxes in the "Add Task" area in index page. They can also choose the tags, status and due date of the task.



Add Task

Name

ICT SBA report

Details

The report requires me to build a system of task management.

Tags

Home School

Status

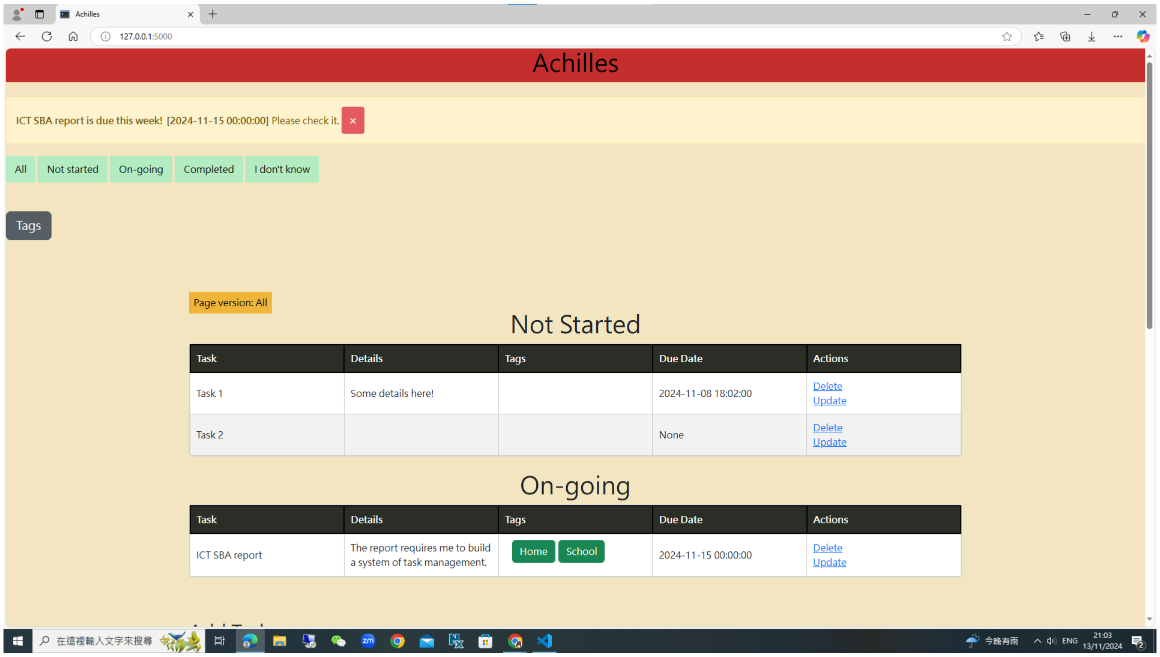
Not started [✓ On-going] Completed I don't know

Due date

15/11/2024 00:00

After clicking the "Add Task" button, the task is inserted into database. An alert of coming task is triggered as 2024-11-15 is less then a week after the current time (2024-11-13). The alert will disappear if the "cross" button is clicked.

The data of the task can be collected and displayed in the home tables in this way:



Achilles

ICT SBA report is due this week! [2024-11-15 00:00:00] Please check it.

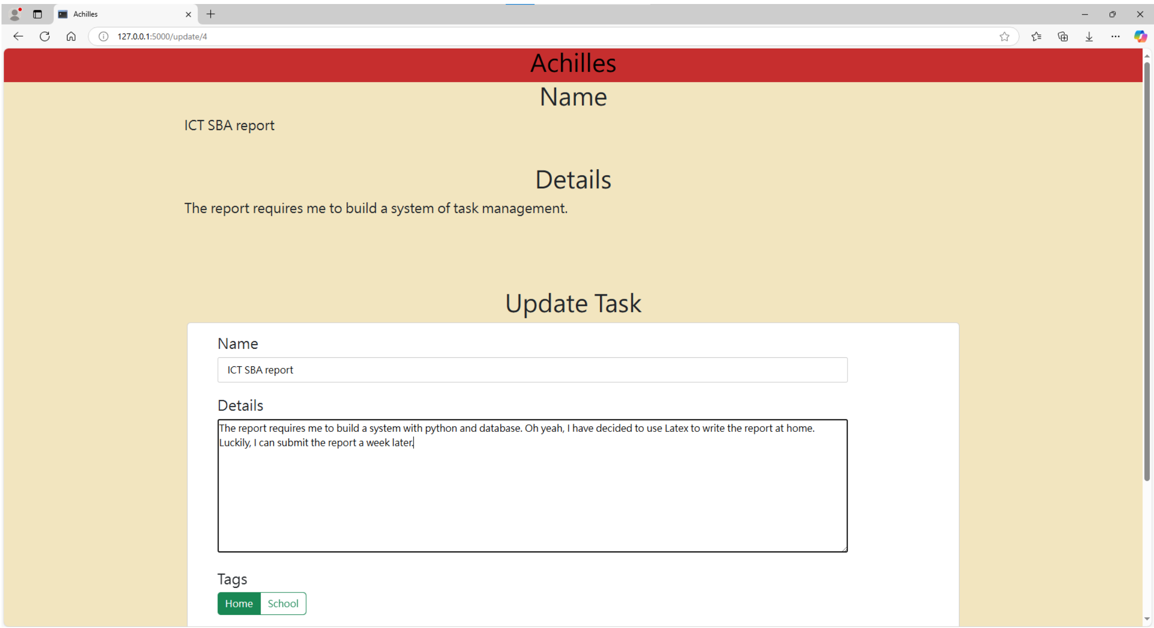
Not Started

Task	Details	Tags	Due Date	Actions
Task 1	Some details here!		2024-11-08 18:02:00	Delete Update
Task 2			None	Delete Update

On-going

Task	Details	Tags	Due Date	Actions
ICT SBA report	The report requires me to build a system of task management.	Home School	2024-11-15 00:00:00	Delete Update

If users want to change the information of the tasks, just click "Update" in the corresponding row. After clicking, users can change the information in the text boxes in "Update Task" area.



Achilles

Name

ICT SBA report

Details

The report requires me to build a system of task management.

Update Task

Name

ICT SBA report

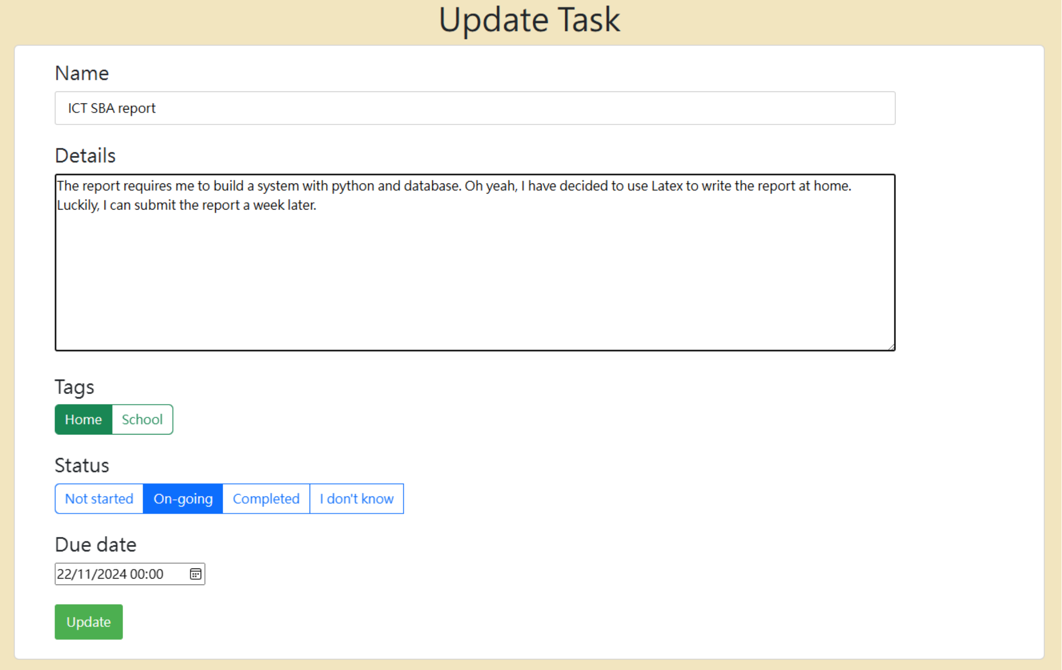
Details

The report requires me to build a system with python and database. Oh yeah, I have decided to use Latex to write the report at home. Luckily, I can submit the report a week later.

Tags

Home School

Change the tags, status and due date, then click "Update"



Update Task

Name

ICT SBA report

Details

The report requires me to build a system with python and database. Oh yeah, I have decided to use Latex to write the report at home. Luckily, I can submit the report a week later.

Tags

[✓ Home] School

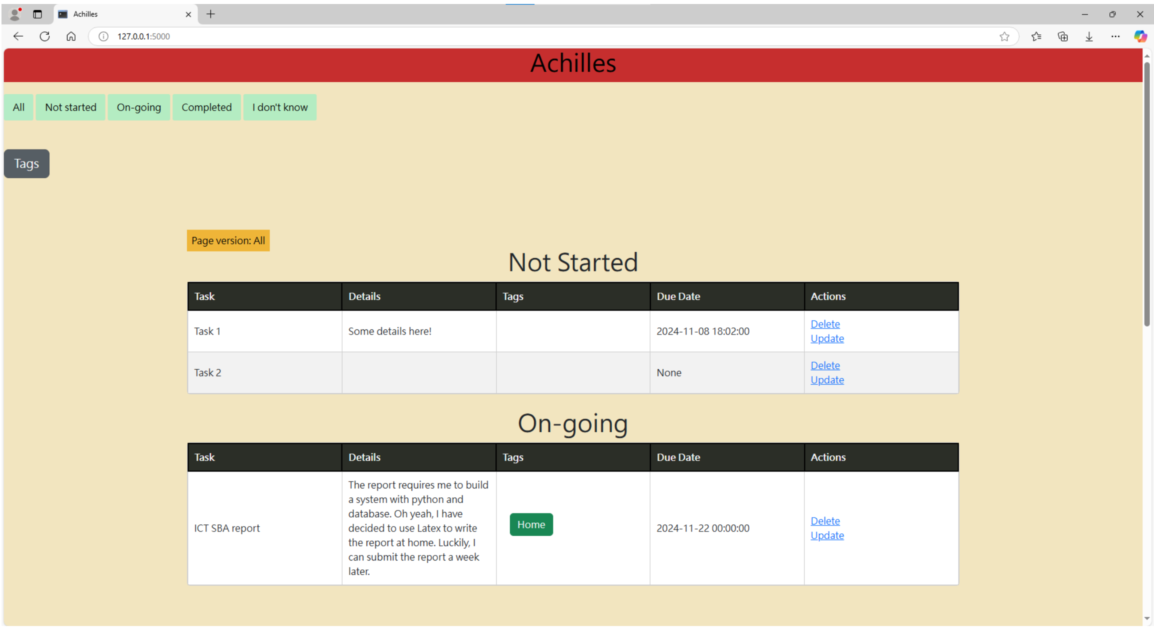
Status

Not started [✓ On-going] Completed I don't know

Due date

22/11/2024 00:00

The data of the task is then updated



Achilles

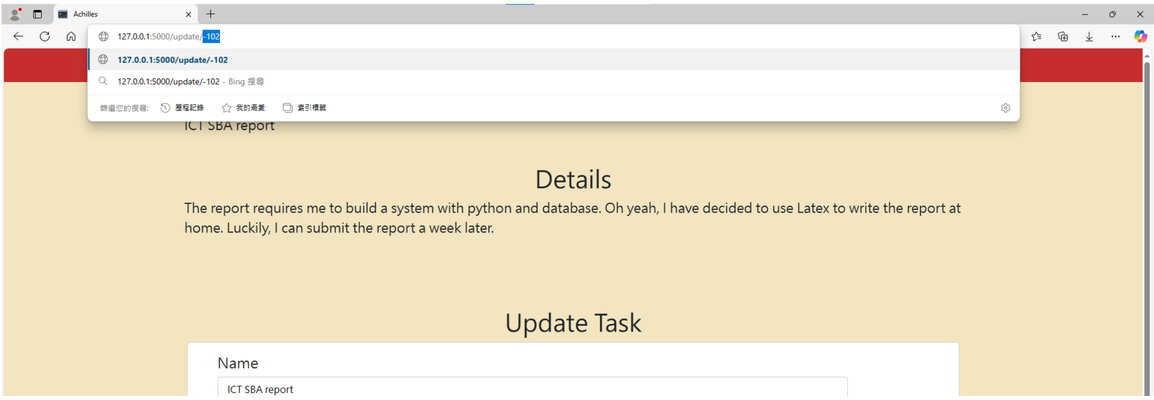
Not Started

Task	Details	Tags	Due Date	Actions
Task 1	Some details here!		2024-11-08 18:02:00	Delete Update
Task 2			None	Delete Update

On-going

Task	Details	Tags	Due Date	Actions
ICT SBA report	The report requires me to build a system with python and database. Oh yeah, I have decided to use Latex to write the report at home. Luckily, I can submit the report a week later.	Home	2024-11-22 00:00:00	Delete Update

If the user types an invalid Task ID in the url, it will return an error page with an error message.


Achilles
127.0.0.1:5000/update/-102
127.0.0.1:5000/update/-102
127.0.0.1:5000/update/-102 - Bing 搜尋
篩選您的搜尋:
歷程記錄
我的最愛
索引標籤

ICT SBA report

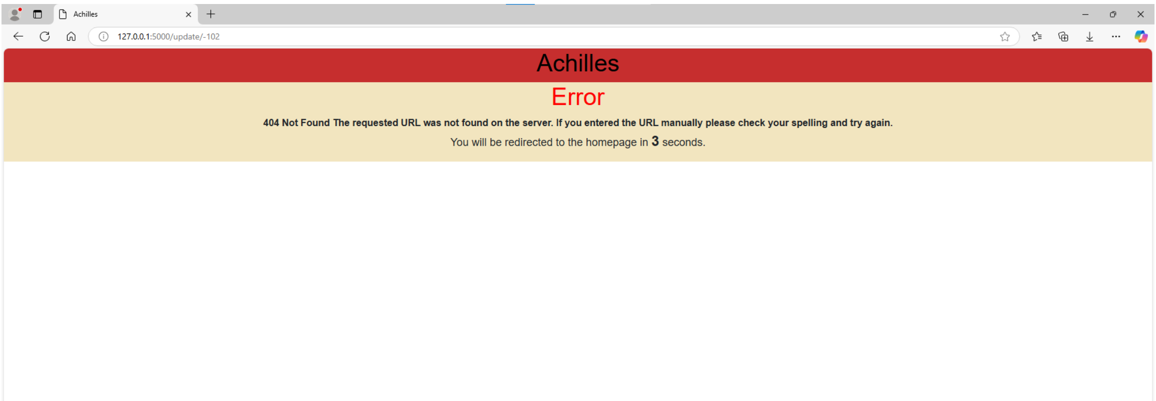
Details

The report requires me to build a system with python and database. Oh yeah, I have decided to use Latex to write the report at home. Luckily, I can submit the report a week later.

Update Task

Name

ICT SBA report



Achilles

Error

404 Not Found The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.

You will be redirected to the homepage in **3** seconds.

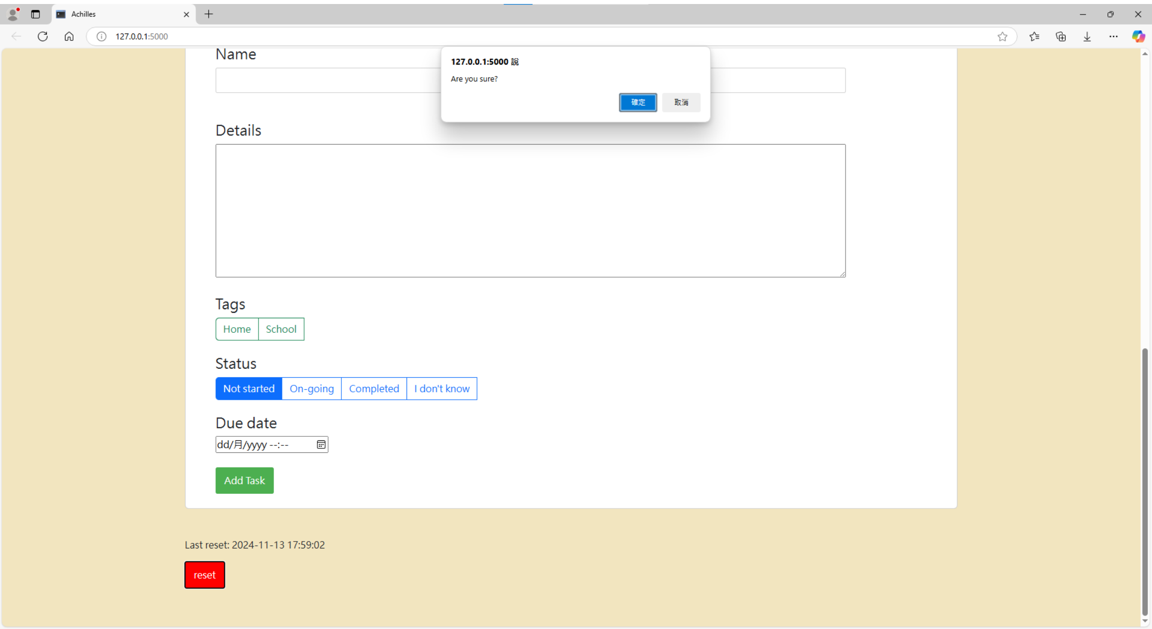
Source code:

```python
def UpdateGet(id):
    item = raww(f"SELECT * FROM Tasks WHERE TID = {id}")
    status = raww("SELECT * FROM Statuses")
    tags = raww("SELECT * FROM Tags ORDER BY Name")
    tt = raww("SELECT * FROM TaskTags")

    # Data validation
    if len(item) == 0:
        return render_template('\textbf{error.html}', s="Task ID invalid")

    item = item[0]
    return render_template('\textbf{update.html}', item=item, status=status, tags=tags,tt=tt)
```

4.3.8 data processing

The tasks are sorted by increasing TID, which is sorted by the task creation time.

