A. Introduction

1. Team Name: **Cyber Shield**
2. Team List:

Team Members	**Email**
Kristine Rivera	kdrivera@hawaii.edu
Josh Constantino	jmc921@hawaii.edu
Hangbo Zhang	hangbo@hawaii.edu
Joshua Rio	jmrio@hawaii.edu

3. Application Title: **OpenTab**
   Description & Functional Requirements Specification: Customer information such as names, table number, and credit card information are stored during their visit and are charged accordingly after they're done and all stored information is deleted from the system.
4. Program Type: Desktop Application
5. Development Tools: PyCharm
   Program Language: Python

B. Requirements

1. Security and Privacy Requirements:
   Security:
   - Access control: need-only base. Only access to whatever is needed to complete the job. One host is only allowed to access his/her tables' information
   - Authentication: each time to access will ask to prove their identity, login required after open the application
   - Encryption: data will be encrypted after inputted in, so only encrypted data will be transmitted or stored.

- Backup: all data will be back up to a cloud server
- Update: application will perform updates during the startup
- Incident reports: any unauthorized access attempts will be recorded and reported

Privacy:

- Data minimization: only necessary personal data will be collected in order to do the job
- Consent: consent will be collected before the data is inputted (agree to leave the tab open)
- Privacy notice: term of privacy notice/policy will be asked before using the application
- Data security: data will be encrypted and protected from unauthorized access
- Data retention: personal data will be erased after the tab is closed
- Incident reports: any unauthorized access attempts will be recorded and reported

2. Quality Gates (or Bug Bars):

Security:

Critical	○ Elevation of privilege: The ability to either execute arbitrary code or obtain more privilege than authorized. For Remote Anonymous User: ■ unauthorized access to user PII, ■ execution of arbitrary code, ■ SQL injection
Important	○ Denial of service ■ Anonymous ■ Temporary DoS with amplification ■ Authenticated DoS against a high value asset ○ Elevation of privilege: The ability to either execute arbitrary code or obtain more privilege than intended ■ Remote authenticated user ■ Local authenticated user ○ Information disclosure (targeted)

	■ Cases where the attacker can locate and read information from anywhere in the system, including system information that was not intended or designed to be exposed. ○ Spoofing ■ An entity (computer, server, user, process) is able to masquerade as a specific entity (user or computer) of his/her choice. ○ Tampering ■ Modification if any high value asset data in a common or default scenario where the modification persists after restarting the affected software ■ Permanent or persistent modification of any user or system data used in a common or default scenario ○ Security features: Breaking or bypassing any security feature provided.
Moderate	○ Denial of service ■ Anonymous temporary DoS without amplification in a default/common install ■ Authenticated Persistent DoS ■ Authenticated Temporary DoS with amplification in a default/common install ○ Information disclosure (targeted) ■ Cases where the attacker can easily read information on the system from specific locations, including system information, which was not intended/designed to be exposed. ○ Spoofing

	■ An entity (computer, server, user, process) is able to masquerade as a different, random entity that cannot be specifically selected. ○ Tampering ■ Permanent or persistent modification of any user or system data in a specific scenario ■ Temporary modification of data in a common or default scenario that does not persist after restarting the application. ○ Security assurances ■ Processes running with normal user privileges cannot gain admin privileges unless admin password/credentials have been provided via intentionally authorized methods.
Low	○ Information disclosure (untargeted) ■ Runtime information: random lea of memory ○ Tampering ■ Temporary modification of data in a specific scenario that does not persist after restarting the application

Privacy:

Critical	○ Lack of notice and consent: transferring of sensitive user PII from the system without prominent notice and explicit opt-in consent in the UI prior to transfer. ○ Lack of user controls: Ongoing collection and transfer of non-essential PII without the ability within the UI for the user to stop subsequent collection and transfer.

	○ Lack of data protection: PII is collected and stored in a persistent general database without an authentication mechanism for users to access and collect stored PII ○ Improper use of cookies: Sensitive PII stored is not encrypted ○ Lack of internal data management and control: Access to PII stored is not restricted only to those who have valid business need or there is no policy to revoke access after it is no longer required ○ Insufficient legal controls: Features transmits data to an agent or independent third party that has not signed a legally approved contract.
Important	○ Lack of notice and consent: Transfer of non-sensitive PII from system without prominent notice and explicit opt-in consent in the IO prior to the transfer. ○ Lack of user controls: Ongoing collection and transfer on non-essential anonymous data without the ability in the UI for the user to stop subsequent collection and transfer. ○ Lack of data protection: Persistently stored non-essential PII lacks mechanism to prevent unauthorized access. ○ Data minimization: Sensitive PII transmitted to an independent third party is not necessary to achieve the disclosed business purpose. ○ Improper use of cookies: Non-sensitive PII stored is not encrypted.
Moderate	○ Lack of user controls: PII is collected and stored locally as hidden metadata without any means for a user to remove the metadata. May be accessible by others or may be transmitted if files or folders are shared.

	○ Lack of data protection: Temporary stored non-sensitive PII lacks a mechanism to prevent unauthorized access during transfer or storage. ○ Data minimization: Non-sensitive PII or anonymous data transmitted to an independent third party is not necessary to achieve disclosed purpose. ○ Improper use of cookies: Use of persistent cookie where a session cookie would satisfy a purpose. ○ Lack of internal data management and control: Data stored does not have a retention policy
Low	○ Lack of notice and consent: PII is collected and stored locally as hidden metadata without discoverable notice. PII is not accessible by others and is not transmitted if files or folders are shared.

3. <u>Risk Assessment Plan for Security and Privacy:</u>

   Initial assessment:
   - Determine the Privacy Impact Rating
     - Software behaviors
       - Stores or transfer PII
       - Targets or attractive to children
       - Continuously monitors the user
       - Installs new software or changes files
       - Transfers anonymous data
   - Detailed Privacy Analysis
     - Describe the PII stored or transferred
     - Describe user value proposition and business justification
     - Describe software installation or files changing
     - Describe notice and consent experiences
     - Describe how users access public disclosure

    - Describe how organizations can control features
    - Describe how users can control features
    - Describe how to prevent unauthorized access to PII
- Identifying potential vulnerabilities and threats
    - List possible way to access PII
    - List possible way to control features
    - List possible way to software crush
    - List possible way to DoS
- Determine the likelihood and impact of risks
    - List impacts of leaking PII
    - List impacts of leaking user authentication information
    - List impacts of software crush
    - List impacts of DoS

Penetration testing:

- Attempting to exploit vulnerabilities listed above

Code review:

- Analyzing the source code
- Introduce a review team

Compliance testing:

- Verifying the program adheres to relevant security and privacy regulations and standards

User testing:

- Getting feedback from users
- Using bug bounty platform

Regular security audit:

- Periodically checking
- Update immediately when newer version is published

C. Design

1. Design Requirements:

- Ask user for consent to temporary store user PII
- PII is stored locally and subjected to deletion once payment is done
- Sensitive PII is stored using encryption
- End of day reports shared to other parties do not contain users' full credit card number, name, address, and food item ordered.

2. Attack Surface Analysis and Reduction:
    - Open ports
    - Enabled accounts
    - Enabled accounts in admin group
    - Null sessions to pipes and shares
    - Weak ACLs in the File System
    - Weak ACLs in Registry
    - Weak ACLs on shares

    Similar program: Credit card readers

    List of vulnerabilities:
    - Can exploit bugs in bluetooth and mobile app connectivity to the device to intercept transactions or modify commands
    - Flaws allows attacker to disable chip-based transactions, forcing customers to use less secure magstripe swipe, and making it easier to steal data and clone customer cards
    - Can make transactions to appear to be declines in order to repeat it multiple times
    - able to modify transaction amount
    - Improperly stored Credit Card information could be targeted

3. Threat Modeling:

- Identify the assets: what information, resources, and functionality need to be protected
- Identify the actors: identify the potential attackers both inside and outside, and their motives

- Identify threats: analyze the potential threats to the assets, taking into account the actors and their motives
- Understand the threats
- Categorize the threats
- Identify the vulnerabilities: identify the weaknesses in the system that could be exploited by the threats
- Mitigating the threats: determine the most effective ways to mitigate the threats
- Monitoring and Testing

Threat Modeling Report

Diagram: Diagram 1

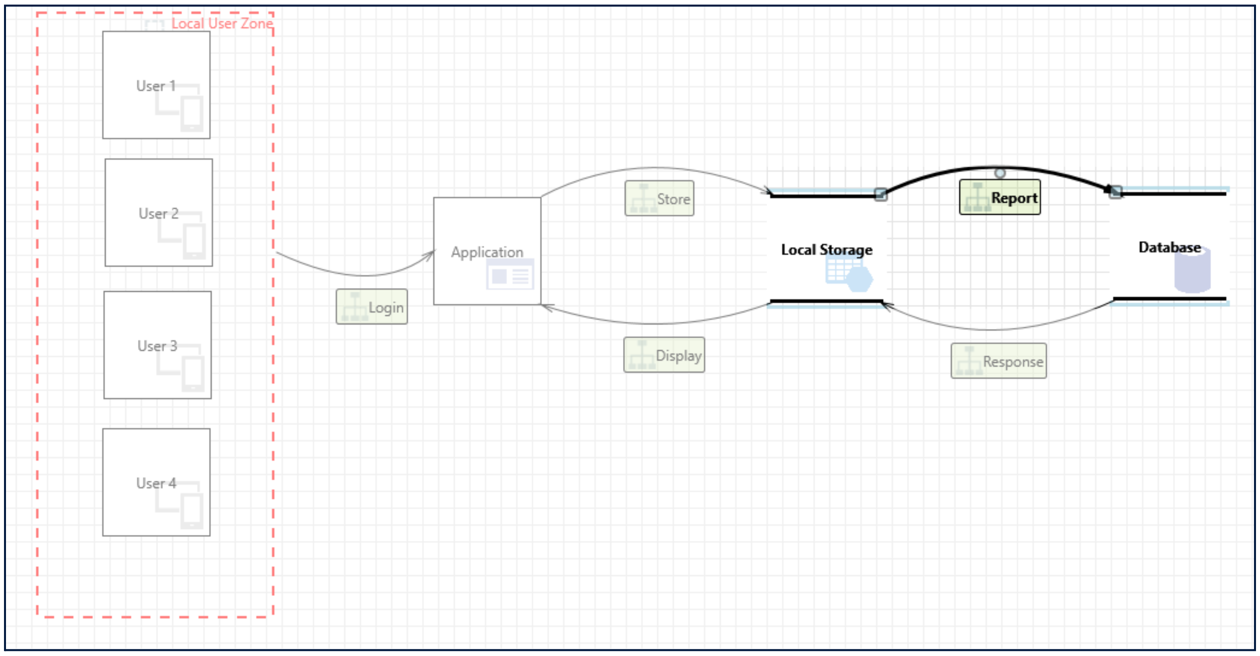
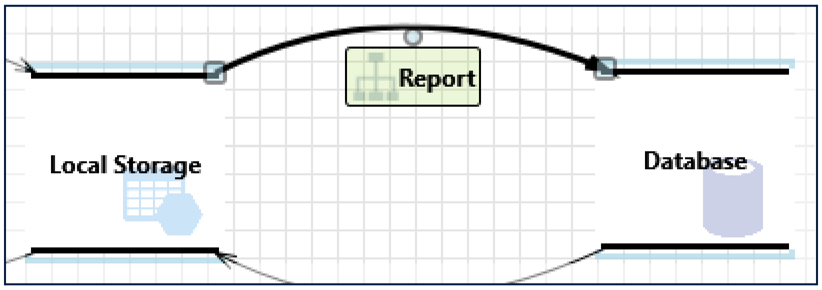