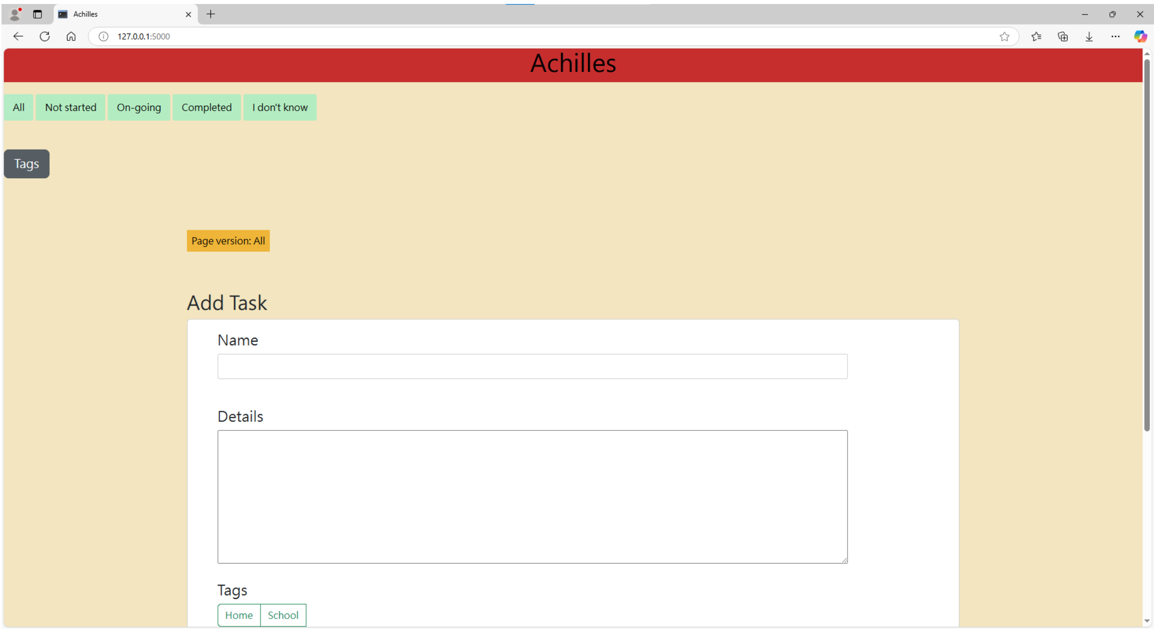
Users can reset everything in the application to the original state if they want to. Simply click the "reset" button in the home page and and click "Yes" in the confirmation pop up. Then everything will be reset.



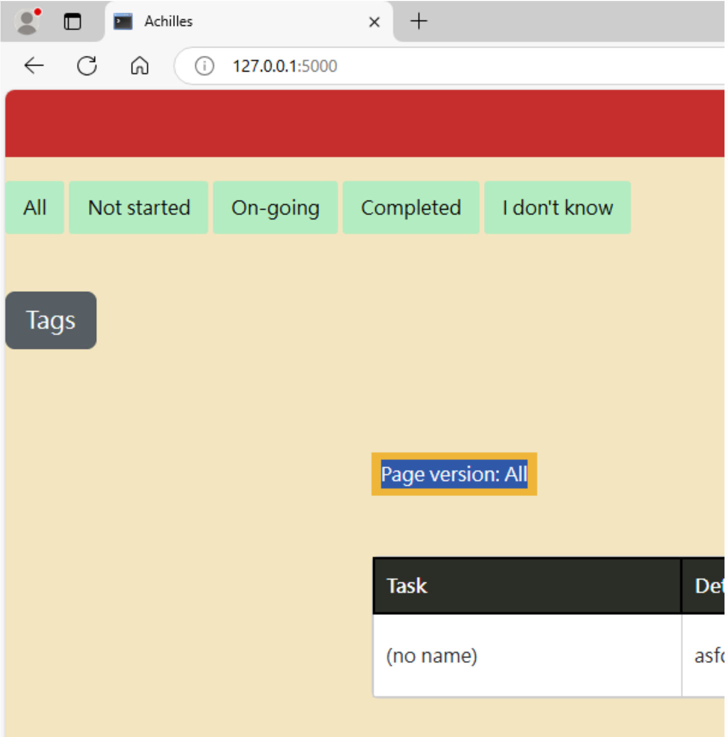
127.0.0.1:5000 說

Are you sure?

Last reset: 2024-11-13 17:59:02



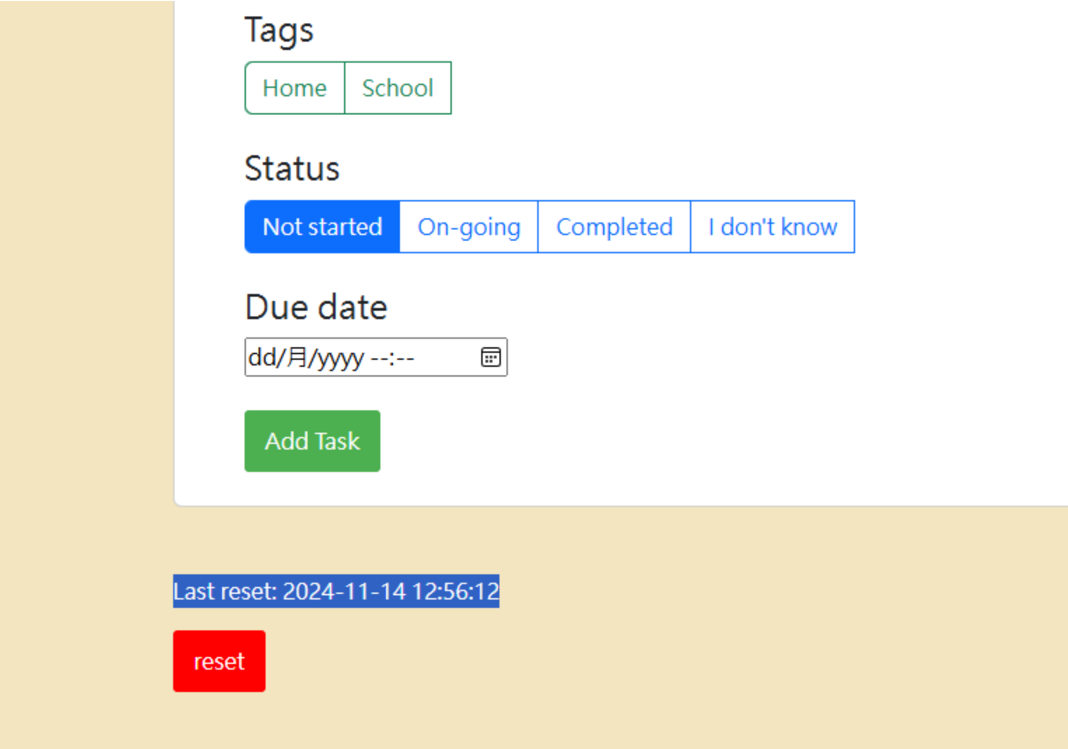
The index page has a block indicating the current version the main page:



Page version: All

Task	Det
(no name)	asf

The application will show the date and time of last reset by retrieving values stored in **data.json**

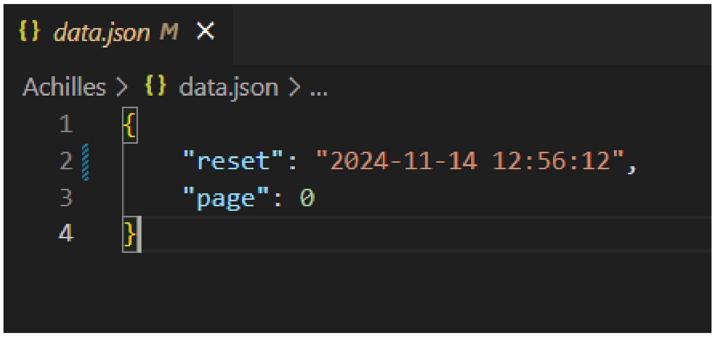


Last reset: 2024-11-14 12:56:12

Source code (**app.py**):

```python
def rjson():
    with open("data.json", "r") as f:
        data = json.load(f)
    return data

def wjson(data):
    with open("data.json", "w") as f:
        json.dump(data,f,indent=4)

def IndexGet():
    # some codes here ...
    data = rjson()
    last = data["reset"]
    page = data["page"]
    ver = version[page]
    return render_template('index.html', last=last, page=page, ver=ver) # some parameters are hidden here to show the data processing part

def IndexPost(response):
    if 'reset' in response:
        data = rjson()
        data["reset"] = str(datetime.now(pytz.timezone('Asia/
    Hong_Kong')).strftime("%Y-%m-%d %H:%M:%S"))
        wjson(data)
        reset()
        return redirect('/')
    if 'page' in response:
        data = rjson()
        c = response['page']
        for i in range(len(version)):
            if version[i] == c:
                num = i
                break
        data["page"] = num
        wjson(data)
        return redirect('/')
    # other codes below are hidden here to show the data processing
     part
```

In **data.json**:

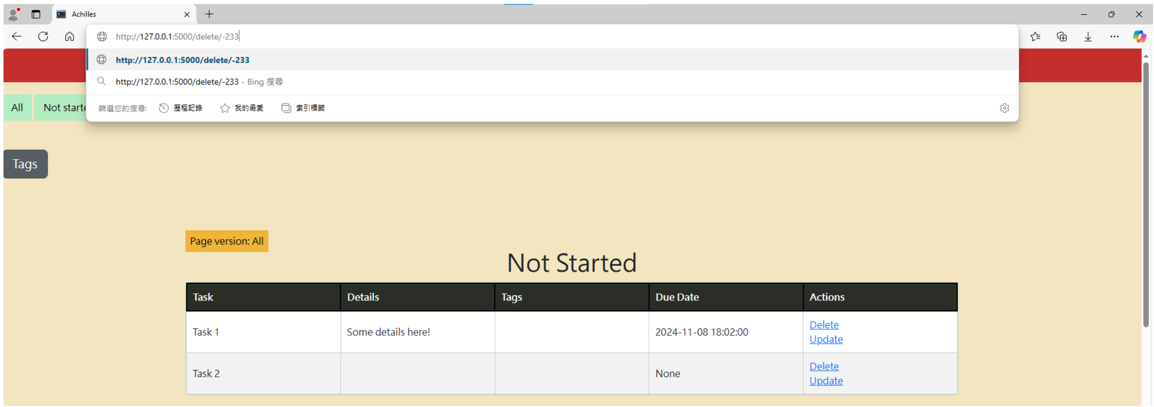


```
{
    "reset": "2024-11-14 12:56:12",
    "page": 0
}
```

4.3.9 error handling

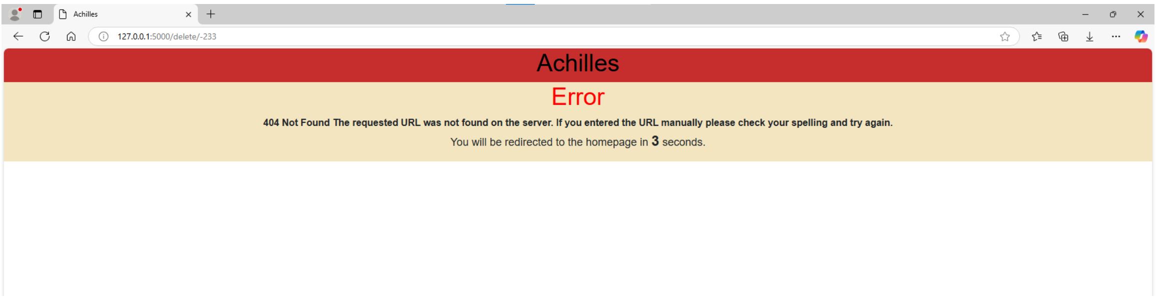
If there is an error in url, the application return an error page with error message, then redirect to the home index page

E.g. user input wrong url link:


Achilles
http://127.0.0.1:5000/delete/-233
http://127.0.0.1:5000/delete/-233
http://127.0.0.1:5000/delete/-233 - Bing 搜尋
篩選您的搜尋: 歷程記錄 我的最愛 索引標籤
All
Not starte
Tags
Page version: All

Not Started

Task	Details	Tags	Due Date	Actions
Task 1	Some details here!		2024-11-08 18:02:00	Delete Update
Task 2			None	Delete Update



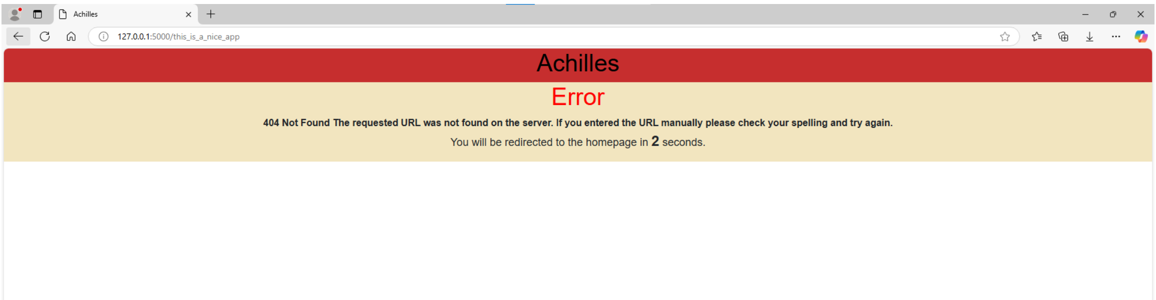
Achilles

Error

404 Not Found The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.

You will be redirected to the homepage in 3 seconds.

Same as above:



Achilles

Error

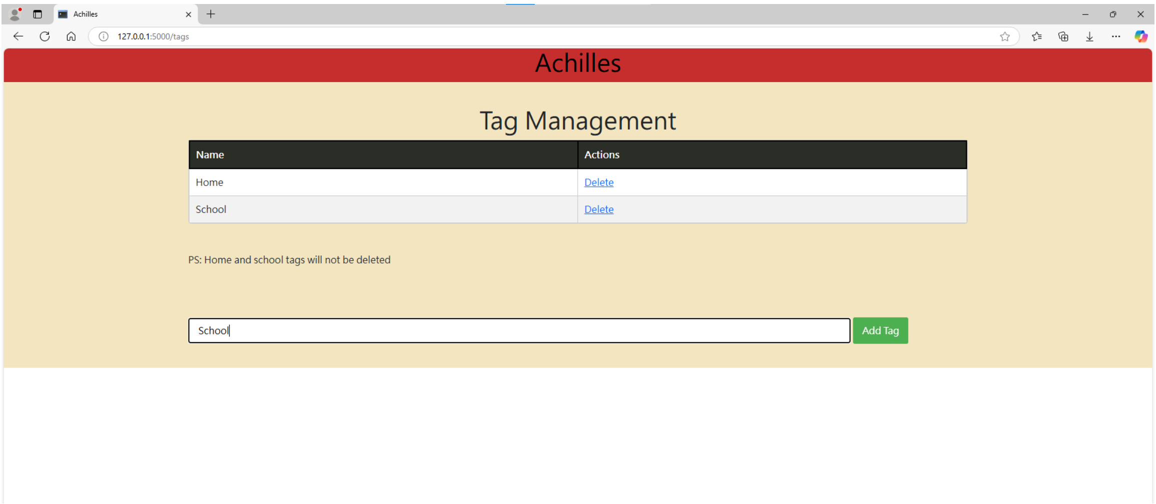
404 Not Found The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.

You will be redirected to the homepage in 2 seconds.