Diagram 1 Diagram Summary:

Not Started	21
Not Applicable	0
Needs Investigation	0
Mitigation Implemented	0

Total	21
Total Migrated	0

Interaction: Report



1. An adversary can gain unauthorized access to database due to lack of network access protection [State: Not Started] [Priority: High]

Category:	Elevation of Privileges
Description:	If there is no restriction at network or host firewall level, to access the database then anyone can attempt to connect to the database from an unauthorized location
Justification:	<no mitigation provided>
Possible Mitigation(s):	Configure a Windows Firewall for Database Engine Access.
SDL Phase:	Implementation

2. An adversary can gain unauthorized access to database due to loose authorization rules [State: Not Started] [Priority: High]

Category:	Elevation of Privileges
Description:	Database access should be configured with roles and privileges based on least privilege and need to know principle.
Justification:	<no mitigation provided>

Possible Mitigation(s):	Ensure that least-privileged accounts are used to connect to Database servers. Implement Row Level Security RLS to prevent tenants from accessing each other's data. Sysadmin role should only have valid necessary users .
SDL Phase:	Implementation

3. An adversary can gain access to sensitive PII or HBI data in database [State: Not Started] [Priority: High]

Category:	Information Disclosure
Description:	Additional controls like Transparent Data Encryption, Column Level Encryption, EKM etc. provide additional protection mechanisms to high value PII or HBI data.
Justification:	<no mitigation provided>
Possible Mitigation(s):	Use strong encryption algorithms to encrypt data in the database. Ensure that sensitive data in database columns is encrypted. Ensure that database-level encryption (TDE) is enabled. Ensure that database backups are encrypted. Use SQL server EKM to protect encryption keys. Use AlwaysEncrypted feature if encryption keys should not be revealed to the Database engine.
SDL Phase:	Implementation

4. An adversary can gain access to sensitive data by performing SQL injection [State: Not Started] [Priority: High]

Category:	Information Disclosure
Description:	SQL injection is an attack in which malicious code is inserted into strings that are later passed to an instance of SQL Server for parsing and execution. The primary form of SQL injection consists of direct insertion of code into user-input variables that are concatenated with SQL commands and executed. A less direct attack injects malicious code into strings that are destined for storage in a table or as metadata. When the stored strings are subsequently concatenated into a dynamic SQL command, the malicious code is executed.

Justification:	<no mitigation provided>
Possible Mitigation(s):	Ensure that logon auditing is enabled on SQL Server. Ensure that least-privileged accounts are used to connect to Database servers. Enable Threat detection on Azure SQL database. Do not use dynamic queries in stored procedures.
SDL Phase:	Implementation

5. An adversary can deny actions on database due to lack of auditing [State: Not Started] [Priority: Medium]

Category:	Repudiation
Description:	Proper logging of all security events and user actions builds traceability in a system and denies any possible repudiation issues. In the absence of proper auditing and logging controls, it would become impossible to implement any accountability in a system.
Justification:	<no mitigation provided>
Possible Mitigation(s):	Ensure that login auditing is enabled on SQL Server.
SDL Phase:	Implementation

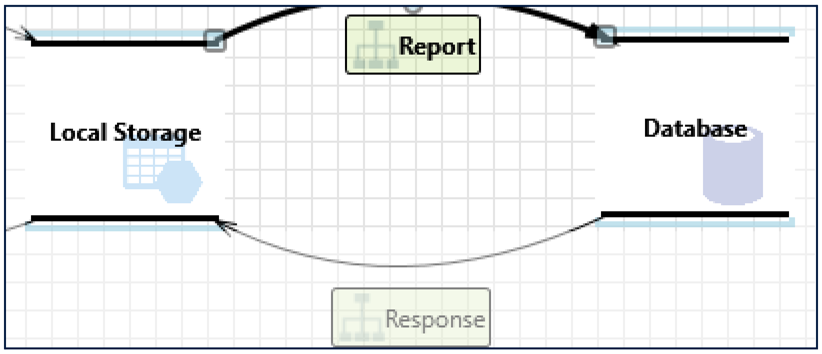
6. An adversary can tamper critical database securables and deny the action [State: Not Started] [Priority: High]

Category:	Tampering
Description:	An adversary can tamper critical database securables and deny the action
Justification:	<no mitigation provided>
Possible Mitigation(s):	Add digital signature to critical database securables.
SDL Phase:	Design

7. An adversary may leverage the lack of monitoring systems and trigger anomalous traffic to database [State: Not Started] [Priority: High]

Category:	Tampering
Description:	An adversary may leverage the lack of intrusion detection and prevention of anomalous database activities and trigger anomalous traffic to database
Justification:	<no mitigation provided>
Possible Mitigation(s):	Enable Threat detection on Azure SQL database.
SDL Phase:	Design

Interaction: Response



8. An adversary can gain unauthorized access to Local Storage due to weak access control restrictions [State: Not Started] [Priority: High]

Category:	Elevation of Privileges
Description:	An adversary can gain unauthorized access to Local Storage due to weak access control restrictions
Justification:	<no mitigation provided>
Possible Mitigation(s):	Grant limited access to objects in Azure Storage using SAS or SAP. It is recommended to scope SAS and SAP to permit only the necessary permissions over a short period of time.
SDL Phase:	Implementation

9. An adversary can gain unauthorized access to Local Storage instances due to weak network configuration [State: Not Started] [Priority: High]

Category:	Elevation of Privileges
Description:	An adversary can gain unauthorized access to Local Storage instances due to weak network configuration
Justification:	<no mitigation provided>
Possible Mitigation(s):	It is recommended to restrict access to Azure Storage instances to selected networks where possible.
SDL Phase:	Implementation

10. An adversary may gain unauthorized access to Local Storage account in a subscription [State: Not Started] [Priority: High]

Category:	Elevation of Privileges
Description:	An adversary may gain unauthorized access to Local Storage account in a subscription
Justification:	<no mitigation provided>
Possible Mitigation(s):	Assign the appropriate Role-Based Access Control (RBAC) role to users, groups and applications at the right scope for the Azure Storage instance.
SDL Phase:	Implementation

11. An adversary can abuse poorly managed Local Storage account access keys [State: Not Started] [Priority: High]

Category:	Elevation of Privileges
Description:	An adversary can abuse poorly managed Local Storage account access keys and gain unauthorized access to storage.
Justification:	<no mitigation provided>
Possible Mitigation(s):	Ensure secure management and storage of Azure storage access keys. It is recommended to rotate storage access keys regularly, in accordance with organizational policies.
SDL Phase:	Implementation

12. An adversary can abuse an insecure communication channel between a client and Local Storage [State: Not Started] [Priority: Medium]

Category:	Information Disclosure
Description:	An adversary can abuse an insecure communication channel between a client and Local Storage
Justification:	<no mitigation provided>
Possible Mitigation(s):	Ensure that communication to Azure Storage is over HTTPS. It is recommended to enable the secure transfer required option to force communication with Azure Storage to be over HTTPS. Use Client-Side Encryption to store sensitive data in Azure Storage.
SDL Phase:	Implementation

13. An adversary can deny actions on Local Storage due to lack of auditing [State: Not Started] [Priority: Medium]

Category:	Repudiation
Description:	Proper logging of all security events and user actions builds traceability in a system and denies any possible repudiation issues. In the absence of proper auditing and logging controls, it would become impossible to implement any accountability in a system.
Justification:	<no mitigation provided>
Possible Mitigation(s):	Use Azure Storage Analytics to audit access of Azure Storage. If possible, audit the calls to the Azure Storage instance at the source of the call.
SDL Phase:	Implementation

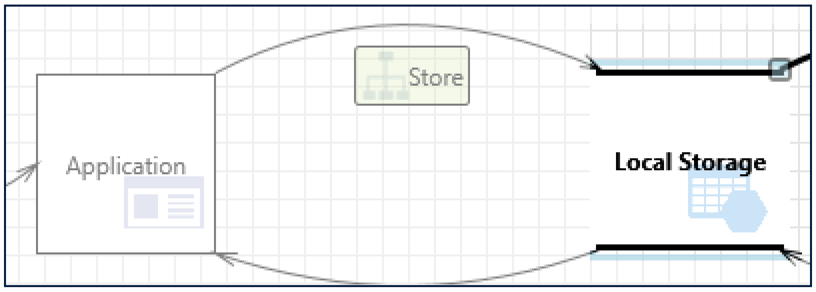
14. An adversary can gain unauthorized access to Local Storage due to weak CORS configuration [State: Not Started] [Priority: High]

Category:	Elevation of Privileges
Description:	An adversary can gain unauthorized access to Local Storage due to weak CORS configuration
Justification:	<no mitigation provided>

Possible Mitigation(s): Ensure that only specific, trusted origins are allowed.

SDL Phase: Implementation

Interaction: Store



15. An adversary can abuse an insecure communication channel between a client and Local Storage [State: Not Started] [Priority: Medium]

Category: Information Disclosure

Description: An adversary can abuse an insecure communication channel between a client and Local Storage

Justification: <no mitigation provided>

Possible Mitigation(s): Ensure that communication to Azure Storage is over HTTPS. It is recommended to enable the secure transfer required option to force communication with Azure Storage to be over HTTPS. Use Client-Side Encryption to store sensitive data in Azure Storage.

SDL Phase: Implementation

16. An adversary can abuse poorly managed Local Storage account access keys [State: Not Started] [Priority: High]

Category: Elevation of Privileges

Description: An adversary can abuse poorly managed Local Storage account access keys and gain unauthorized access to storage.