Source code:

```
def error(s):
    return render_template('error.html', s=s)

@app.errorhandler(HTTPException)
def handleError(err):
    return error(str(err.code)+" "+err.name+" "+err.description)
```

The application can handle the error in tag creation, which forces users cannot input tags with NULL or repeated name



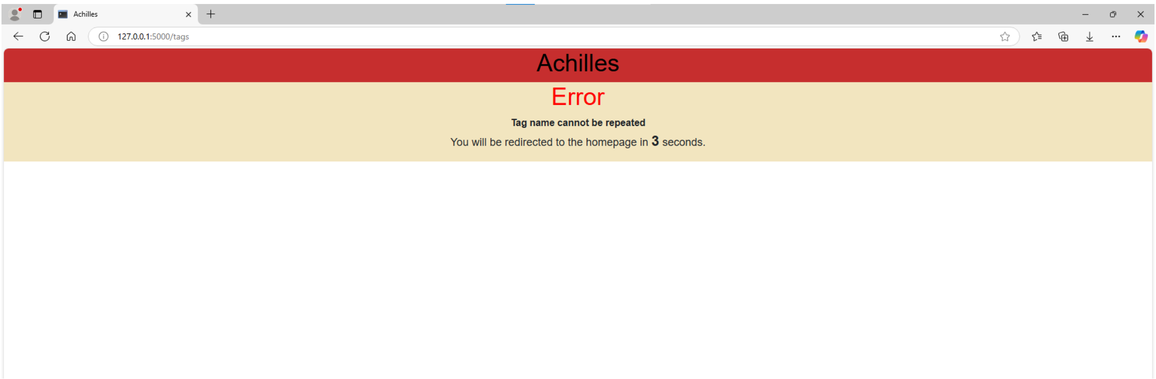
Achilles

Tag Management

Name	Actions
Home	Delete
School	Delete

PS: Home and school tags will not be deleted

School

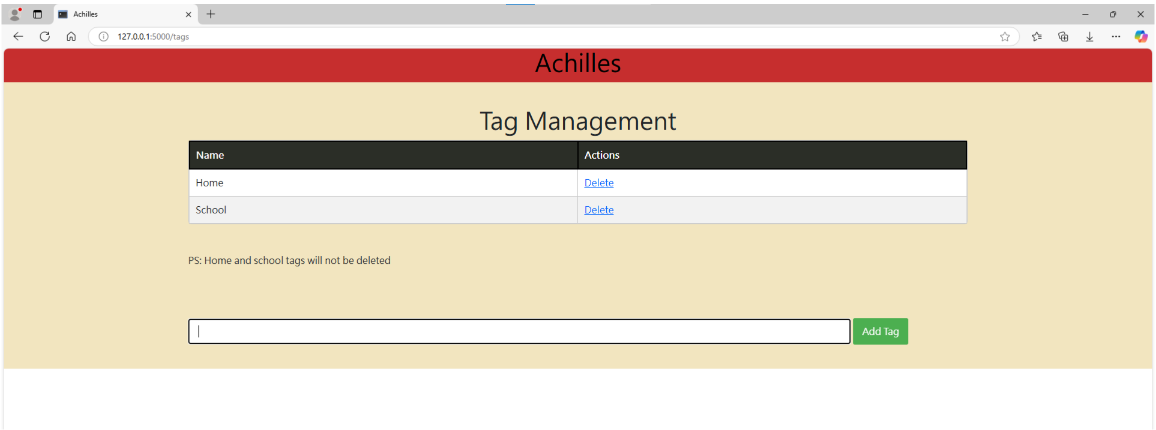


Achilles

Error

Tag name cannot be repeated

You will be redirected to the homepage in **3** seconds.

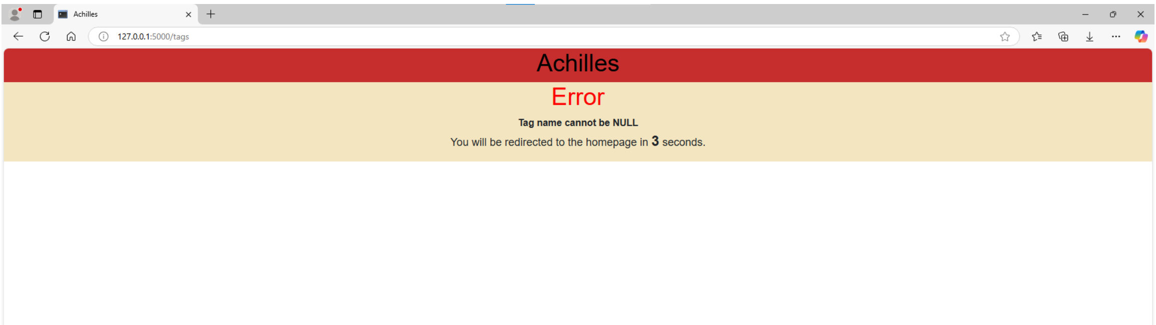


Achilles

Tag Management

Name	Actions
Home	Delete
School	Delete

PS: Home and school tags will not be deleted



Achilles

Error

Tag name cannot be NULL

You will be redirected to the homepage in **3** seconds.

Users cannot delete a tag that is currently assigned to any task. Tags can only be deleted when they are no longer associated with any tasks

Source code:

```python
def TagPost(response):
    if response['tagname'] == "":
        return error("Tag name cannot be NULL")
    try:
        cur.execute("INSERT INTO Tags (Name) VALUES (?)", (response['tagname'],))
        con.commit()
        return redirect('/tags')
    except:
        return error("Tag name cannot be repeated")
```

4.3.10 Portability

Currently, users can only host application in their home server. The advantage is no internet connection is required to use the application. The disadvantage is users cannot access the application when they are not at home

4.4 Functions

1. add tasks
2. list all tasks
3. list uncompleted tasks
4. list completed tasks
5. list not started tasks
6. list on-going tasks
7. list tasks which users don't know the status
8. set tasks as completed
9. set tasks as not started
10. set tasks as on-going
11. set tasks as unknown status
12. update tasks
13. delete tasks
14. add tags
15. delete tags
16. add due date
17. update due date
18. reset all tasks
19. reset confirmation
20. notify users for coming tasks
21. tasks and tags deletion confirmation

5 Testing

5.1 Database

5.1.1 pros and cons of the database design

Originally, it is planned to implement user accounts to support multi-user workspaces. However, this would increase storage requirements for an additional user table and potentially raise data security risks if proper access controls aren't in place. Therefore, single-user approach is used in the final application.

- Multi-user database design
  - Pros
    - Allow collaborating with different users
    - Protect data privacy as login is required if users want to access the application through Internet
  - Cons
    - Management cost increases
    - More time and cost in data security measures are required to protect data privacy
    - More storage space is required to store users information
    - Users are required to memorize login username, password and email which make access the application more difficult
    - Higher power consumption
- Single-user database design
  - Pros
    - Lower management cost
    - Less time and cost in data security measures
    - Easier application access process as login is not required
    - Lower power consumption
    - reduce data security risks
  - Cons
    - Multi-user collaboration is disabled
    - Data privacy cannot be protected if the application is accessible from Internet as there is no login
  - Solution to protect data privacy if the application is accessible from Internet
    - Adding a pass-code to ensure only the authorized users can access the application

5.1.2 concepts of relational database

5.1.3 database security

For all SQL queries involving user text entries, queries with placeholders and separate parameters are used. The parameterized queries can prevent SQL injection so as to safeguard database security.
Format:

```
cur.execute(<query>, <parameters>)
```

Source code:

```python
def IndexPost(response):
    # some codes hidden here
    para = ()
    if len(date) == 0:
        para = (cont, det, opt)
    else:
        para = (cont, det, opt, datetime.strptime(date, "%Y-%m-%dT%H:%M"))
    if len(date) == 0:
        cur.execute("INSERT INTO Tasks (Name, Details, SID) VALUES (?, ?, ?)", para)
    else:
        cur.execute("INSERT INTO Tasks (Name, Details, SID, DueDate) VALUES (?, ?, ?, ?)", para)
    con.commit()

def UpdatePost(response, id):
    # some codes hidden here
    gg = ()
    if len(date) == 0:
        gg = (cont, det, opt, id)
        cur.execute("UPDATE Tasks SET Name = ?, Details = ?, SID = ? WHERE TID = ?", gg)
    else:
        gg = (cont, det, opt, datetime.strptime(date, "%Y-%m-%dT%H:%M"), id)
        cur.execute("UPDATE Tasks SET Name = ?, Details = ?, SID = ?, DueDate = ? WHERE TID = ?", gg)
    cur.execute("DELETE FROM TaskTags WHERE TaskID = ?", (id,))
    for i in response:
        if 'tag' in i:
            cur.execute("INSERT INTO TaskTags (TaskID, TagID) VALUES (?, ?)",(id, response[i]))
    con.commit()

def TagPost(response):
    if response['tagname'] == "":
        return error("Tag name cannot be NULL")
    try:
        cur.execute("INSERT INTO Tags (Name) VALUES (?)", (response['tagname'],))
        con.commit()
        # some codes hidden here
```

5.1.4 data privacy issues

As the application will be hosted in home server and it is designed as single-user, entering pass-code is the simplest way to protect data privacy, though this application does not set a pass-code

5.1.5 data validation

After several testing, it is proven that data validation of the application works. The result is same as **Section 4.3.9**

No data verification is needed as fact-checking of the task information and the range check of the due date is unnecessary in this case

5.1.6 needs and procedures of denormalisation

As the current database is in Third Normal Form, it offers high data integrity, reduced data redundancy, and prevents anomalies like update, insertion, and deletion anomalies. Therefore, denormalization is not necessary if complex SQL queries can be properly managed.

5.2 Program

5.2.1 pros and cons of the program design

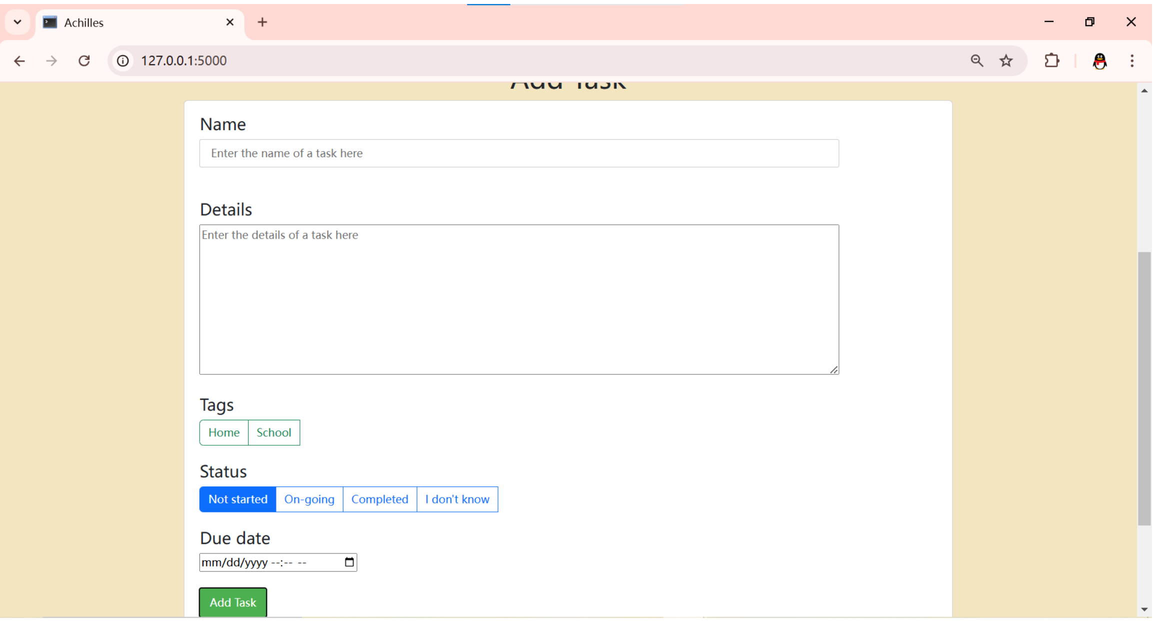
Instead of putting all functions into **app.py** file, interface functions and database execution functions are separated into **app.py** and **db.py** files. Also, the detailed procedures of the functions are

5.2.2 test data and test cases

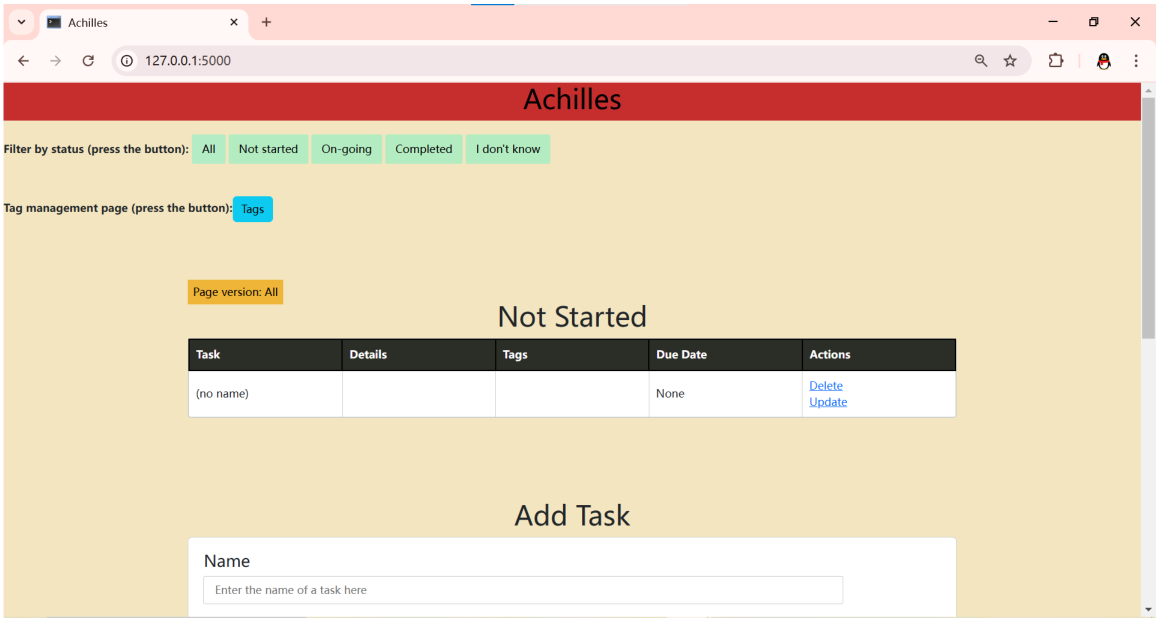
I changed the design of the UI so the interface may look different from the above

Test 1: Input an empty task

Test case:



Result:



Not Started

Task	Details	Tags	Due Date	Actions
(no name)			None	Delete Update

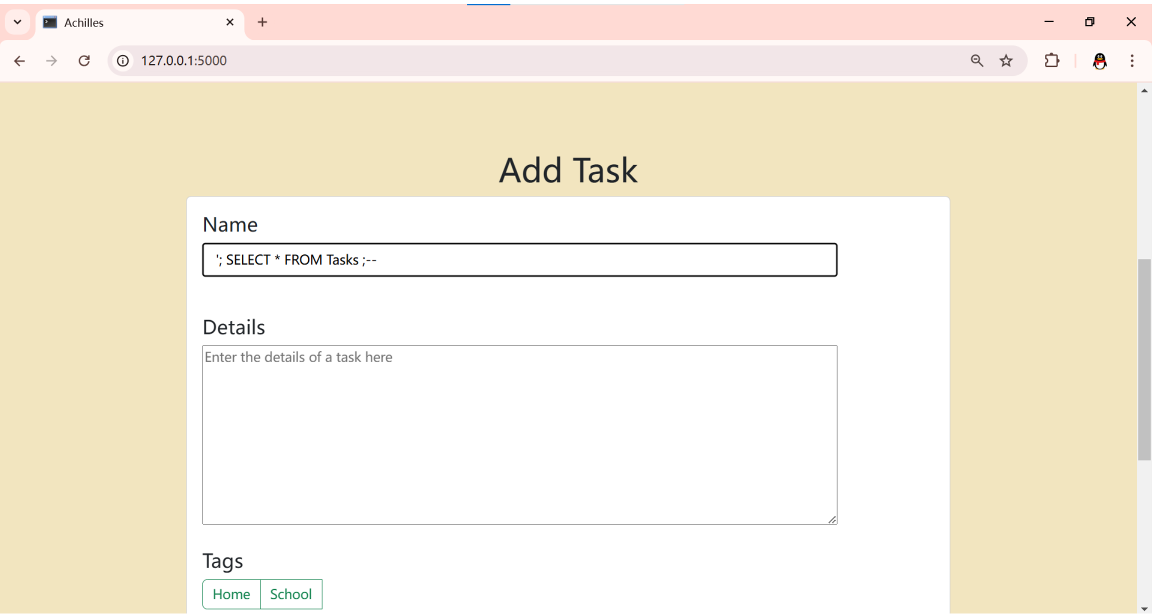
Add Task

Name

Enter the name of a task here

Test 2: SQL Injection

Test case:



Add Task

Name

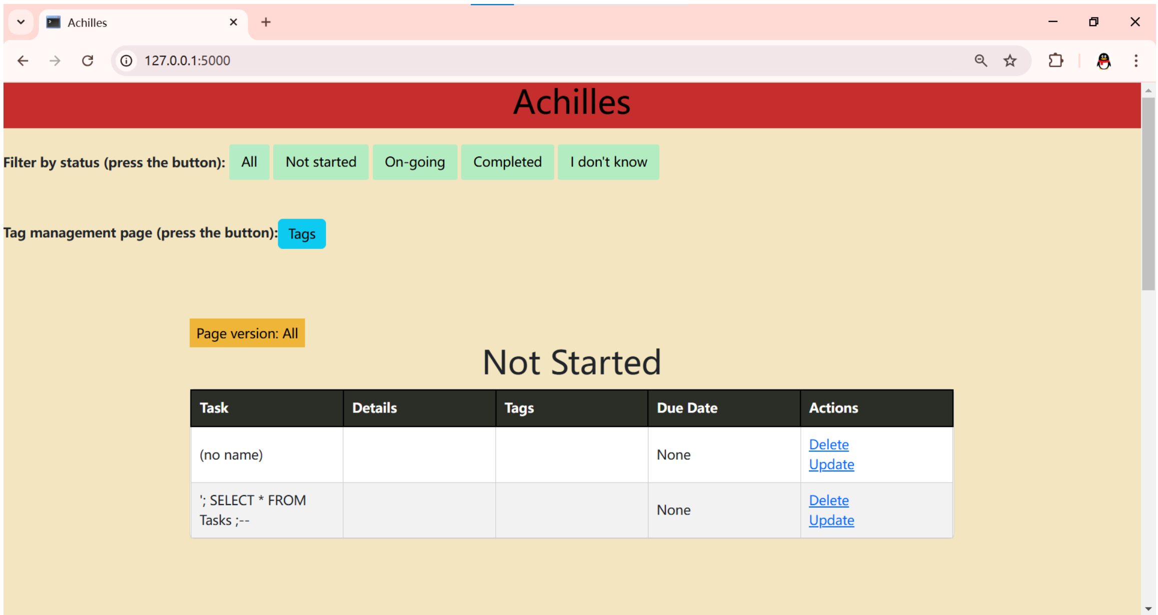
'; SELECT * FROM Tasks ;--

Details

Enter the details of a task here

Tags

Result:



Achilles

Filter by status (press the button): All | Not started | On-going | Completed | I don't know

Tag management page (press the button): Tags

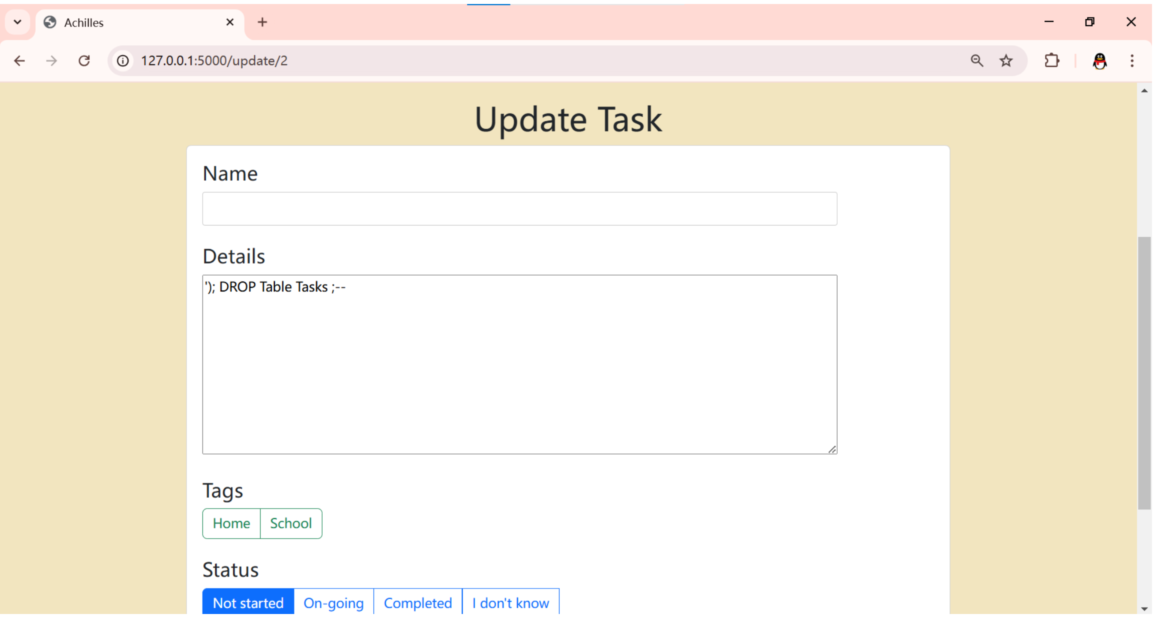
Page version: All

Not Started

Task	Details	Tags	Due Date	Actions
(no name)			None	Delete Update
'; SELECT * FROM Tasks ;--			None	Delete Update

Test 3: SQL Injection (Update page)

Test case:



Update Task

Name

Details

```
'); DROP Table Tasks ;--
```

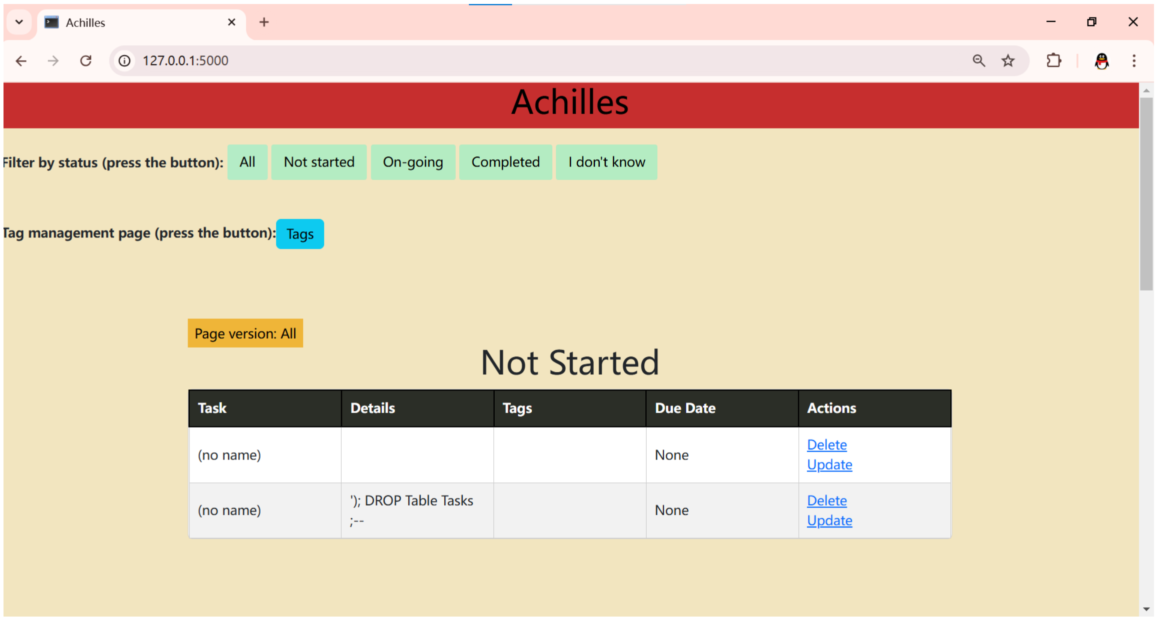
Tags

Home | School

Status

● Not started | On-going | Completed | I don't know

Result:



Achilles

Filter by status (press the button): All | Not started | On-going | Completed | I don't know

Tag management page (press the button): Tags

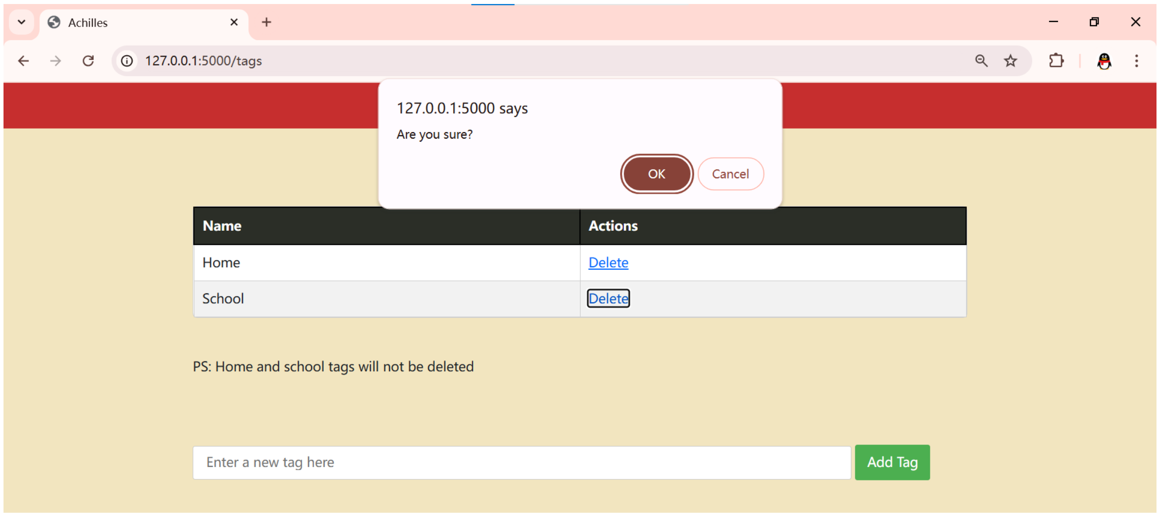
Page version: All

Not Started

Task	Details	Tags	Due Date	Actions
(no name)			None	Delete Update
(no name)	'); DROP Table Tasks ;--		None	Delete Update

Test 4: Delete default tag

Test case:



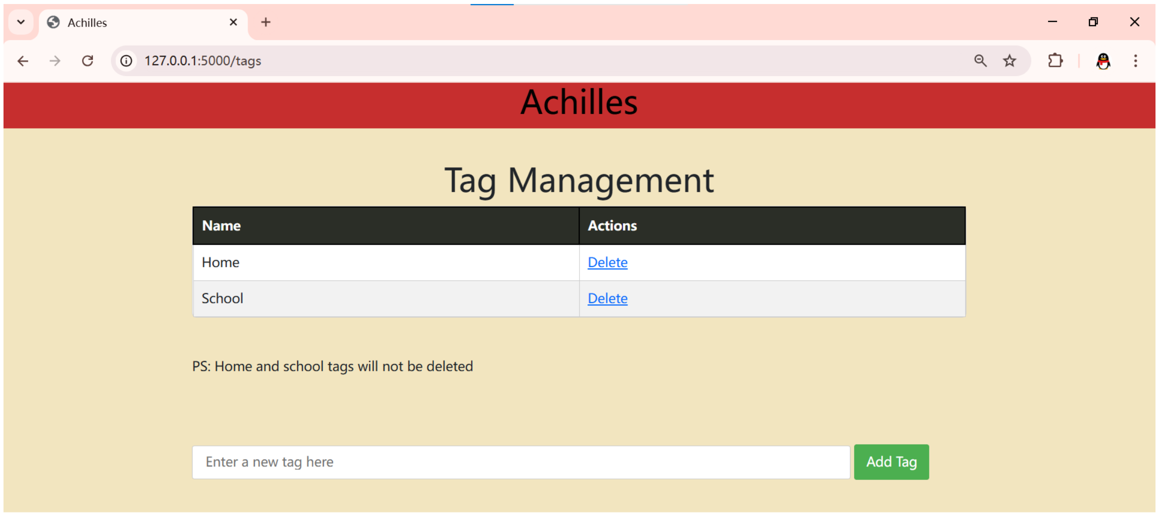
127.0.0.1:5000 says

Are you sure?

Name	Actions
Home	Delete
School	Delete

PS: Home and school tags will not be deleted

Result:



Achilles

Tag Management

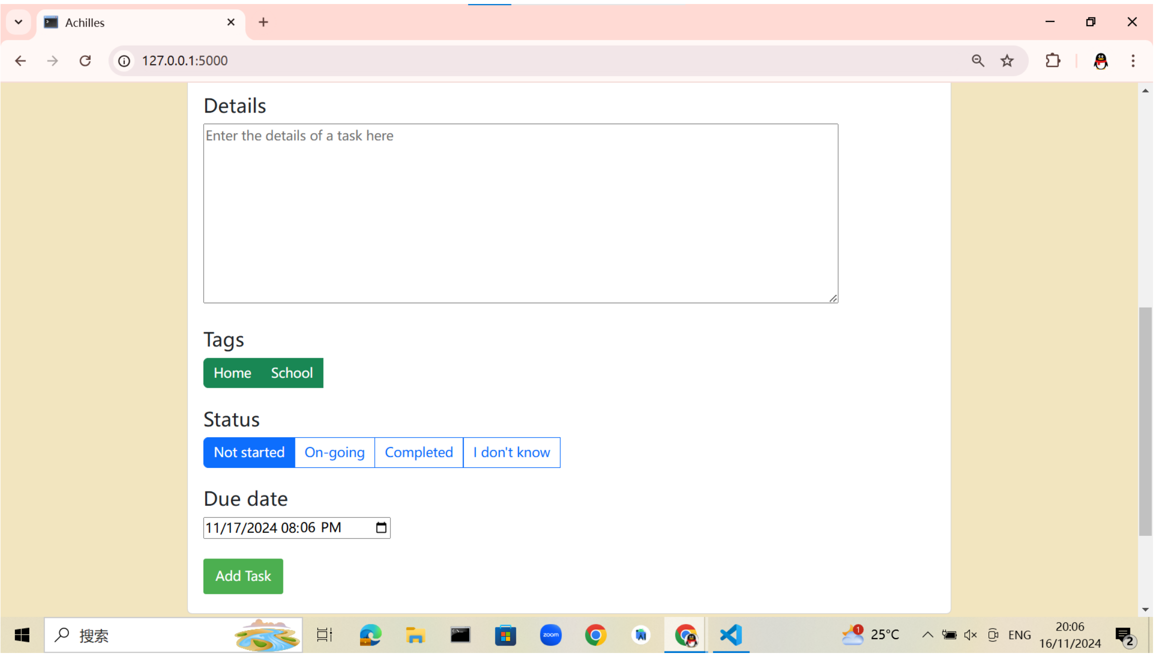
Name	Actions
Home	Delete
School	Delete

PS: Home and school tags will not be deleted

5.2.3 unit test

Test 1: Notification of coming tasks

Test case:



Details

Enter the details of a task here

Tags

Home School

Status

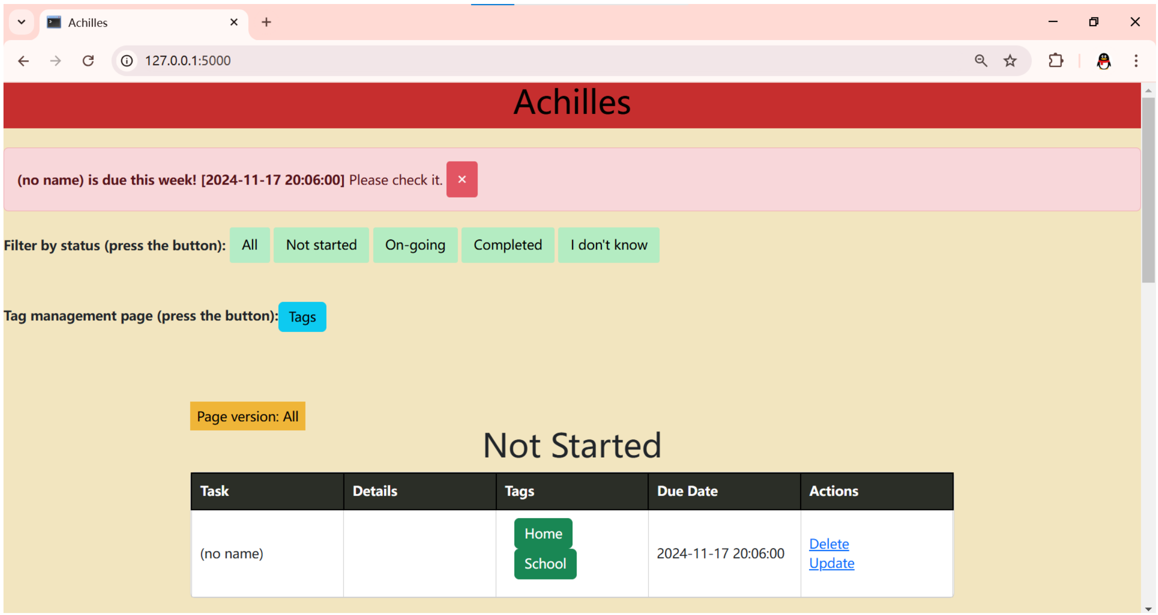
Not started On-going Completed I don't know