Justification: <no mitigation provided>

Possible Mitigation(s):	Ensure secure management and storage of Azure storage access keys. It is recommended to rotate storage access keys regularly, in accordance with organizational policies.
SDL Phase:	Implementation

17. An adversary may gain unauthorized access to Local Storage account in a subscription [State: Not Started] [Priority: High]

Category:	Elevation of Privileges
Description:	An adversary may gain unauthorized access to Local Storage account in a subscription
Justification:	<no mitigation provided>
Possible Mitigation(s):	Assign the appropriate Role-Based Access Control (RBAC) role to users, groups and applications at the right scope for the Azure Storage instance.
SDL Phase:	Implementation

18. An adversary can gain unauthorized access to Local Storage instances due to weak network configuration [State: Not Started] [Priority: High]

Category:	Elevation of Privileges
Description:	An adversary can gain unauthorized access to Local Storage instances due to weak network configuration
Justification:	<no mitigation provided>
Possible Mitigation(s):	It is recommended to restrict access to Azure Storage instances to selected networks where possible.
SDL Phase:	Implementation

19. An adversary can gain unauthorized access to Local Storage due to weak access control restrictions [State: Not Started] [Priority: High]

Category:	Elevation of Privileges
Description:	An adversary can gain unauthorized access to Local Storage due to weak access control restrictions

Justification:	<no mitigation provided>
Possible Mitigation(s):	Grant limited access to objects in Azure Storage using SAS or SAP. It is recommended to scope SAS and SAP to permit only the necessary permissions over a short period of time.
SDL Phase:	Implementation

20. An adversary can deny actions on Local Storage due to lack of auditing [State: Not Started] [Priority: Medium]

Category:	Repudiation
Description:	Proper logging of all security events and user actions builds traceability in a system and denies any possible repudiation issues. In the absence of proper auditing and logging controls, it would become impossible to implement any accountability in a system.
Justification:	<no mitigation provided>
Possible Mitigation(s):	Use Azure Storage Analytics to audit access of Azure Storage. If possible, audit the calls to the Azure Storage instance at the source of the call.
SDL Phase:	Implementation

21. An adversary can gain unauthorized access to Local Storage due to weak CORS configuration [State: Not Started] [Priority: High]

Category:	Elevation of Privileges
Description:	An adversary can gain unauthorized access to Local Storage due to weak CORS configuration
Justification:	<no mitigation provided>
Possible Mitigation(s):	Ensure that only specific, trusted origins are allowed.
SDL Phase:	Implementation

Repository: https://github.com/cybershield427/OpenTab

Implementation

1. **Approved tools:**

Type of tool	Name	Minimum Version	Comment
Compiler	Python3	3.10.10	In the context of Python, there is no compiler in the traditional sense. Instead, Python is an interpreted language, meaning that the source code is executed directly without being compiled into machine code beforehand. When you run a Python script, the Python interpreter reads the source code, parses it, and executes it line by line. The interpreter will report any errors it encounters during this process. This is different from a compiled language like C or C++, where the source code is compiled into machine code before it is executed.
IDE	PyCharm	2022.1.1 (Professional Edition) Build #PY-221.5591.52 Runtime version: 11.0.14.1	PyCharm is an integrated development environment (IDE) used in computer programming, specifically for the Python language. It is developed by JetBrains, and provides a range of features such as syntax highlighting, code completion, debugging tools, version control integration, and more. PyCharm is designed to make it easier and more efficient for developers to write and maintain Python code. It is available in both free and paid versions.
Source code Analyze	PyCharm Analyze code Option	Built-in with PyCharm 2022.1.1	When you run code analysis in PyCharm, it generates a report that highlights any issues found in your code. You can then use this information to identify and fix problems in your code, and improve its overall quality. Some of the features of PyCharm's code analysis include: 1. Syntax highlighting and error

			highlighting 2. Code inspections and quick fixes 3. Code style analysis and formatting suggestions 4. Unused code detection 5. Code completion and suggestion 6. Code coverage analysis 7. Refactoring suggestions
Automatic Testing tool	Pytest	7.2.1	It allows you to write and run tests for your Python code in a simple and efficient manner. Some of the features provided by Pytest include: 1. Support for testing with Python's built-in unittest module. 2. Support for testing with third-party testing frameworks such as nose. 3. Ability to run tests in parallel for faster test execution. 4. Detailed test reports with configurable output formats. 5. Support for testing asynchronous code. 6. Powerful assertion introspection and output capture.
UI toolkit	PySide6	6.4.2	PySide6 is a set of Python bindings for the Qt application framework and runs on all platforms supported by Qt including Windows, OS X, Linux, iOS, and Android. It allows Python programmers to create GUI applications with the Qt framework.
Code style	PyCharm Inspectio	Buit-in with PyCharm 2022.1.1	It performs a wide range of checks on your code, including syntax and type checking, PEP8 compliance, code duplication, and many others. You can configure the inspections to run automatically as you code or manually run them at any time.

2. **Deprecated/Unsafe Functions :**
    - exec() is a function that can execute arbitrary code, which can be a major security risk if the input is not properly sanitized. In the case of our project,

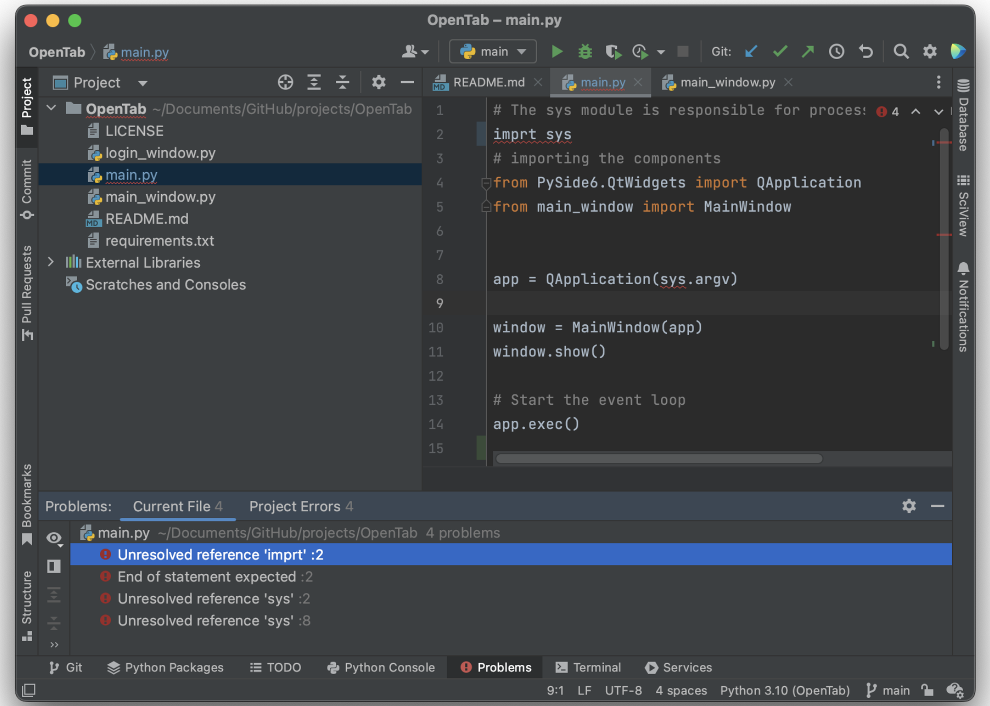
we first use it to start the application and will not take input. Therefore, it will not be vulnerable to malicious input. However, later on while we implement the connection with the database then the function will be called to execute the query command. There is no alternative function. 1. But we can validate and sanitize the input. Instead of directly putting the input in the function, we will store the input in a string variable and limit the size. And then we will perform the validation and sanitation by checking if any special command is included. 2. Also, we will use prepared statements instead. 3. We will use the principle of least privilege to limit what a user can do to protect the database.

- file.read() can be used to read the database or csv file into the program. However, this function can read the entire file into memory at once, which can be a performance issue for large files. The alternative way would be to use file.readline() iteratively to read the file line by line instead.
- Httplib was the previous library used to make connections with remote servers (databases). However, it is not secure as it does not perform any validation or sanitization of user input, which could lead to vulnerabilities such as HTTP response splitting attacks. The alternative function will be http.client(). By using secure protocols such as HTTPS and strong encryption algorithms (RSA), it can make a secure connection with the server.
- There are some deprecated functions that need to pay attention on such as use int() instead of string.atoi() (from C/C++), use open() instead of file(), use print() instead of print, use input() instead of raw_input(), use comparison operators instead of cmp().

3. **<u>Static Analysis:</u>**
    - Pycharm Inspection is good at detecting and correcting abnormal python codes. The IDE should find and highlight these various abnormal codes, locate dead codes, find potential bugs, spelling errors and help improve coding structure. Inspections also display the severity of these problems by highlighting them in different colors, red being the most critical. It

provides suggestions for improving your code's readability, maintainability, and performance. Some examples of the types of issues it can detect include unused code, unused imports, potential bugs, code smells, and style violations.

- Experiences: Inspection successfully locates problematic code, where import is misspelled.



```
# The sys module is responsible for proces:
imprt sys
# importing the components
from PySide6.QtWidgets import QApplication
from main_window import MainWindow


app = QApplication(sys.argv)

window = MainWindow(app)
window.show()

# Start the event loop
app.exec()
```

```
Problems:   Current File 4   Project Errors 4
main.py ~/Documents/GitHub/projects/OpenTab  4 problems
    Unresolved reference 'imprt' :2
    End of statement expected :2
    Unresolved reference 'sys' :2
    Unresolved reference 'sys' :8
```

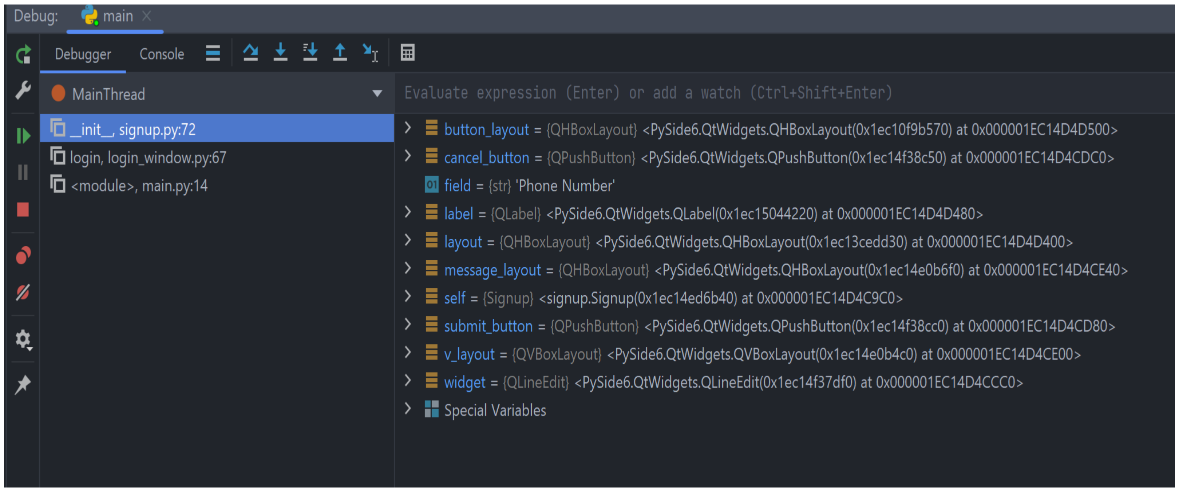
Verification

Dynamic Analysis

The dynamic analysis tool we have decided to use is the built-in tool set in the PyCharm as it can check multiple areas and provide feedback instantly. Also since they are built-in to the IDE, it’s very easy to use and navigate. The following are the 5 key tools we will be using for dynamic analysis for different purposes.

1. Debugger: PyCharm's debugger allows us to step through our Python code and inspect variables and data structures while the code is running. We can set

breakpoints in our code to stop the execution at a specific line, and we can also run the code step-by-step to see how it executes. The debugger supports remote debugging, which means we can debug Python code running on a remote machine, and it also supports multi-threaded debugging, which allows us to debug code that uses threads. Here is an example of a debugger.

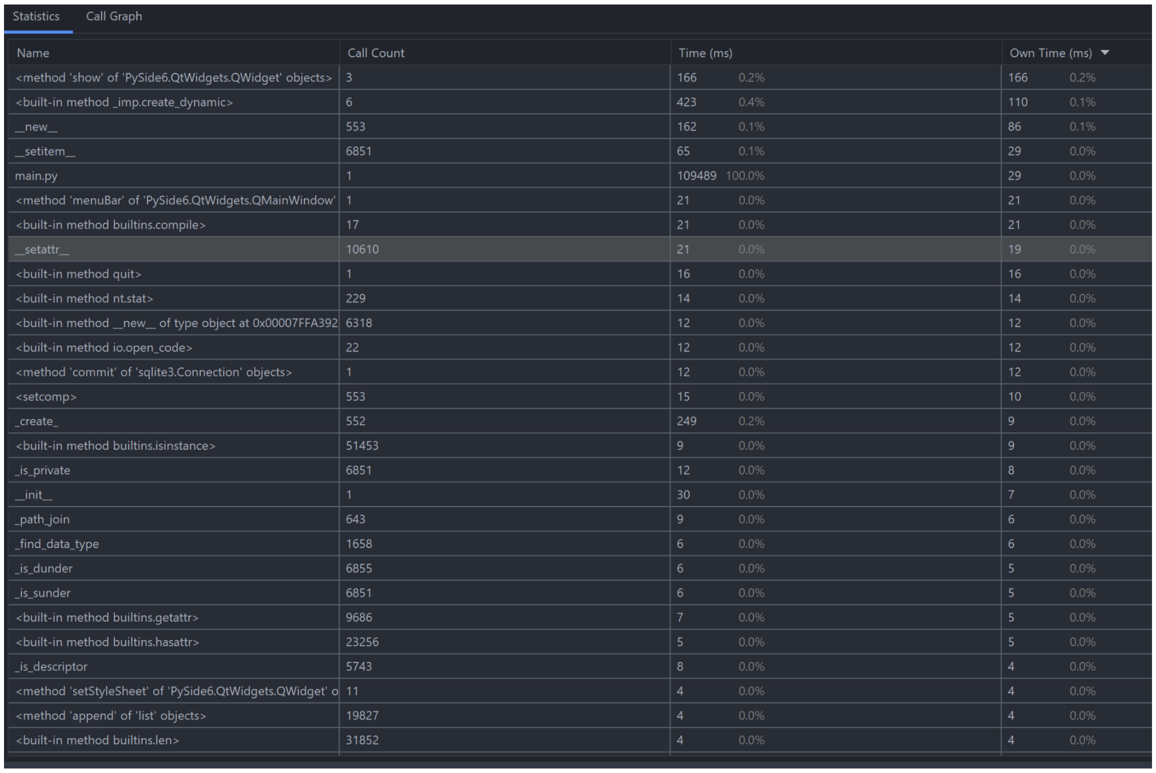


```
Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)

__init__, signup.py:72
login, login_window.py:67
<module>, main.py:14

button_layout = {QHBoxLayout} <PySide6.QtWidgets.QHBoxLayout(0x1ec10f9b570) at 0x000001EC14D4D500>
cancel_button = {QPushButton} <PySide6.QtWidgets.QPushButton(0x1ec14f38c50) at 0x000001EC14D4CDC0>
field = {str} 'Phone Number'
label = {QLabel} <PySide6.QtWidgets.QLabel(0x1ec15044220) at 0x000001EC14D4D480>
layout = {QHBoxLayout} <PySide6.QtWidgets.QHBoxLayout(0x1ec13cedd30) at 0x000001EC14D4D400>
message_layout = {QHBoxLayout} <PySide6.QtWidgets.QHBoxLayout(0x1ec14e0b6f0) at 0x000001EC14D4CE40>
self = {Signup} <signup.Signup(0x1ec14ed6b40) at 0x000001EC14D4C9C0>
submit_button = {QPushButton} <PySide6.QtWidgets.QPushButton(0x1ec14f38cc0) at 0x000001EC14D4CD80>
v_layout = {QVBoxLayout} <PySide6.QtWidgets.QVBoxLayout(0x1ec14e0b4c0) at 0x000001EC14D4CE00>
widget = {QLineEdit} <PySide6.QtWidgets.QLineEdit(0x1ec14f37df0) at 0x000001EC14D4CCC0>
Special Variables
```

Experience:

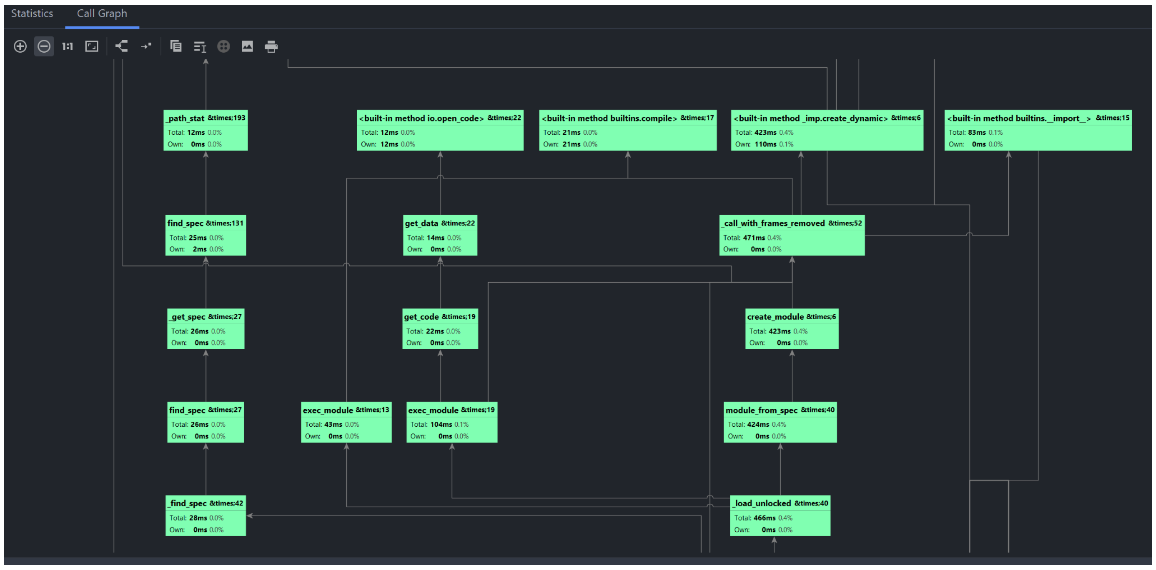
The debugger is great to have when we are trying to test the detailed behavior of a function and it is easy to use. However, in terms of our project which will create a UI window and wait for the user input, it is not ideal. It will not test the functionality of the application as a whole. Also, the breakpoint will initiate the crash of the application as the input process has been stopped which will lead to the “Not Responding” issue. Therefore, it is not very useful unless we are trying to find some specific issue in the functions.

2. Profiler: PyCharm's profiler can help us identify performance bottlenecks in our Python code. The profiler measures the time spent in each function and generates a report that shows us the number of calls, the total time spent, and other performance metrics. We can use this information to identify slow-running functions and optimize our code for better performance. Here is an example of the profiler.



Statistics | Call Graph

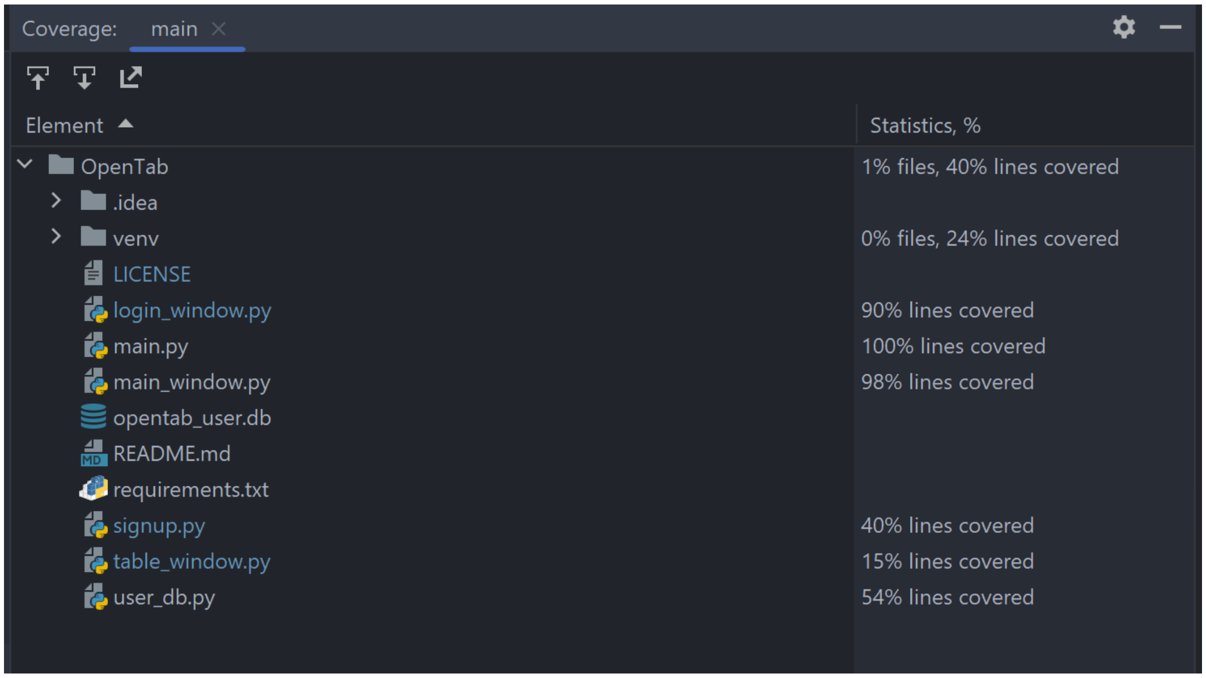
Name	Call Count	Time (ms)	Own Time (ms)
<method 'show' of 'PySide6.QtWidgets.QWidget' objects>	3	166 0.2%	166 0.2%
<built-in method _imp.create_dynamic>	6	423 0.4%	110 0.1%
__new__	553	162 0.1%	86 0.1%
__setitem__	6851	65 0.1%	29 0.0%
main.py	1	109489 100.0%	29 0.0%
<method 'menuBar' of 'PySide6.QtWidgets.QMainWindow'	1	21 0.0%	21 0.0%
<built-in method builtins.compile>	17	21 0.0%	21 0.0%
__setattr__	10610	21 0.0%	19 0.0%
<built-in method quit>	1	16 0.0%	16 0.0%
<built-in method nt.stat>	229	14 0.0%	14 0.0%
<built-in method __new__ of type object at 0x00007FFA392	6318	12 0.0%	12 0.0%
<built-in method io.open_code>	22	12 0.0%	12 0.0%
<method 'commit' of 'sqlite3.Connection' objects>	1	12 0.0%	12 0.0%
<setcomp>	553	15 0.0%	10 0.0%
create	552	249 0.2%	9 0.0%
<built-in method builtins.isinstance>	51453	9 0.0%	9 0.0%
_is_private	6851	12 0.0%	8 0.0%
__init__	1	30 0.0%	7 0.0%
_path_join	643	9 0.0%	6 0.0%
_find_data_type	1658	6 0.0%	6 0.0%
_is_dunder	6855	6 0.0%	5 0.0%
_is_sunder	6851	6 0.0%	5 0.0%
<built-in method builtins.getattr>	9686	7 0.0%	5 0.0%
<built-in method builtins.hasattr>	23256	5 0.0%	5 0.0%
_is_descriptor	5743	8 0.0%	4 0.0%
<method 'setStyleSheet' of 'PySide6.QtWidgets.QWidget' o	11	4 0.0%	4 0.0%
<method 'append' of 'list' objects>	19827	4 0.0%	4 0.0%
<built-in method builtins.len>	31852	4 0.0%	4 0.0%