Due date

11/17/2024 08:06 PM

Add Task

Result:



Achilles

(no name) is due this week! [2024-11-17 20:06:00] Please check it. ×

Filter by status (press the button): All Not started On-going Completed I don't know

Tag management page (press the button): Tags

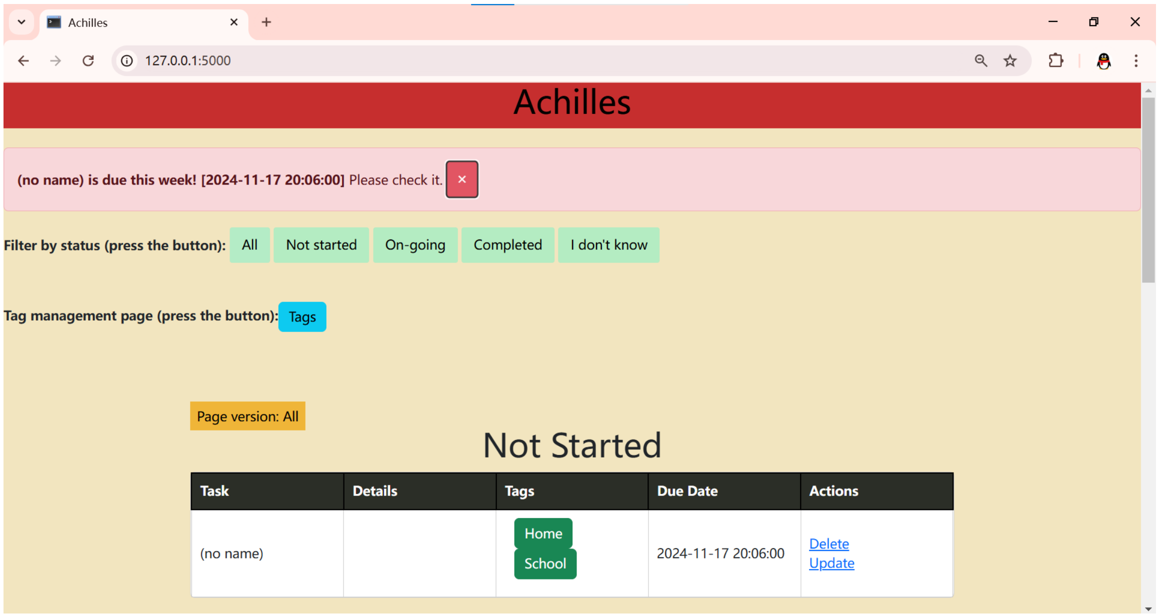
Page version: All

Not Started

Task	Details	Tags	Due Date	Actions
(no name)		Home School	2024-11-17 20:06:00	Delete Update

Test 2: Remove notification

Test case:



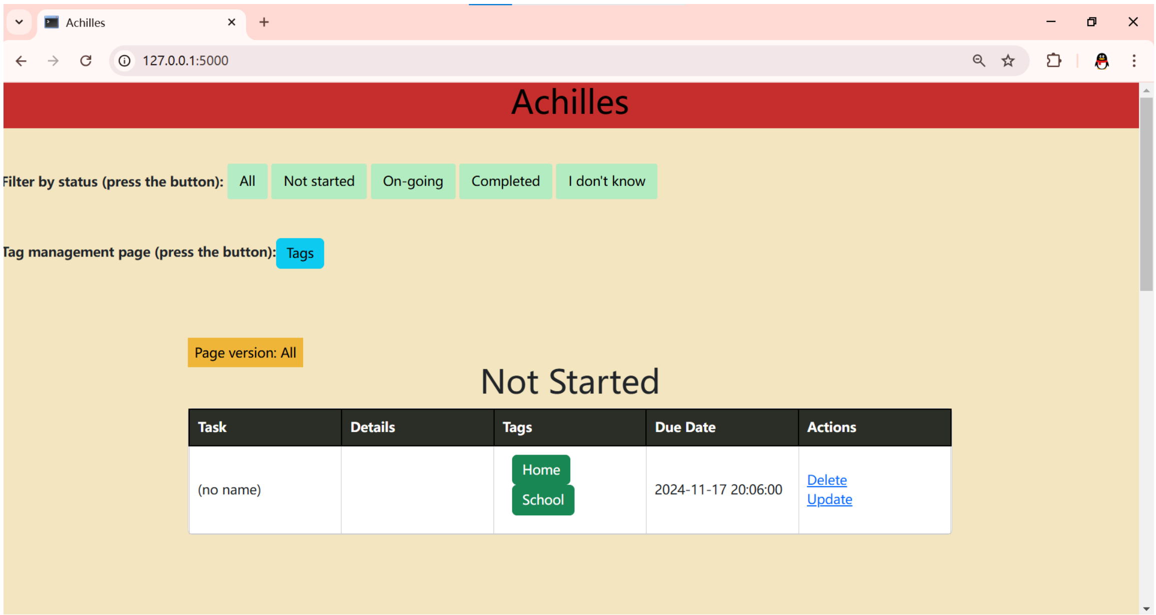
(no name) is due this week! [2024-11-17 20:06:00] Please check it.

×
Filter by status (press the button):
All
Not started
On-going
Completed
I don't know
Tag management page (press the button):
Tags
Page version: All

Not Started

Task	Details	Tags	Due Date	Actions
(no name)		Home School	2024-11-17 20:06:00	Delete Update

Result:

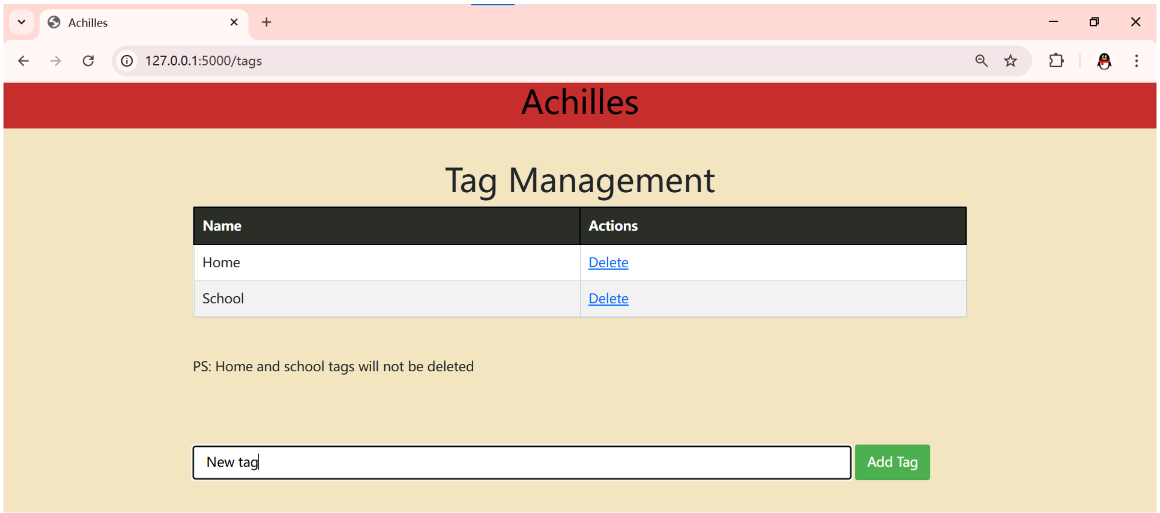


Not Started

Task	Details	Tags	Due Date	Actions
(no name)		Home School	2024-11-17 20:06:00	Delete Update

Test 3: Tag adding

Test case:

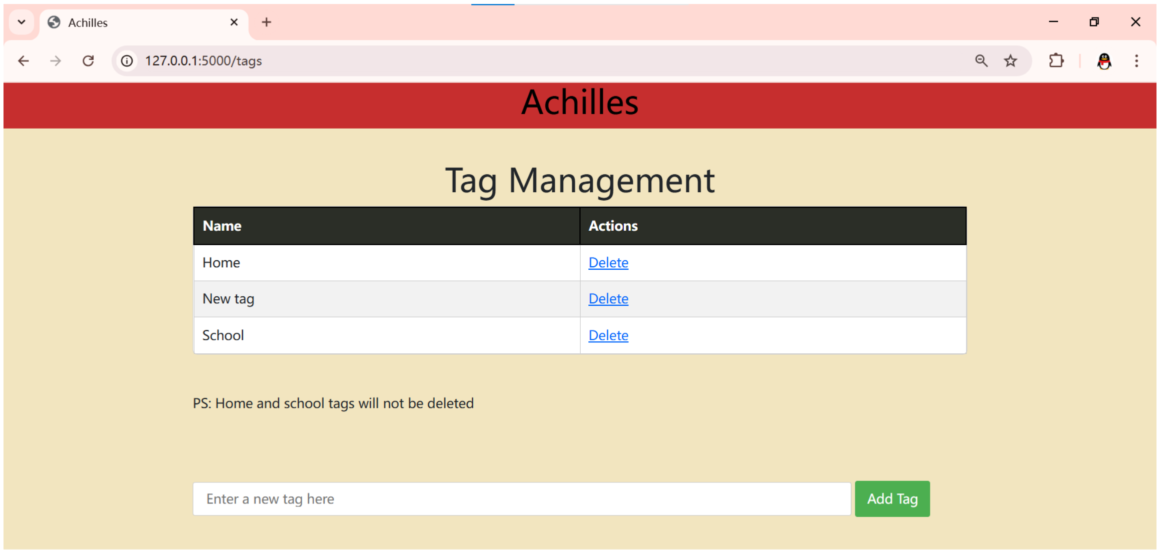


Tag Management

Name	Actions
Home	Delete
School	Delete

PS: Home and school tags will not be deleted

Result:



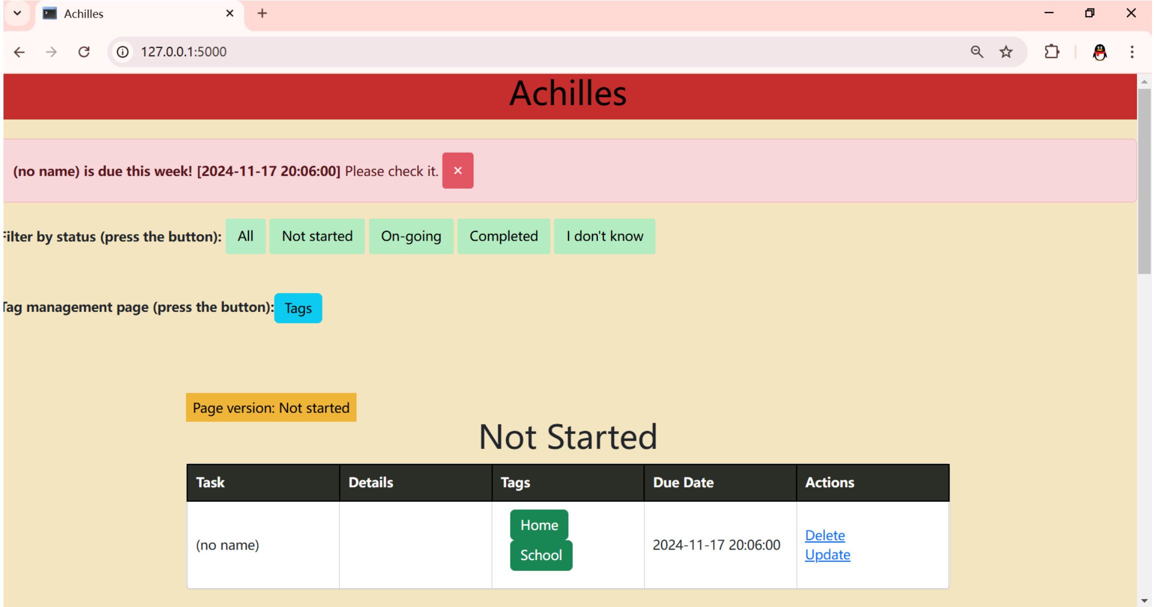
Tag Management

Name	Actions
Home	Delete
New tag	Delete
School	Delete

PS: Home and school tags will not be deleted

Test 4: Page version switching

Result:



Achilles

(no name) is due this week! [2024-11-17 20:06:00] Please check it. ×

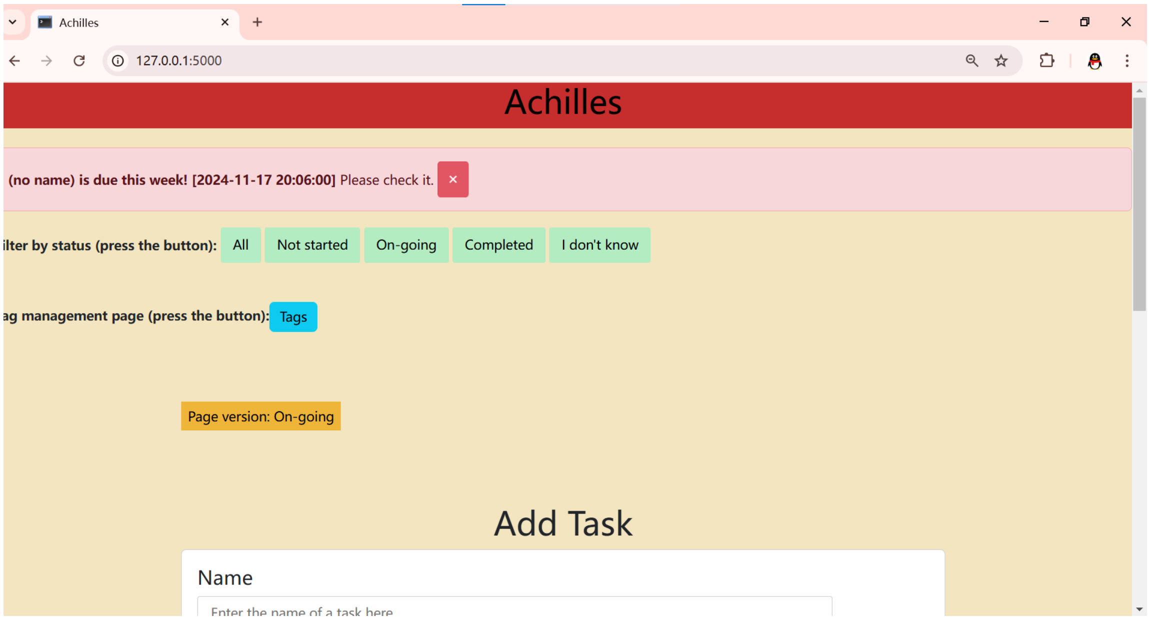
Filter by status (press the button):

Tag management page (press the button):

Page version: Not started

Not Started

Task	Details	Tags	Due Date	Actions
(no name)		Home School	2024-11-17 20:06:00	Delete Update



Achilles

(no name) is due this week! [2024-11-17 20:06:00] Please check it. ×

ilter by status (press the button):

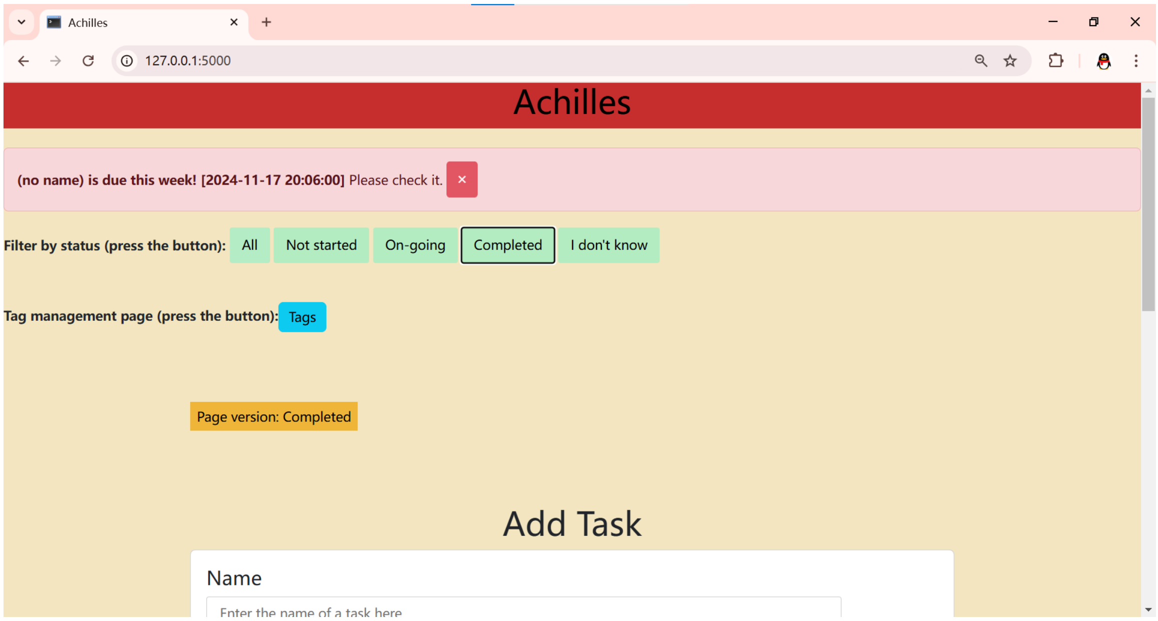
ag management page (press the button):