Statistics
Call Graph
1:1
_path_stat &times;193 Total: 12ms 0.0% Own: 0ms 0.0%
<built-in method io.open_code> &times;22 Total: 12ms 0.0% Own: 12ms 0.0%
<built-in method builtins.compile> &times;17 Total: 21ms 0.0% Own: 21ms 0.0%
<built-in method _imp.create_dynamic> &times;6 Total: 423ms 0.4% Own: 110ms 0.1%
<built-in method builtins.__import__> &times;15 Total: 83ms 0.1% Own: 0ms 0.0%
find_spec &times;131 Total: 25ms 0.0% Own: 2ms 0.0%
get_data &times;22 Total: 14ms 0.0% Own: 0ms 0.0%
_call_with_frames_removed &times;52 Total: 471ms 0.4% Own: 0ms 0.0%
_get_spec &times;27 Total: 26ms 0.0% Own: 0ms 0.0%
get_code &times;19 Total: 22ms 0.0% Own: 0ms 0.0%
create_module &times;6 Total: 423ms 0.4% Own: 0ms 0.0%
find_spec &times;27 Total: 26ms 0.0% Own: 0ms 0.0%
exec_module &times;13 Total: 43ms 0.0% Own: 0ms 0.0%
exec_module &times;19 Total: 104ms 0.1% Own: 0ms 0.0%
module_from_spec &times;40 Total: 424ms 0.4% Own: 0ms 0.0%
_find_spec &times;42 Total: 28ms 0.0% Own: 0ms 0.0%
_load_unlocked &times;40 Total: 466ms 0.4% Own: 0ms 0.0%

Experience:

Profiling is very useful in terms of dynamic analysis as we will be able to find the flow of the function calls to find out if the structure of the application is implemented correctly as we designed. Also, the metrics are very helpful for us to understand the performance of each function. And it is very easy to use as well.

3. Coverage analysis: PyCharm's code coverage analysis tool can help us measure the coverage of our code during test execution. The tool generates a report that shows us which lines of code were executed during the test, and which lines were not executed. We can use this information to ensure that our tests cover all the important parts of our code. Here is an example of coverage analysis.

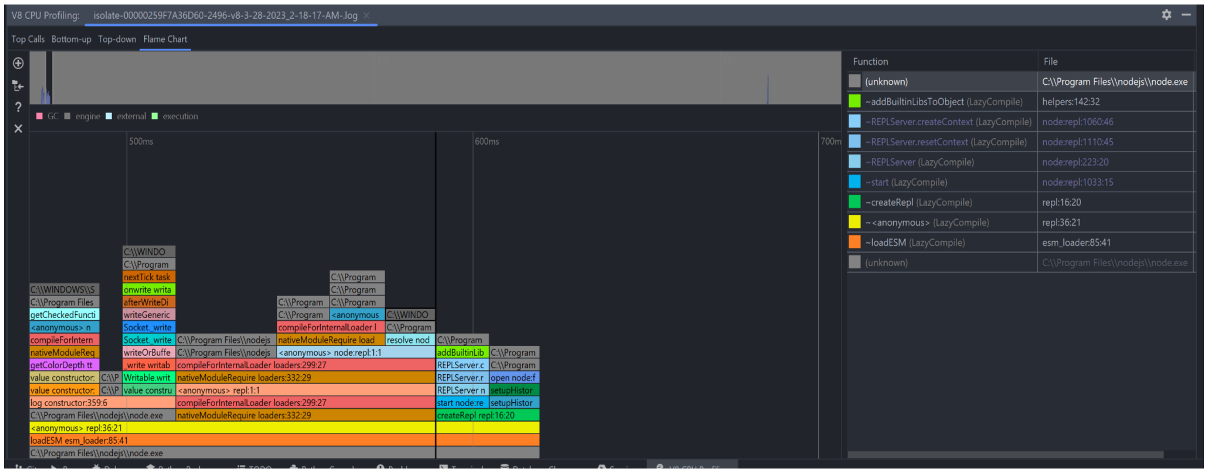


Element	Statistics, %
OpenTab	1% files, 40% lines covered
.idea	
venv	0% files, 24% lines covered
LICENSE	
login_window.py	90% lines covered
main.py	100% lines covered
main_window.py	98% lines covered
opentab_user.db	
README.md	
requirements.txt	
signup.py	40% lines covered
table_window.py	15% lines covered
user_db.py	54% lines covered

Experience:

Code coverage analysis is another very useful tool. In terms of our project, it is more of the testing verification process. In order to get 100% coverage of the code, we need to test all the possible cases as we designed. Also, this is a good way to help us find some unreachable code or dead loops. As it was built-in with PyCharm, it is very easy to use.

4. Memory profiler: PyCharm's memory profiler allows us to identify memory leaks and other memory-related issues in our Python code. The memory profiler measures the memory usage of our program and generates a report that shows us the memory allocation and deallocation over time. We can use this information to identify parts of our code that are using too much memory or not releasing memory properly. Here is an example of the memory profiler.

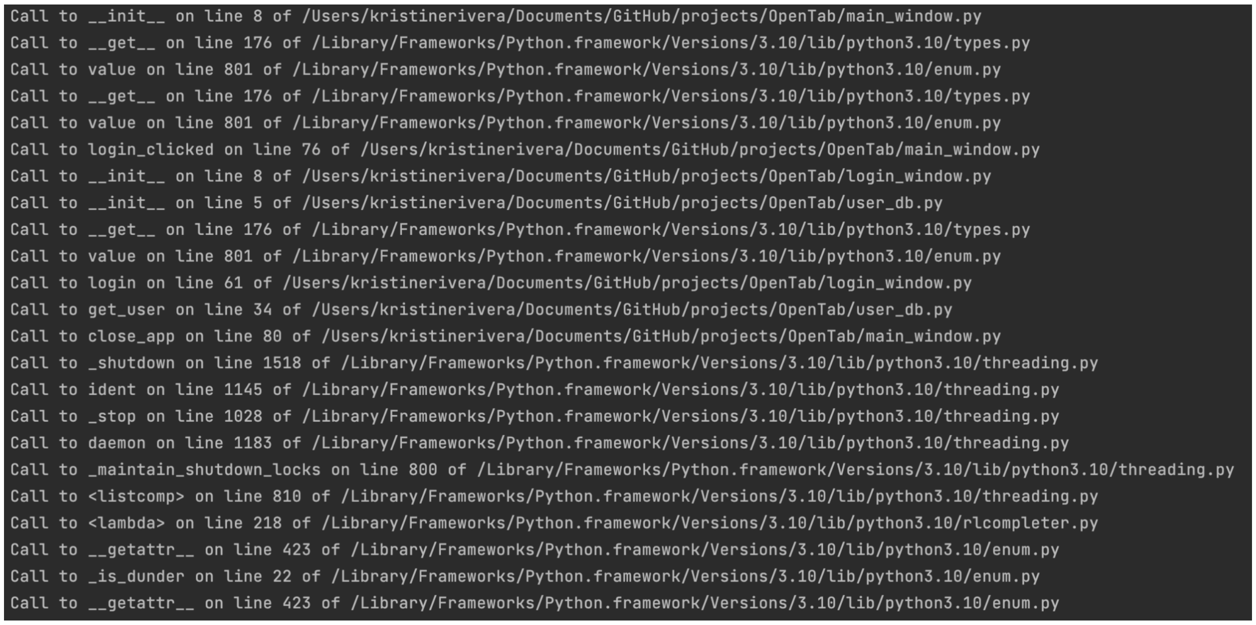

V8 CPU Profiling: isolate-00000259F7A36D60-2496-v8-3-28-2023_2-18-17-AM-.log
Top Calls
Bottom-up
Top-down
Flame Chart
GC
engine
external
execution
500ms
600ms
700m

Function	File
(unknown)	C:\\Program Files\\nodejs\\node.exe
~addBuiltinLibsToObject (LazyCompile)	helpers:142:32
~REPLServer.createContext (LazyCompile)	node:repl:1060:46
~REPLServer.resetContext (LazyCompile)	node:repl:1110:45
~REPLServer (LazyCompile)	node:repl:223:20
~start (LazyCompile)	node:repl:1033:15
~createRepl (LazyCompile)	repl:16:20
~<anonymous> (LazyCompile)	repl:36:21
~loadESM (LazyCompile)	esm_loader:85:41
(unknown)	C:\\Program Files\\nodejs\\node.exe

Experience:

Memory profiling is a good tool to use to check the performance and memory usage of our project. It gives a visual representation of the memory usage. However, it is a little hard to use. We have to spend a good while to find out how to set it up and use it. And there are limited resources available to help with using it. Also, our project is not too big or uses a lot of memory. Thus, it is a good tool to have to show if the memory usage is normal but not necessary.

5. Program Tracing: Tracing is the process of monitoring a program's execution to determine which lines of code are executed and in what order. In Python, we can easily implement tracing by using the sys.settrace() function to set a tracing function that is called for each line of code executed by our program. Program tracing can provide several benefits:
    a. Debugging: Tracing allows us to identify specific lines of codes that are causing errors or unexpected behavior.
    b. Performance Analysis: By tracing execution of our program, we can easily identify which parts of our code are taking the most time to execute, and optimize accordingly.
    c. Code optimization: It can help identify redundant or unnecessary code to improve code efficiency.
    d. Testing: Program tracing can be used to create tests that cover different parts of our code. Tracing can ensure that the program is functioning as expected.



```
Call to __init__ on line 8 of /Users/kristinerivera/Documents/GitHub/projects/OpenTab/main_window.py
Call to __get__ on line 176 of /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/types.py
Call to value on line 801 of /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/enum.py
Call to __get__ on line 176 of /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/types.py
Call to value on line 801 of /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/enum.py
Call to login_clicked on line 76 of /Users/kristinerivera/Documents/GitHub/projects/OpenTab/main_window.py
Call to __init__ on line 8 of /Users/kristinerivera/Documents/GitHub/projects/OpenTab/login_window.py
Call to __init__ on line 5 of /Users/kristinerivera/Documents/GitHub/projects/OpenTab/user_db.py
Call to __get__ on line 176 of /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/types.py
Call to value on line 801 of /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/enum.py
Call to login on line 61 of /Users/kristinerivera/Documents/GitHub/projects/OpenTab/login_window.py
Call to get_user on line 34 of /Users/kristinerivera/Documents/GitHub/projects/OpenTab/user_db.py
Call to close_app on line 80 of /Users/kristinerivera/Documents/GitHub/projects/OpenTab/main_window.py
Call to _shutdown on line 1518 of /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/threading.py
Call to ident on line 1145 of /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/threading.py
Call to _stop on line 1028 of /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/threading.py
Call to daemon on line 1183 of /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/threading.py
Call to _maintain_shutdown_locks on line 800 of /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/threading.py
Call to <listcomp> on line 810 of /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/threading.py
Call to <lambda> on line 218 of /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/rlcompleter.py
Call to __getattr__ on line 423 of /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/enum.py
Call to _is_dunder on line 22 of /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/enum.py
Call to __getattr__ on line 423 of /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/enum.py
```

Above: Lines generated during program tracing.

Experience:

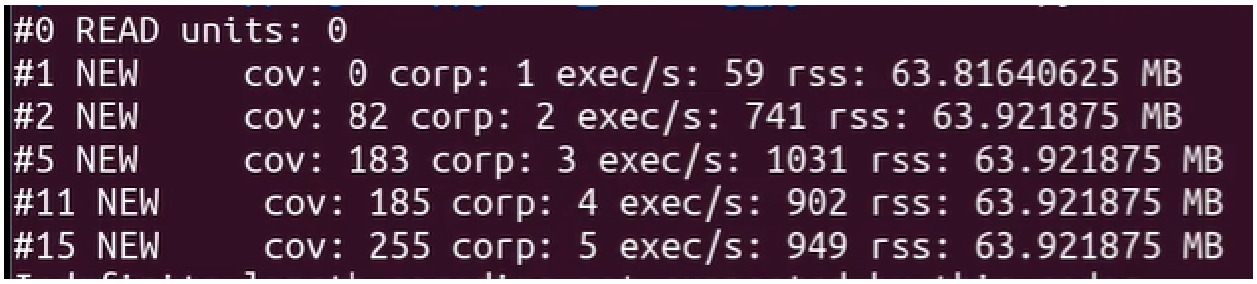
In the event of the app crashing, the time it takes going through each line to indicate which line of code made the error is very tedious.

Attack surface review:

There haven't been any new changes, updates or patches with our approved tools list. Also, there are no new vulnerabilities being reported for any of those tools.

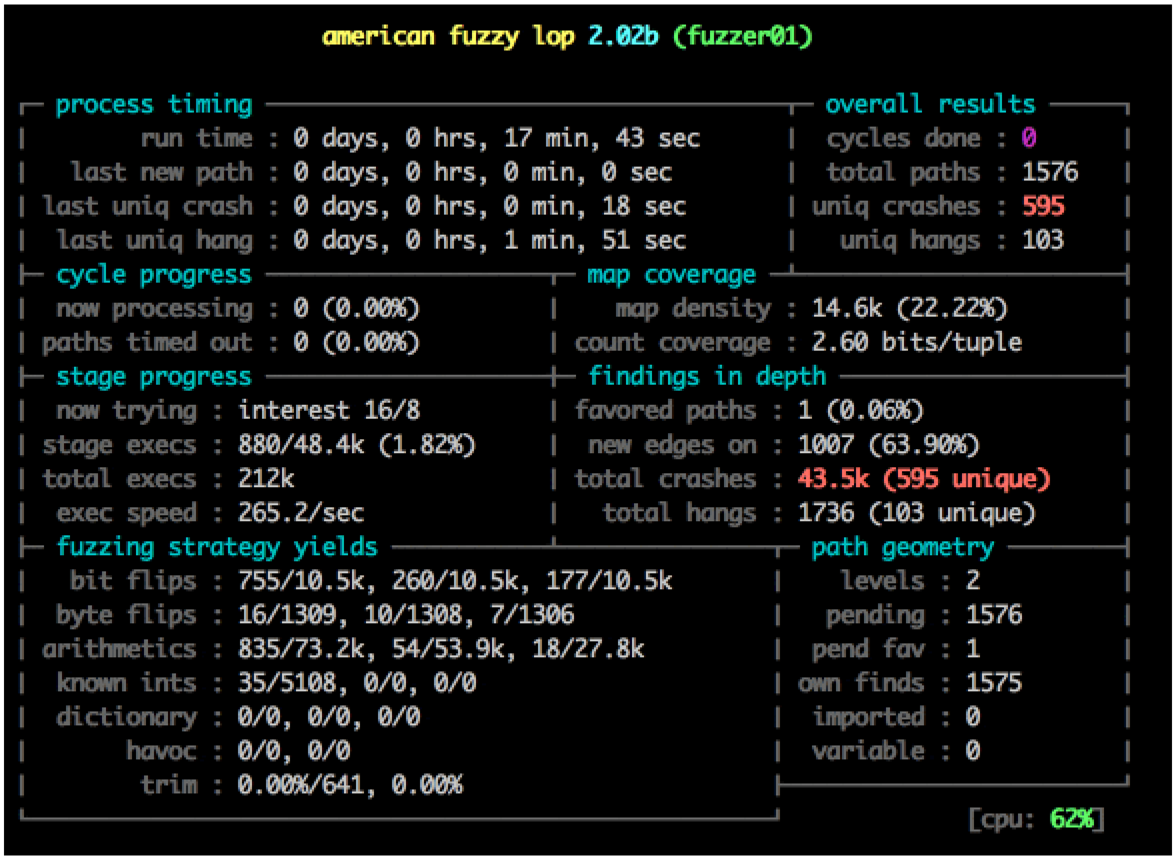
Fuzz Testing:

The fuzzer we are using is called PythonFuzz which is a coverage-guided fuzzer for testing python packages. It was acquired from here. It is a powerful strategy for finding bugs like unhandled exceptions, logic bugs, security bugs that arise from both logic bugs and DoS caused by hangs and excessive memory usage.



```
#0 READ units: 0
#1 NEW     cov: 0 corp: 1 exec/s: 59 rss: 63.81640625 MB
#2 NEW     cov: 82 corp: 2 exec/s: 741 rss: 63.921875 MB
#5 NEW     cov: 183 corp: 3 exec/s: 1031 rss: 63.921875 MB
#11 NEW     cov: 185 corp: 4 exec/s: 902 rss: 63.921875 MB
#15 NEW     cov: 255 corp: 5 exec/s: 949 rss: 63.921875 MB
```

We have also used AFLpy (American fuzzy lop python) for a try-out and comparison. It is a security-oriented fuzzer that employs a novel type of compile-time instrumentation and genetic algorithms to automatically discover clean, interesting test cases that trigger new internal states in the targeted binary. This substantially improves the functional coverage for the fuzz code.



```
                    american fuzzy lop 2.02b (fuzzer01)

┌─ process timing ─────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 0 hrs, 17 min, 43 sec       │  cycles done : 0     │
│   last new path : 0 days, 0 hrs, 0 min, 0 sec         │  total paths : 1576  │
│ last uniq crash : 0 days, 0 hrs, 0 min, 18 sec        │ uniq crashes : 595   │
│  last uniq hang : 0 days, 0 hrs, 1 min, 51 sec        │   uniq hangs : 103   │
├─ cycle progress ────────────────────┬─ map coverage ─┴──────────────────────┤
│  now processing : 0 (0.00%)         │    map density : 14.6k (22.22%)       │
│ paths timed out : 0 (0.00%)         │ count coverage : 2.60 bits/tuple      │
├─ stage progress ────────────────────┼─ findings in depth ───────────────────┤
│  now trying : interest 16/8         │ favored paths : 1 (0.06%)             │
│ stage execs : 880/48.4k (1.82%)     │  new edges on : 1007 (63.90%)         │
│ total execs : 212k                  │ total crashes : 43.5k (595 unique)    │
│  exec speed : 265.2/sec             │   total hangs : 1736 (103 unique)     │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ───────┤
│   bit flips : 755/10.5k, 260/10.5k, 177/10.5k       │    levels : 2         │
│  byte flips : 16/1309, 10/1308, 7/1306              │   pending : 1576      │
│ arithmetics : 835/73.2k, 54/53.9k, 18/27.8k         │  pend fav : 1         │
│  known ints : 35/5108, 0/0, 0/0                     │ own finds : 1575      │
│  dictionary : 0/0, 0/0, 0/0                         │  imported : 0         │
│       havoc : 0/0, 0/0                              │  variable : 0         │
│        trim : 0.00%/641, 0.00%                      ├───────────────────────┘
└─────────────────────────────────────────────────────┘          [cpu: 62%]
```