Page version: On-going

Add Task

Name

Enter the name of a task here

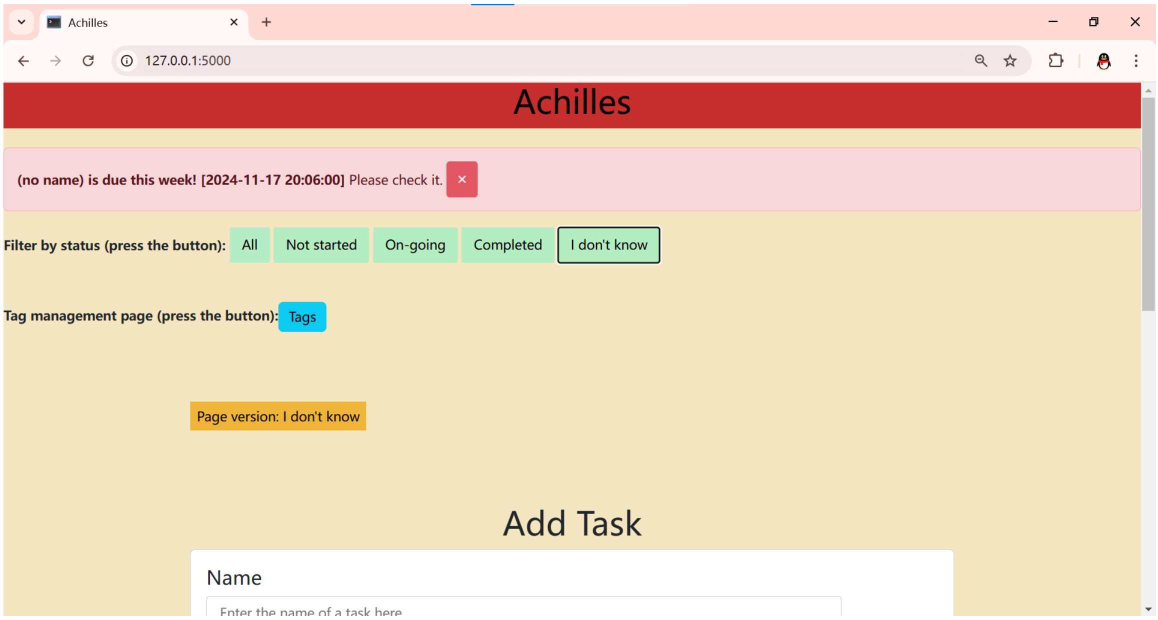

Achilles
127.0.0.1:5000
Achilles

(no name) is due this week! [2024-11-17 20:06:00] Please check it.

×
Filter by status (press the button):
All
Not started
On-going
Completed
I don't know
Tag management page (press the button):
Tags

Page version: Completed

Add Task
Name
Enter the name of a task here


Achilles
127.0.0.1:5000
Achilles

(no name) is due this week! [2024-11-17 20:06:00] Please check it.

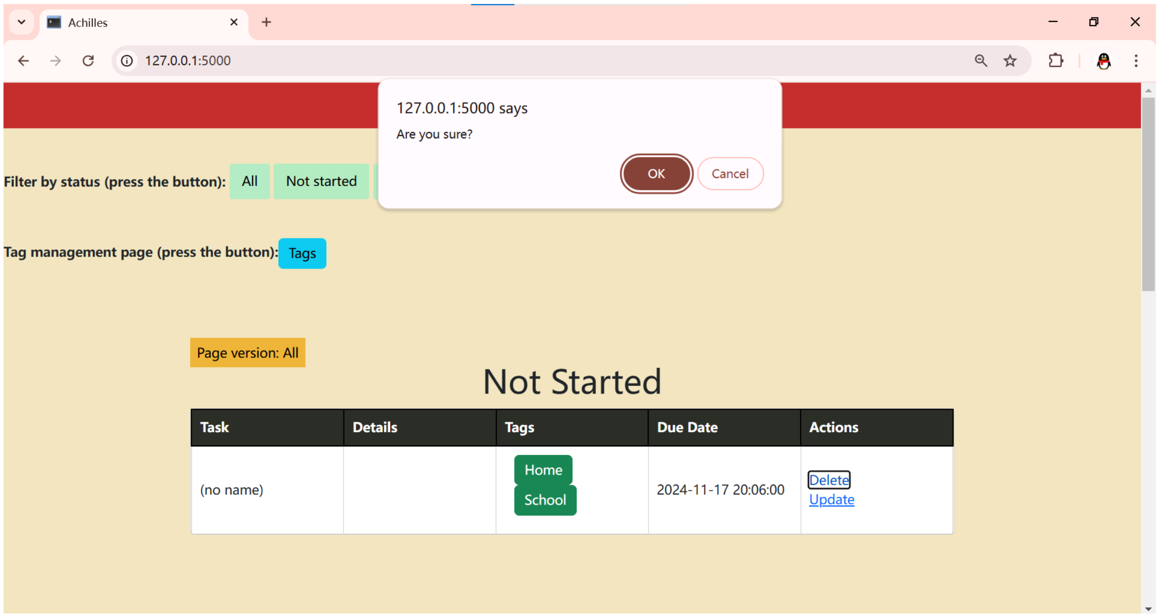
×
Filter by status (press the button):
All
Not started
On-going
Completed
I don't know
Tag management page (press the button):
Tags

Page version: I don't know

Add Task
Name
Enter the name of a task here

Test 5: Task deletion

Test case:


Achilles
127.0.0.1:5000
OK
Cancel
Filter by status (press the button):
All
Not started
Tag management page (press the button):
Tags
Page version: All

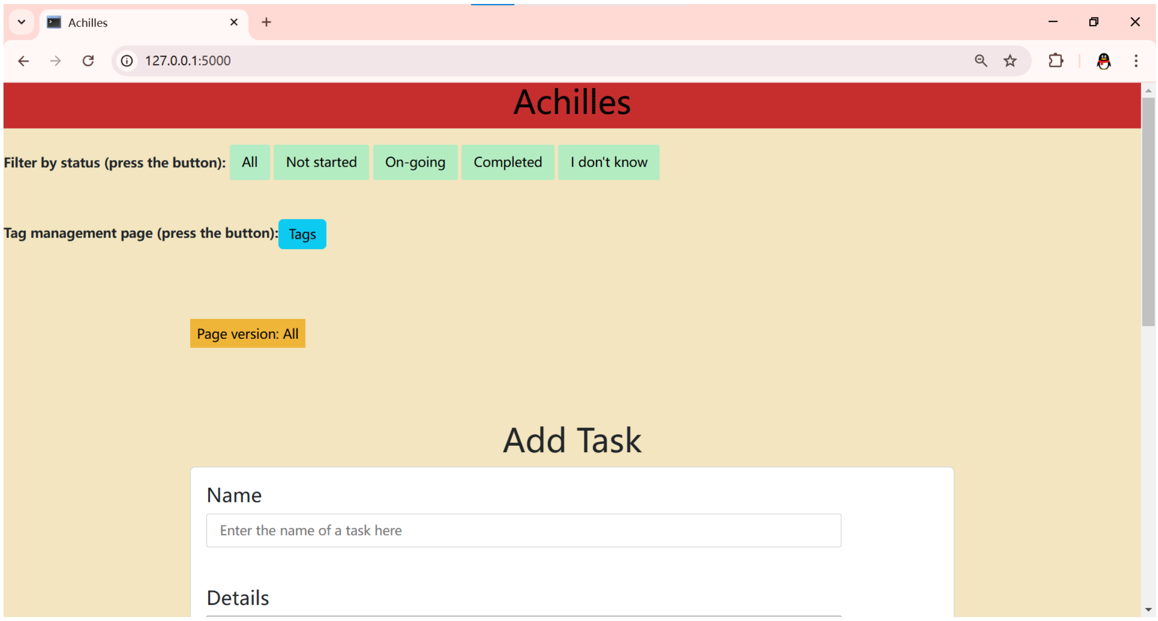
127.0.0.1:5000 says

Are you sure?

Not Started

Task	Details	Tags	Due Date	Actions
(no name)		Home School	2024-11-17 20:06:00	Delete Update

Result:



Add Task

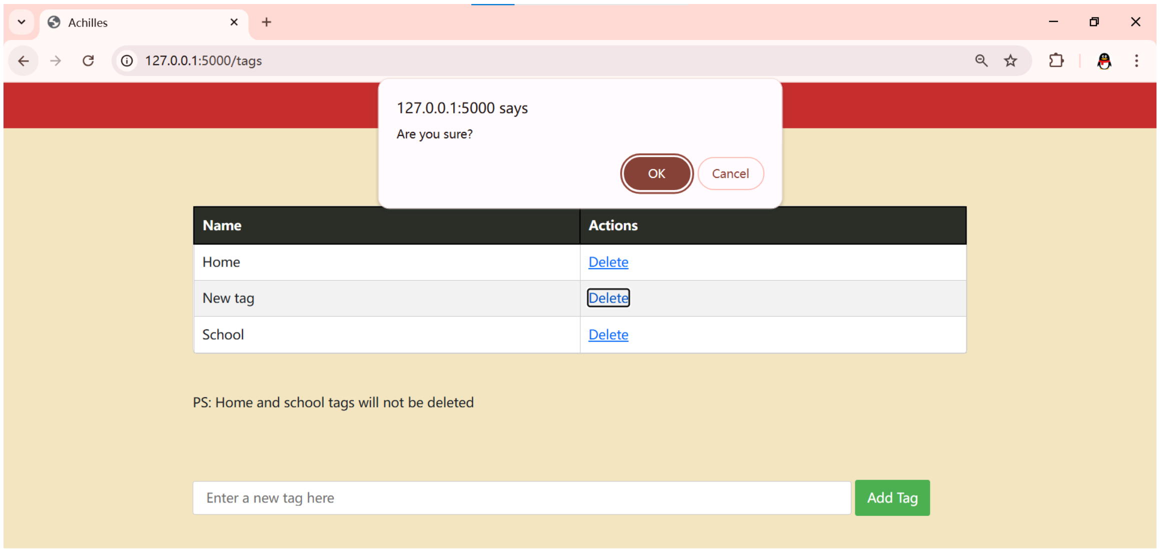
Name

Enter the name of a task here

Details

Test 6: Tag deletion

Test case:



127.0.0.1:5000 says

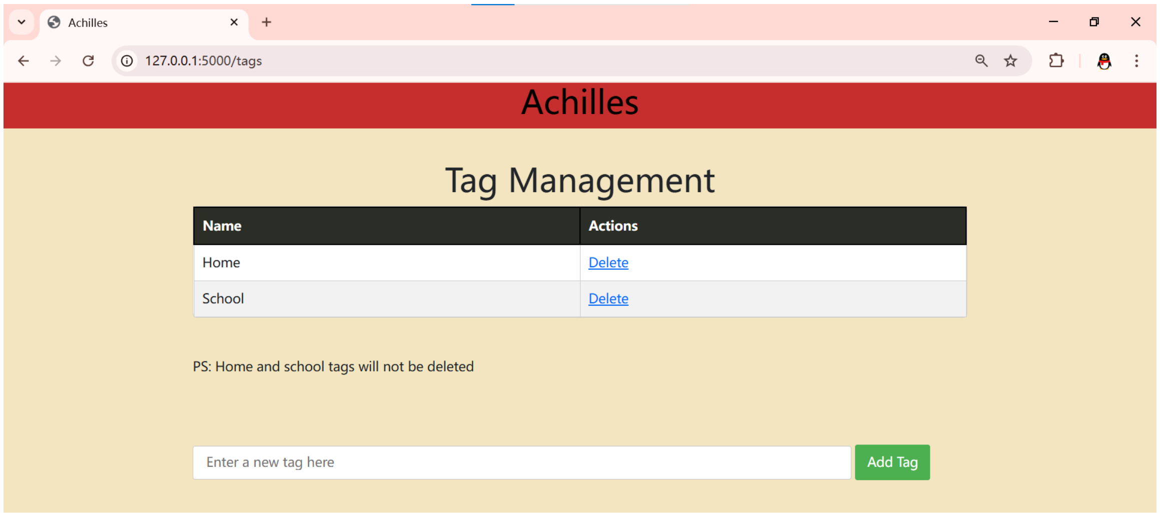
Are you sure?

Name	Actions
Home	Delete
New tag	Delete
School	Delete

PS: Home and school tags will not be deleted

Enter a new tag here

Result:



Achilles

Tag Management

Name	Actions
Home	Delete
School	Delete

PS: Home and school tags will not be deleted

Enter a new tag here

5.2.4 user acceptance test

Survey is conducted to collect the opinion from users. See **Section 6.1**

5.2.5 system test

From the **Section 5.2.2**, **Section 5.2.3** and **Section 5.2.4**, it is proven that the application meets all the functional and non-functional requirements mentioned in **Section 2.3.1** and **Section 2.3.1**

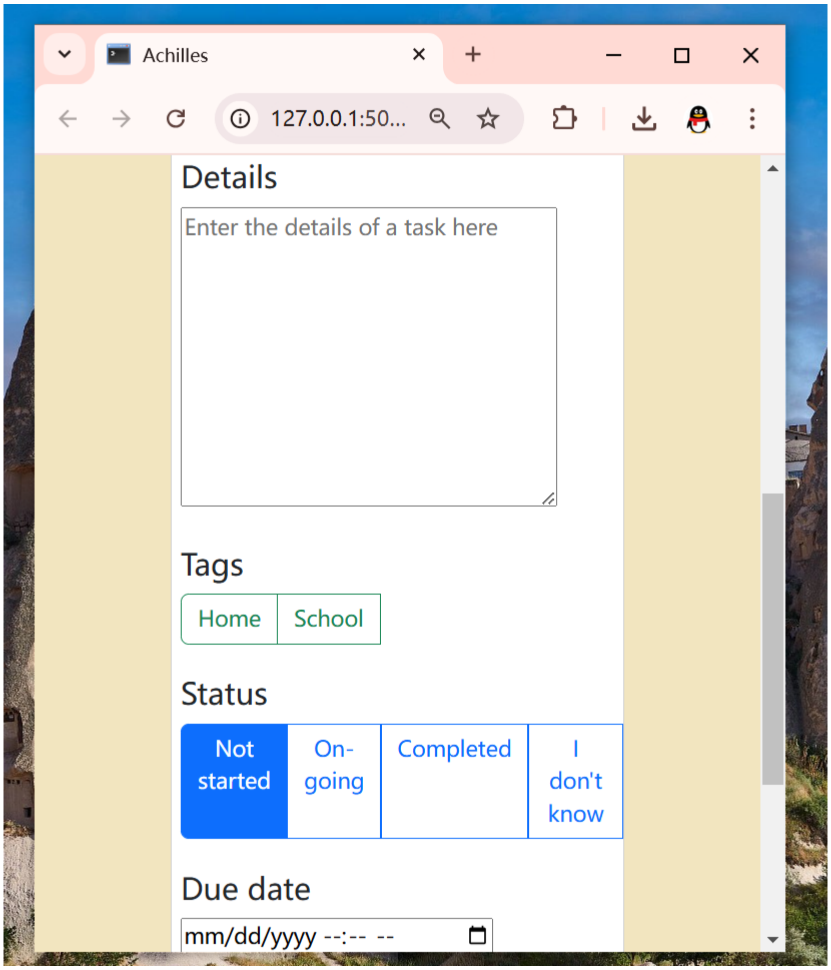
5.2.6 debugging

During developing this application, I have faced a lot of issues:

1. HTML elements
   - There are many HTML elements such as buttons, textarea which are out-of-syllabus of HKDSE, while they are very useful in web developing
2. JavaScript functions
   - As I have not learnt JavaScript before, while many web functions such as count-down timer requires knowledge of JavaScript. There are many bugs in the AI-generated JavaScript code and I spent a lot of time to debug and understand how it works so as to fix bugs.
3. SQLite table setting not working
   - From my experience, the SQLite table constraints such as default value setting may not work. I have to perform the constraint checking in the main program **app.py** instead.
4. Framework choosing
   - It takes me a lot of time to test which framework is the best. I chose Django originally but later on I found out that Django has a lot settings like account management to be made. In contrast, Flask enable me to host a new page by simply adding a function to the main program.
5. Typo
   - con.commit() vs cur.commit()