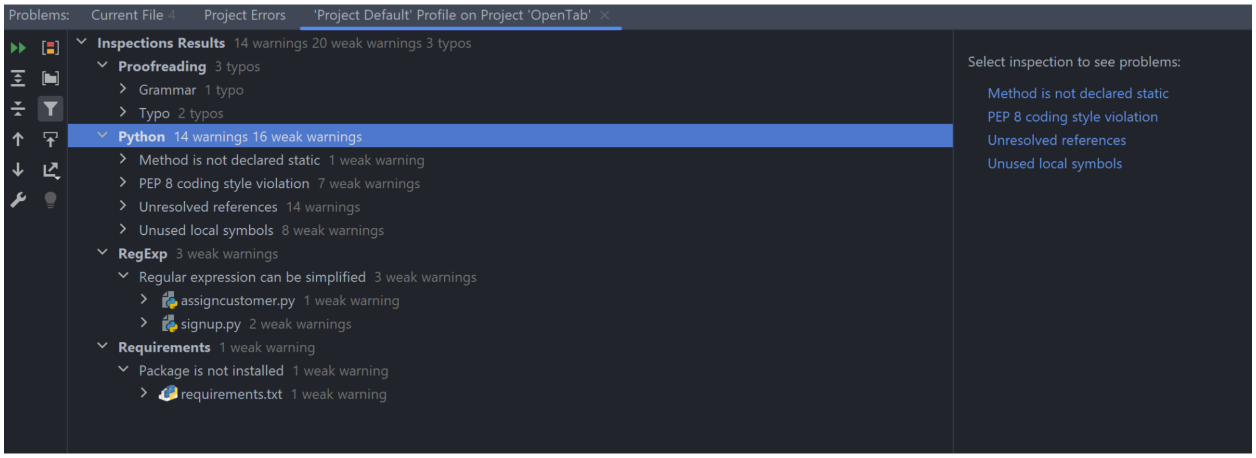
We had a hard time finding a good fuzz test tool as most of these tools have been discontinued which means it was out of date even though it might still perform the task it may encounter some problems. Also, these are kind of hard to try out as very few resources are out there to provide guidance and also since it was discontinued there are some outdated dependencies that can not be installed anymore. We were able to run some fuzzing input to the login window with a lot of mutated input strings which have not yet produced any problem nor vulnerabilities. However, the testing will still be undergoing.

Fuzz 1, we have attempted to conduct SQL injection attacks on our program since our project has a database interaction. In each input string, we have attempted to add some query commands to see if it can affect our database. We have run 30 attacks with different SQL query and none of them has been successful. The reason why is we have used parameterized queries such as in get_user() function we have used {"SELECT * FROM users WHERE username = ? AND password = ?", (username, password) } instead of directly using the user input. Also, we have sanitized user input by using check_password(), submit() function to verify the input. Moreover, we have limited the user privileges according to their roles, such as admin is the only one who has the right to do all queries on the user database. And only the users in the user database have the right to read, write to the customer database. Therefore, most SQL injections will not be able to succeed in our database.

Fuzz 2, we have attempted to conduct DoS attacks on our program since our program takes input from users. After massive attempts of inputs, we weren't able to crash our program however it did get slower. Even though we did not use load balancers or rate limiting, the program is intended to be used locally with only a couple machines. The program naturally limited the rate by accessing one at a time. However, if this program is used widely, then we have to implement the features to prevent the DoS attacks.

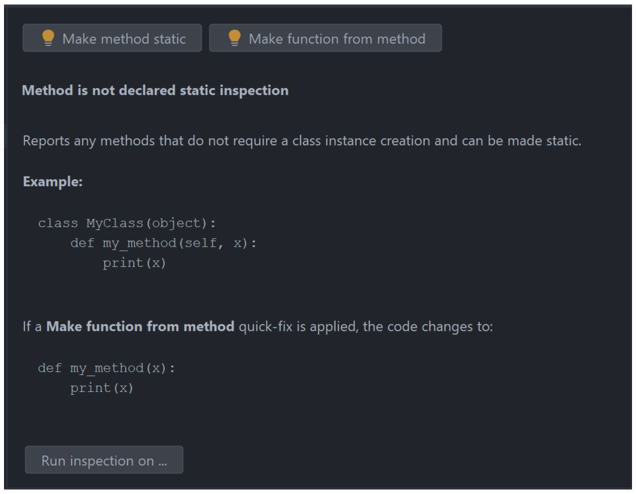
Fuzz 3, we have attempted to conduct buffer overflow attacks on our project as there is an input buffer. We conduct this attack by passing a very long string to each field where the input is needed. We were not able to archive the attack due the fact that the QLineEdit() from the PySide6 has a default size limitation of 32767 characters which will dump the contents that passed that size. Therefore, the buffer overflow attack fails.

Static Analysis Review:



```
Problems:  Current File 4   Project Errors   'Project Default' Profile on Project 'OpenTab'

Inspections Results  14 warnings 20 weak warnings 3 typos
  Proofreading  3 typos
    Grammar  1 typo
    Typo  2 typos
  Python  14 warnings 16 weak warnings
    Method is not declared static  1 weak warning
    PEP 8 coding style violation  7 weak warnings
    Unresolved references  14 warnings
    Unused local symbols  8 weak warnings
  RegExp  3 weak warnings
    Regular expression can be simplified  3 weak warnings
      assigncustomer.py  1 weak warning
      signup.py  2 weak warnings
  Requirements  1 weak warning
    Package is not installed  1 weak warning
      requirements.txt  1 weak warning

Select inspection to see problems:
  Method is not declared static
  PEP 8 coding style violation
  Unresolved references
  Unused local symbols
```

The static analyzer we are using is the PyCharm Code Inspector. After we conduct the inspection, there are 14 warnings, 20 weak warnings and 3 typos have been found which also provides some detailed information about the problem and possible solutions as follow.



Method is not declared static inspection

Reports any methods that do not require a class instance creation and can be made static.

Example:

```
class MyClass(object):
    def my_method(self, x):
        print(x)
```

If a **Make function from method** quick-fix is applied, the code changes to:

```
def my_method(x):
    print(x)
```

We were able to fix typos and some warnings even though they are not considered as any type of vulnerabilities. There are some unresolved reference warnings which we can fix as the function is binded into the signal which somehow didn't get references from the imported package. However, after testing, we find it will not affect the performance nor create vulnerabilities. The most important issue we found during the static analysis is the package version issue. At the time this project started, the PySide was at version 6.4.1 and now it was updated to 6.5.0. It is always a good idea to update the package to the latest version as it may have some vulnerabilities or errors fixed during the update. At the time of the submission of this report, we have removed all the warnings (except the reference ones) that are generated during the static code analyzing process.

Dynamic Review:

Out of these 5 dynamic analysis tools we have tried during assignment 3, we have decided to use the most important 3 tools that we think suit our project best which are debugging, coverage analysis and program tracing. We have implemented some test cases (which are stored locally) to test the input to the program during the login process, signup process, and adding customer process, also we have tested all the functionalities of all the buttons. Using the debugging tool, we are able to see step by

step process of the testing which gives us a clear picture of the process of the program handling the input. There are no bugs found during the testing and debugging process. Then we are using the coverage analysis to check the coverage of the code. At the beginning it only reached about 50% coverage due to the limitation of testing implementation, therefore, we manually tested all possible branches. After a couple rounds of testing, the coverage was able to reach 100% which means all the lines of code can be executed by our program and there is no dead end. Lastly, we used the program tracing to see the flow of the program where we found all the functions and classes are called in the order as we designed. Also, the performance time of our program is at a normal range (disregard the time of user input). The fuzzer mentioned above has been able to perform SQL injection, DoS attack and buffer overflow attack. The process and result have been provided above.
README can be accessed by
https://github.com/cybershield427/OpenTab

Release:

1. **Incident - Response Plan:**
    a. **Privacy Escalation Team:**
        i. In case of any future threats to our software, our Privacy Escalation Team will be responsible for handling the situation promptly and efficiently. The team will consist of the following members:
            1. Escalation Manager: Hangbo Zhang
               The escalation manager will be responsible for coordinating the incident response plan and ensuring that all team members are informed about the situation. The escalation manager will also work closely with the Security Engineer to assess the nature and scope of the incident, and determine the appropriate response.
            2. Legal Representative: Kristine Rivera

The Legal representative will provide guidance on any legal issues related to the incident with external legal counsel if necessary.

3. Public Relations Representative: Kristine Rivera
   The Public Relations Representative will be responsible for communicating with the public which include our customers and media about the incident and the actions taken to address it.
4. Security Engineer: Hangbo Zhang
   The Security Engineer will be responsible for investigating the incident and identifying any security vulnerabilities that were exploited. They will work closely with the escalation manager to determine the appropriate response.

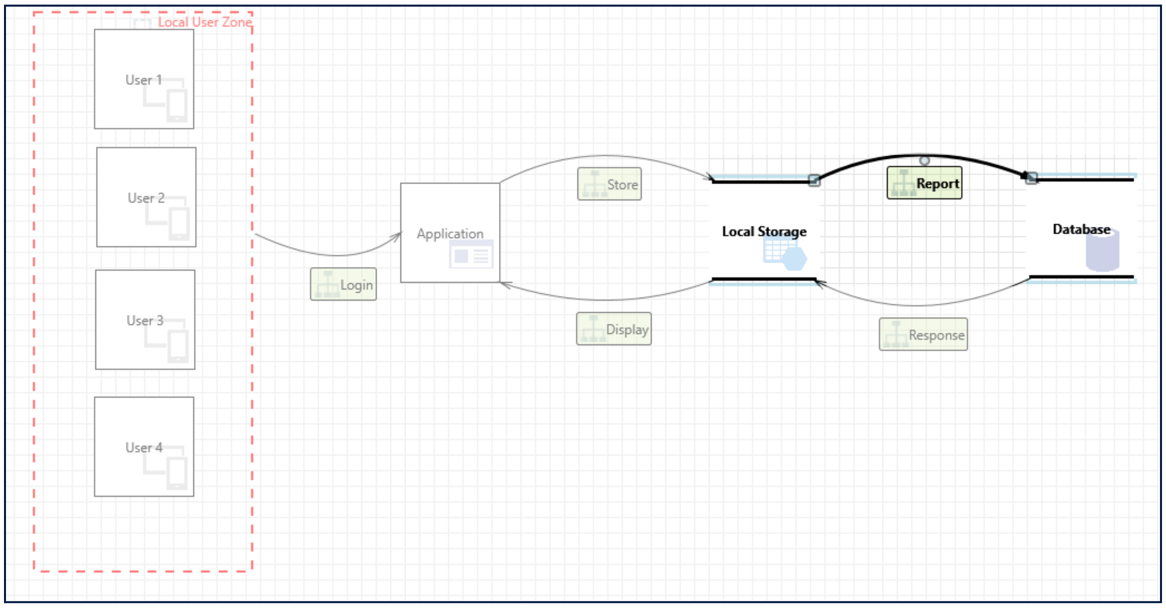
b. **Contact Email:**
   The email address dedicated for users to contact our team in case of an emergency is: incidentresponse@opentab.com. This email address will be monitored by all members of the Privacy Escalation Team, and any emails received will be addressed as quickly as possible.

c. **Procedures:**
   1. Notify the Escalation Manager immediately of any incidents.
   2. The Escalation Manager will assess the situation and initiate the Privacy Escalation Team.
   3. The Security Engineer will investigate the incident and determine the scope and nature to the threat.
   4. The Legal Representative will provide guidance on any legal issues related to the incident with external legal counsel if necessary.
   5. The Public Relations Representative will communicate with the public about the incident and the actions being taken to address it.
   6. The team will work together to develop and implement a plan to address the incident, mitigate any damage, and prevent future incidents.