5.2.7 Responsive Design

Example 1:



Details

Enter the details of a task here

Tags

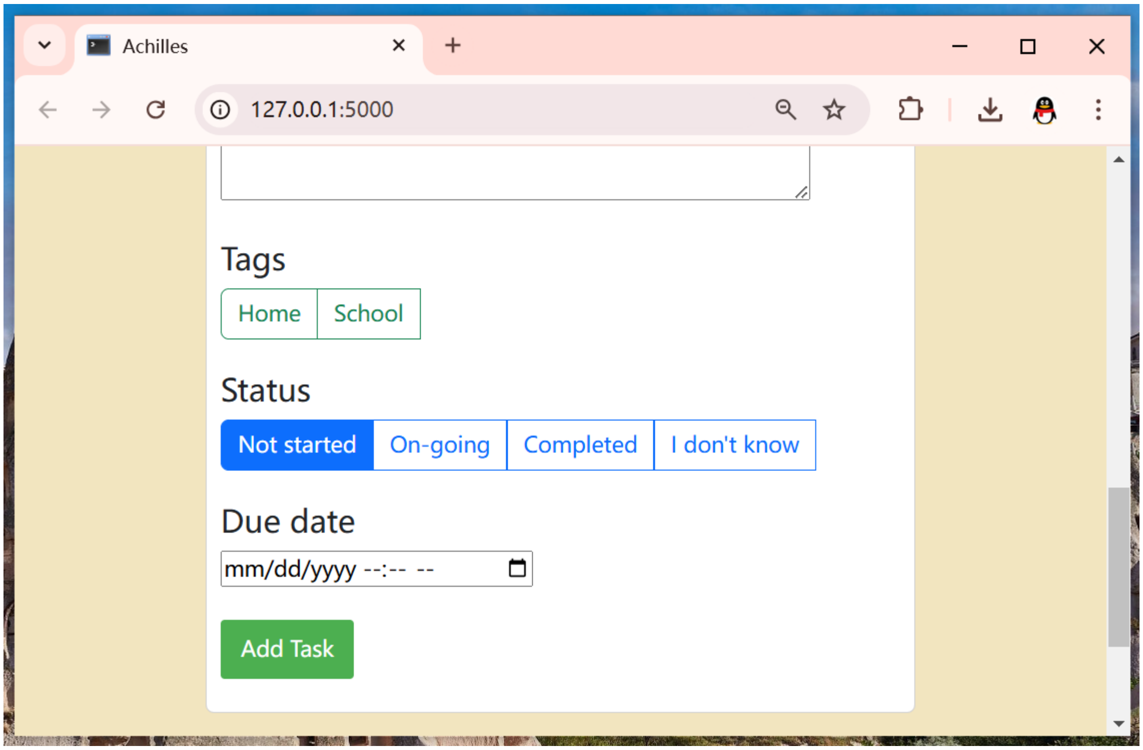
Home | School

Status

[✓ Not started] On-going | Completed | I don't know

Due date

mm/dd/yyyy --:-- --



Tags

Home School

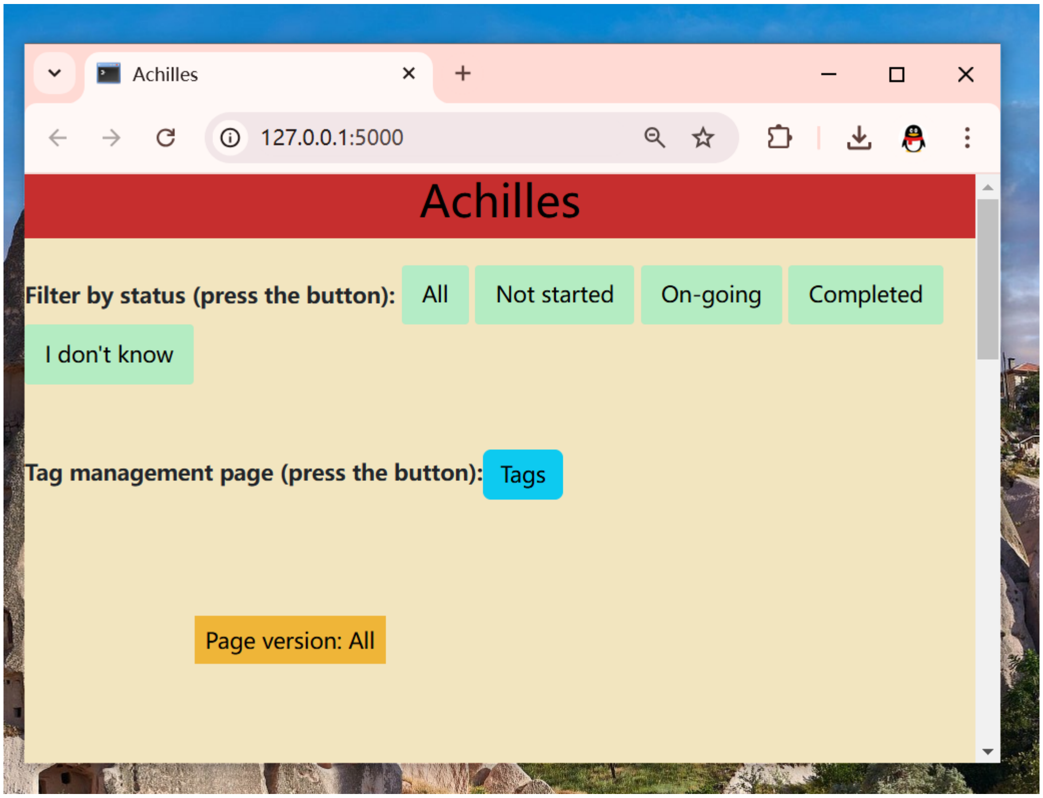
Status

[✓ Not started] On-going Completed I don't know

Due date

mm/dd/yyyy --:-- --

Example 2:

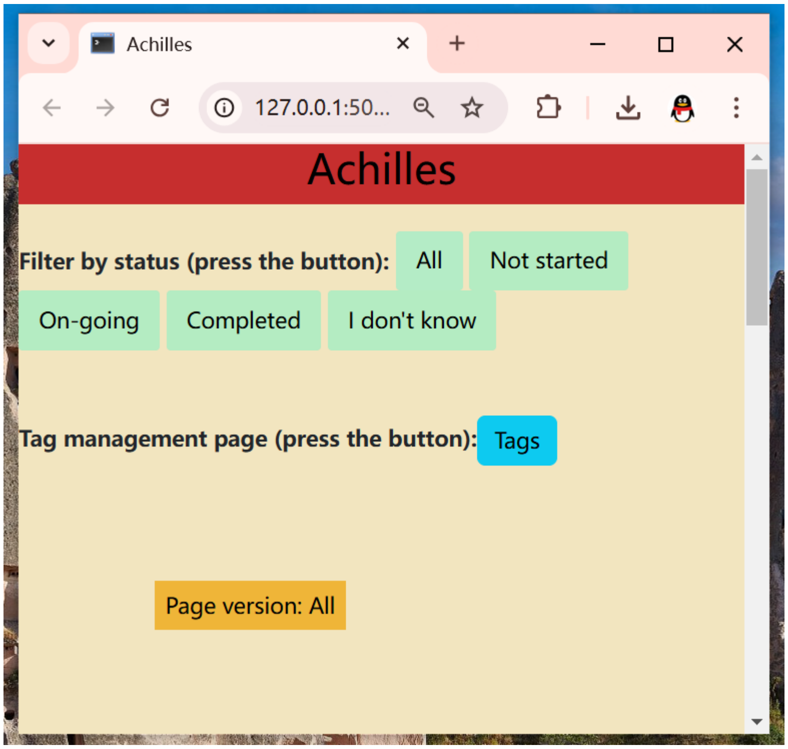


Achilles

Filter by status (press the button): All | Not started | On-going | Completed | I don't know

Tag management page (press the button): Tags

Page version: All



Achilles

Filter by status (press the button):

Tag management page (press the button):

6 Evaluation

Surveys are conducted for evaluation

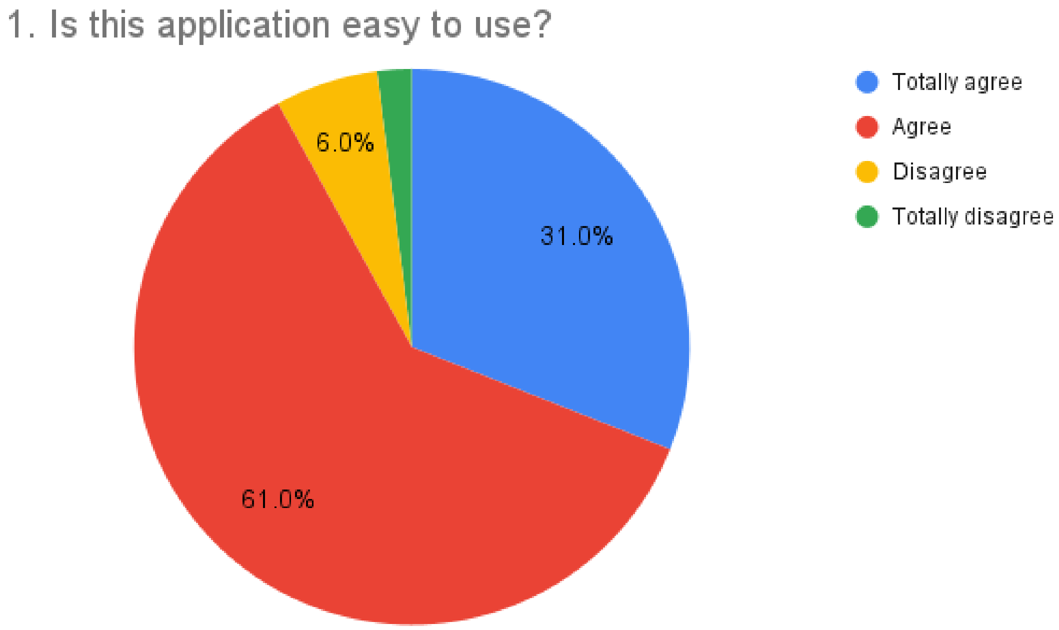
6.1 Feedback

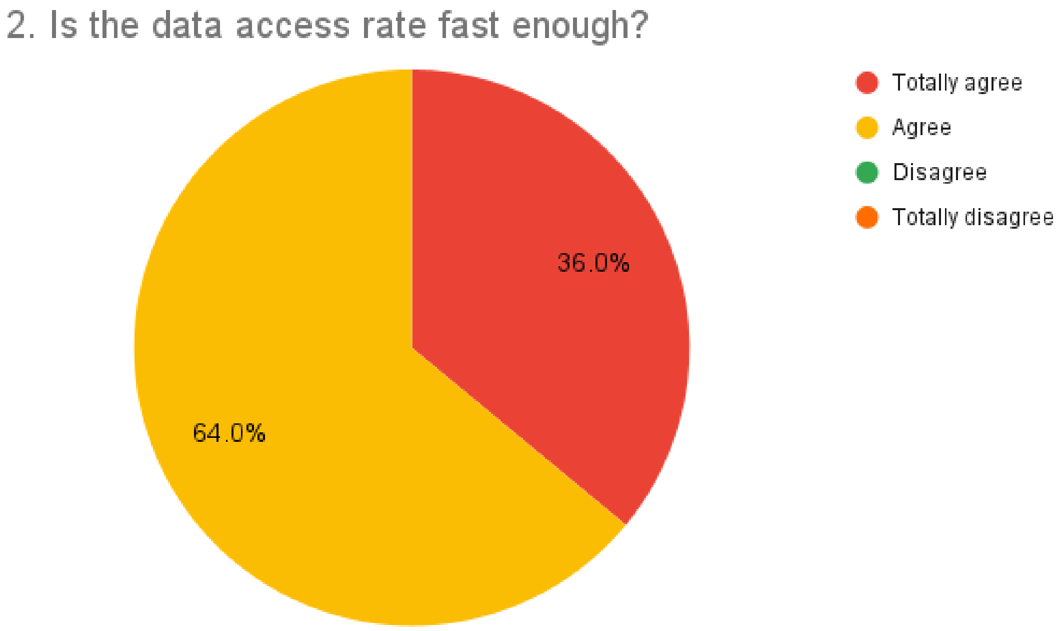
There are a few points of the feedback received through Google Form:

- Accessible from Internet
  - As most of the students do not have enough knowledge to setup the application, they want the application can be deployed to Internet so that they can access it simply entering the link and login information
- Make the tag button responsive in the index page
  - As some the students need to enter a tag name with a long length, they want the button element to be responsive
- Add a sort function on due date
  - The order of the tasks in the home tables may be messy, and users want to sort the tasks by due date for clarity
- Add scroll-to-top and scroll-to-bottom buttons
  - Users may add thousands of tasks in the application. When there are so many tasks in the table, they need to scroll through the page. Clicking a button to reach the desired part of the page would be more efficient
- Color
  - Some users prefer better font and background of higher color contrast
- Indentation
  - Some users want to maintain consistency on indentation over the pages (solved)
- Instruction
  - Some users do not know there is a button so words like ”filter by status (press the button)” are needed (solved)

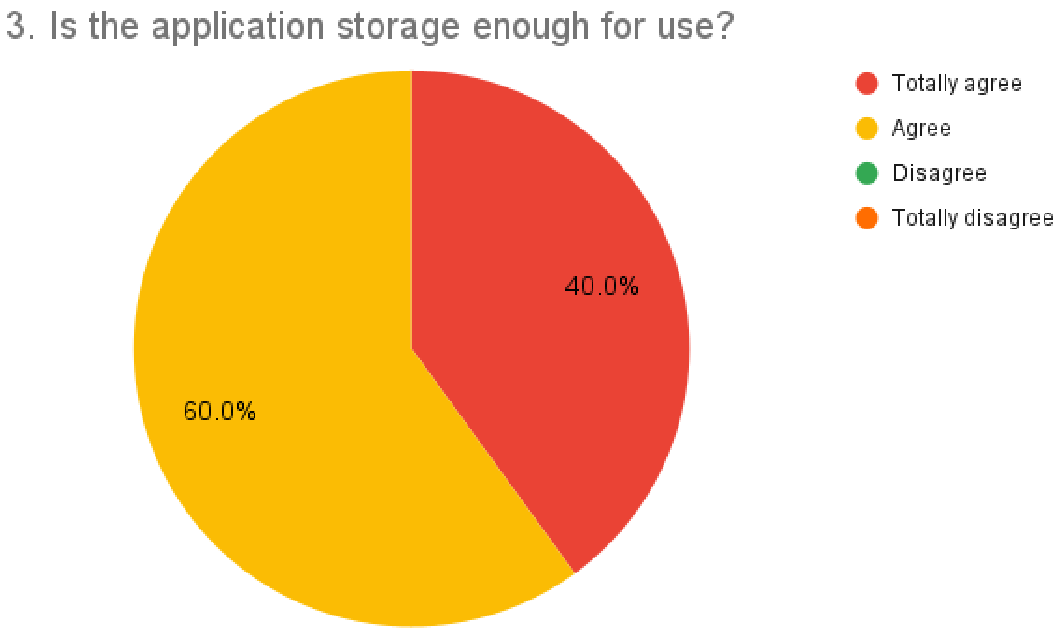
6.1.1 Usability Testing

Survey results:



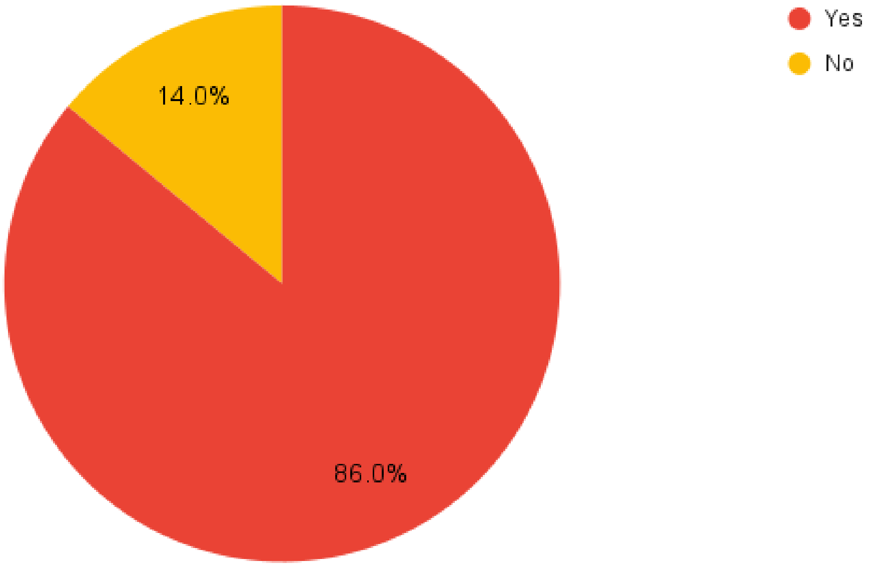


2. Is the data access rate fast enough?

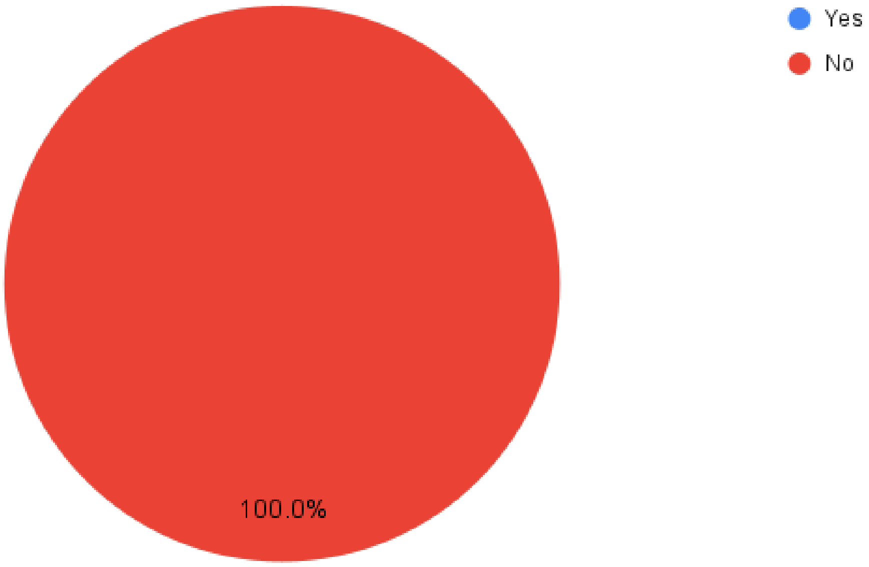


3. Is the application storage enough for use?

4. Are the functions sufficient?

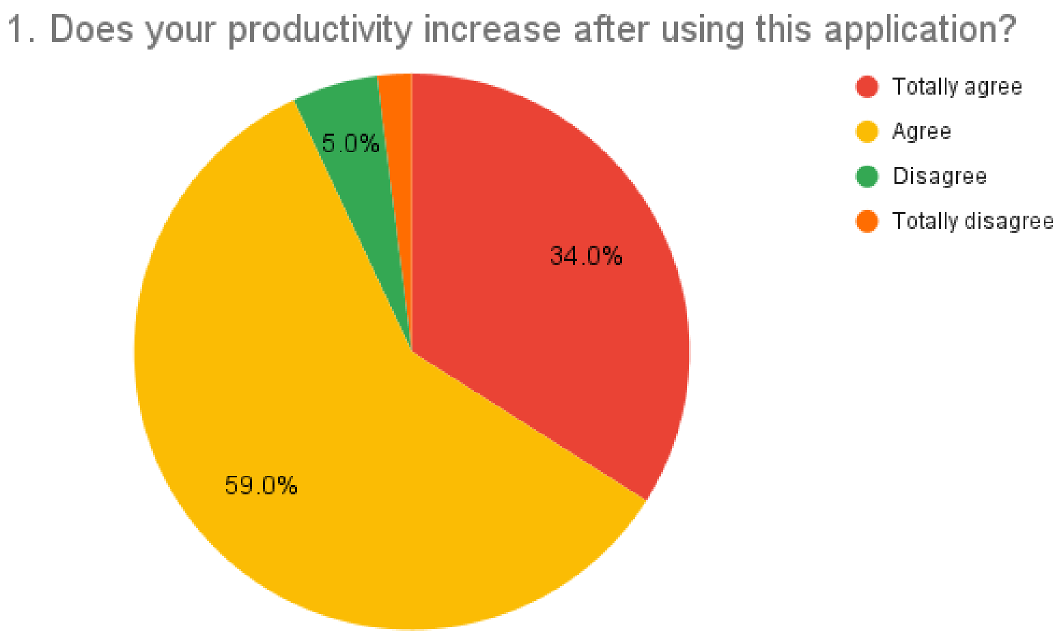


5. Is there any error on the functions?

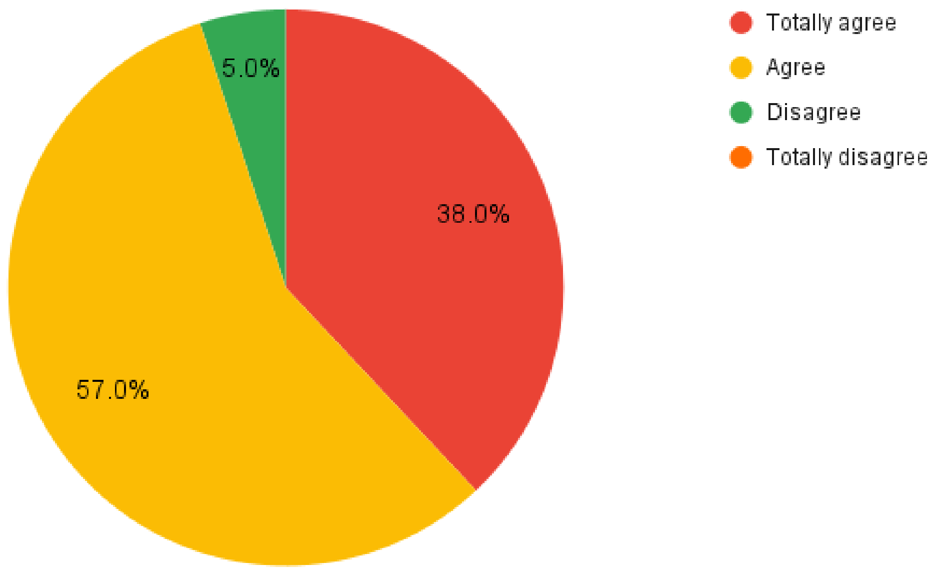


6.1.2 User Satisfaction

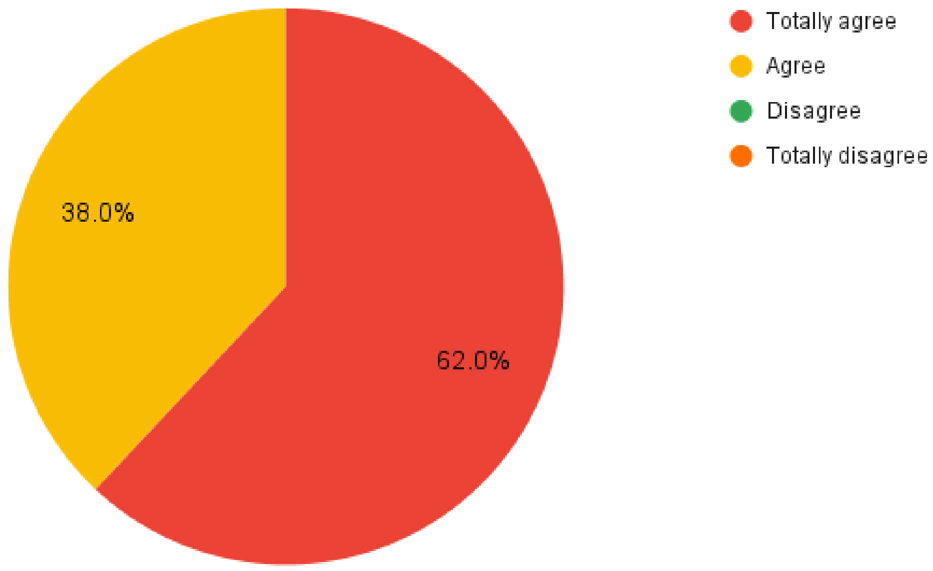
Survey results:

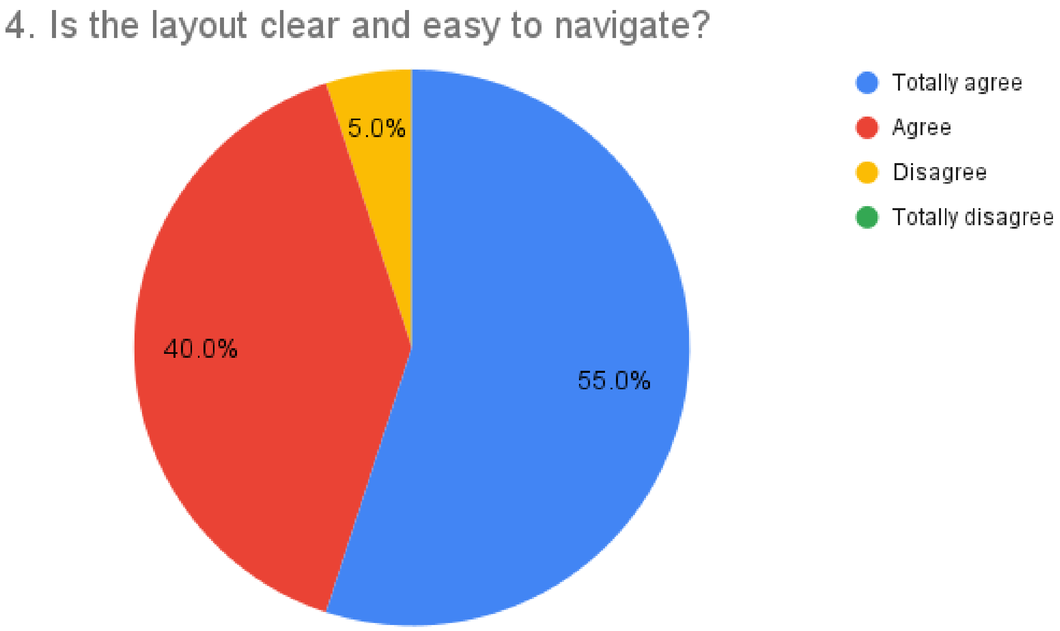


2. Is this application user-friendly?

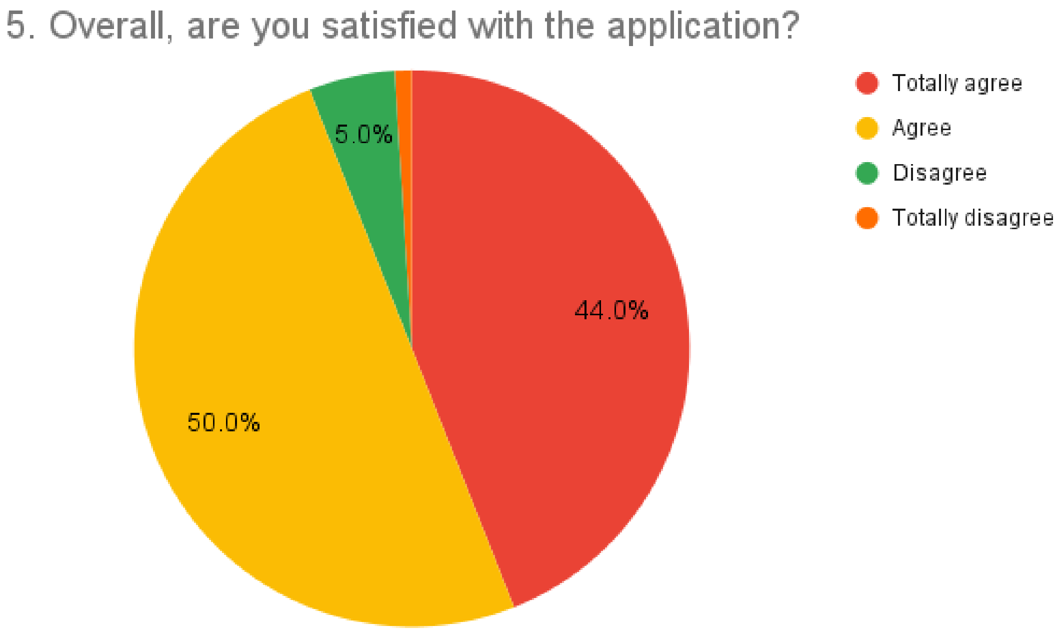


3. Is the task notification effective?





4. Is the layout clear and easy to navigate?



5. Overall, are you satisfied with the application?

6.2 Improvement after evaluation

1. Change the background color of notification to red
2. Change the indentation of ”Add Task” in **index.html** to center
3. Add guidelines e.g. ”filter by status (press the button):” and ”Tag management page (press the button):”
4. Change the color of tag button in **index.html** from grey to cyan
5. Reduce the padding-left value of ”Add Task” area in **index.html** and that of ”Update Task” area in **update.html**
6. ”Name” and ”Details” in **update.html** change to ”Current Name” and ”Current Details” in the display part

6.3 Suggestions from AI

Exact response from **Gemini**:

```

Here are some suggestions for your task manager's code:

**Frontend Improvements:**

* **Search functionality:**  Consider adding a search bar to filter tasks by name, tag, or due date. This would be helpful for users with many tasks.
* **Task prioritization:** Allow users to prioritize tasks (high, medium, low) to help them focus on the most important ones first. This could be implemented with dropdown menus or drag-and-drop functionality.
* **Progress tracking:**  For complex tasks,  a progress bar or checkbox system could be useful for tracking completion stages.
* **Visual enhancements:** Explore using different colors or icons to visually differentiate tasks based on status, priority, or tags. This can improve readability and information organization at a glance.

**Backend Improvements:**

* **Data validation:**  Implement data validation on the server-side to ensure users enter task details in the correct format (e.g., checking for valid date formats). This can prevent errors and improve data integrity.
* **User authentication:** If you plan to allow multiple users to access the task manager, consider implementing user authentication to restrict access and personalize task lists.
* **Database storage:** Consider storing tasks in a database (like SQLite or MySQL) instead of relying on temporary variables. This would allow data persistence and make the task manager more robust.


**General Improvements:**
```

```

* **Responsiveness:** Ensure the task manager layout adapts to different screen sizes for a good user experience on mobile devices.
* **Documentation:**  Provide clear documentation for users on how to use the task manager effectively, including features and functionalities.

These are just a few suggestions, and the best approach will depend on your specific needs and goals for the task manager.
```

6.4 Future Plan

6.4.1 Additional Features

1. Multi-user workspace and login
2. AI chat-bot page
3. Calendar
4. Missing tasks list
   - To show the missing tasks which are not completed by deadline
5. Dragging and dropping
   - Allow users to prioritize the tasks easily by simply changing the order of tasks in the table with dragging and dropping
6. Analysis function
   - To analysis the productivity based on how many tasks solved by user
7. Customization of reminder function
   - Users can set conditions for triggering reminders and specify specific times for reminders to be sent
8. Interactive elements
   - limited character or patterns of lines moving with cursor on the screen, random pictures of attractions retrieving from API could increase users' motivation to perform tasks

6.4.2 Performance Optimization

1. Indexing
   - Create indexes on frequently queried fields to speed up search and sorting operations
2. Data deletion

- In case the multi-user application is developed in the future, a function deleting the data of inactive users would be needed to save storage

6.4.3 Accessibility

1. Versions of different languages
2. Font-size adjustment feature
3. More visual elements or icons
    - Less words, more icons would make things clear, concise

7 Conclusion

This single-user application meets the project requirements and provides a user-friendly interface. It also includes additional, useful and effective features to enhance users productivity. Together with high access speed and accuracy, this application is highly esteemed.

8 References

8.1 Youtube

```
1. https://www.youtube.com/watch?v=09wwzzo30Vc
2. https://www.youtube.com/watch?v=8mfL-t0dn1M
3. https://www.youtube.com/watch?v=y8y_KIs9JLs
```

8.2 AI

1. https://gemini.google.com/

   - asking for suggestion on clarity, grammar proofread, JavaScript, HTML (forms, block extend) and CSS code generation

2. https://www.perplexity.ai/

   - asking for suggestion on project structure

8.3 Online resources

```
1. https://getbootstrap.com/docs/5.3/getting-started/introduction/
2. https://www.markdownguide.org/
3. https://colorhunt.co/
4. https://fontawesome.com/
5. https://www.securityjourney.com/post/how-to-prevent-sql-
    injection-vulnerabilities-how-prepared-statements-work
6. https://www.geeksforgeeks.org/python-sqlite-working-with-date-
    and-datetime/
7. https://stackoverflow.com/questions/68344204/how-to-make-a-
    button-a-double-checkcancel-confirm-so-no-coincidence-would-
    happ
8. https://devncoffee.com/drag-drop-sortable-table-rows-in-html-
    javascript/#google_vignette
9. https://github.com/clovon/Multipurpose-Laravel-and-Livewire-
    Application
10. https://github.com/dev-soumya-naskar/Drag-and-drop-table-rows-
    php-mysql-
11. https://www.w3schools.com/howto/howto_js_scroll_to_top.asp
12. https://medium.com/@ajay.monga73/sql-injection-prevention-for-c
    -developers-parameterized-queries-explained-b5a4cb1b6207
```