7. Document all details of the incident, including the source of the incident, scope of the incident, the affected systems and data, as well as the procedures used to resolve and mitigate the incident. This documentation should be stored securely and should be accessible to all members of the Privacy Escalation Team for future references.
8. Once the incident has been resolved, the team will conduct a post-incident review to identify any areas for improvement in our incident response plan.

2. Final Security Review

Our threat model remained unchanged throughout the development of our SDLC.



All static and dynamic analysis errors have been corrected and our database log indicates that no issues are present. We conducted additional penetration testing on our application that was limited in scope to not include our generated databases. After completing our tests and analyzing the results, we settled on a security assessment grade of Passed FSR. Our application prevents users from creating accounts with the same email address and username, forces users to use strong passwords, prevents users from inserting malicious code into text fields then executed in the database, user passwords are not stored in plain text, and user windows are separated with all private information protected.

3. **Certified Release & Archive Report**
    a. **Our release version of the program can be found:**
    https://github.com/cybershield427/OpenTab/releases/tag/v1.0.0
    b. **Summary of Features, Version Number, Future Development Plans:**

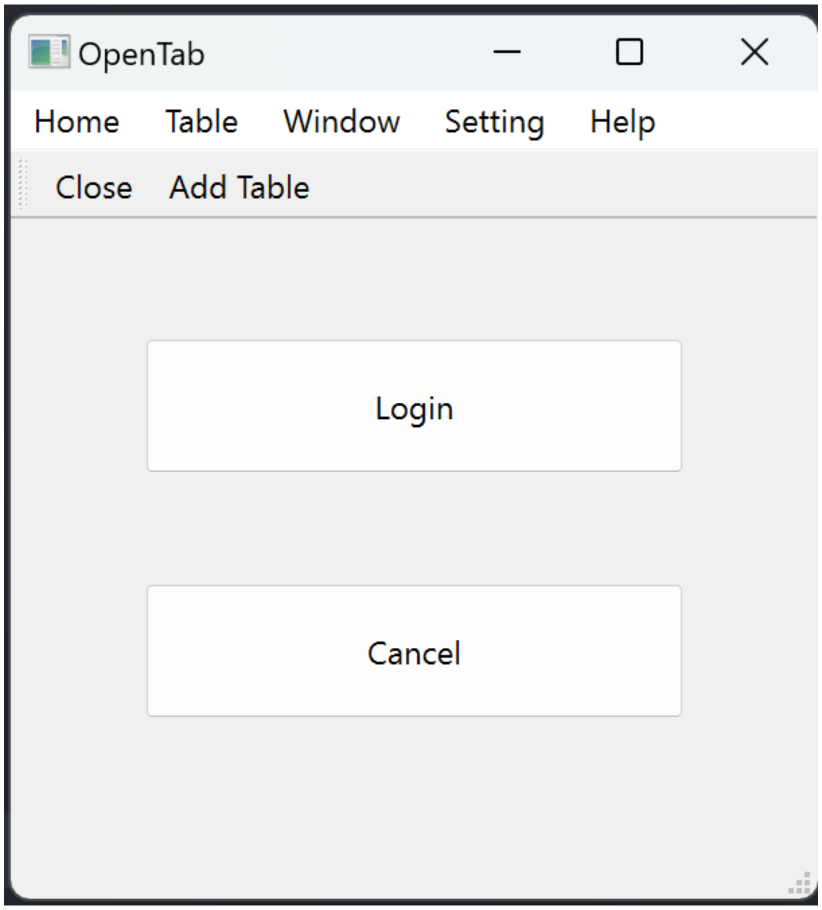
    Summary of Features:
    - Allows users to seat customers to available tables in a restaurant
    - Stores sensitive information such as names, phone numbers, and credit card information until customers is ready for payment
    - Manage the waiters/waitress accounts

    Version number: 1.0.0

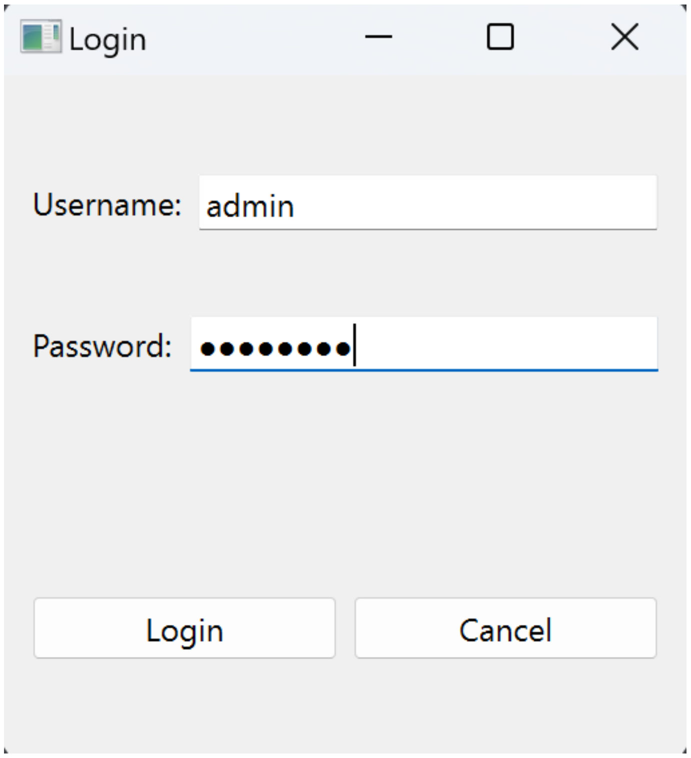
    Future Development Plans:
    - Generate receipts for each transaction
    - Allows users to view reports on customer data and transactions
    - Offers customizable themes and layouts for the user interface
    - Add the ability to store orders for each customer
    - Add the ability to calculate the total payment amount per customer
    - Add the ability to split bills among customers at a table
    - Enhance the security measures to better protect sensitive customer information
    - Develop a mobile version of the application
    - Implement integration with popular payment gateways for seamless transactions

    c. **Technical Notes:**
    For detailed information on how to use our program, please refer to our User Guide wiki page, which can be found at:
    https://github.com/cybershield427/OpenTab/wiki/User-Guide

- The software requires a Windows operating system (Windows 7 or Later) and at least 1 GB of RAM to run smoothly
- Install Pyside6 or running this command in the terminal:
  `pip install pyside6`
- Go to https://github.com/cybershield427/OpenTab and click the “clone or download” button to download the repository to your local file system. Using the GitHub Desktop is a great choice if you use Windows or MacOS.
- Use a terminal application or IDE to cd into the directory of your local copy.
- Run the application by running this command in the terminal:
  `python3 main.py` this will start the application immediately which will show as the following



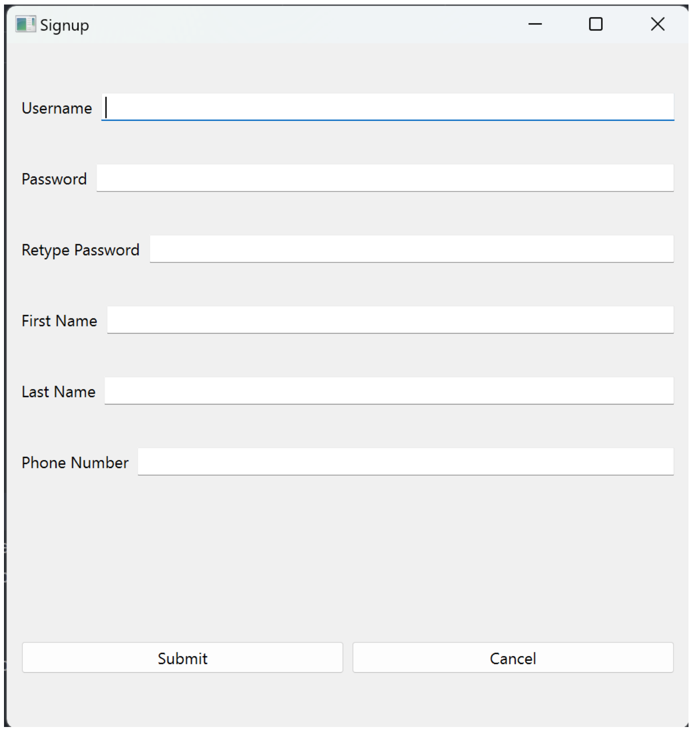
- You can login with `admin` and `changeme`. From then, you can sign up for all your waiters/waitresses.



Login

Username: admin

Password: ●●●●●●●●



Signup

Username

Password

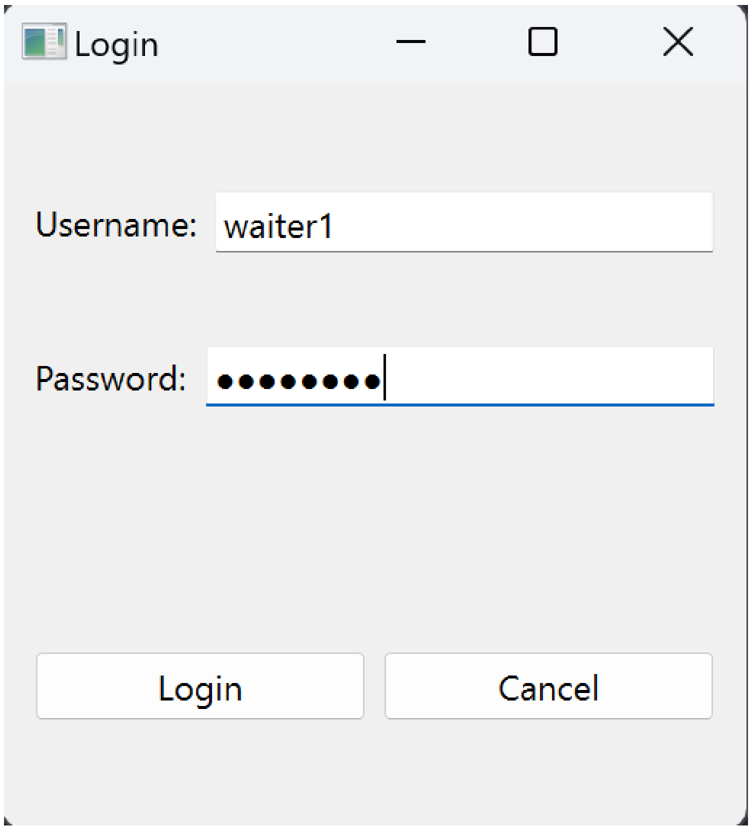
Retype Password

First Name

Last Name

Phone Number

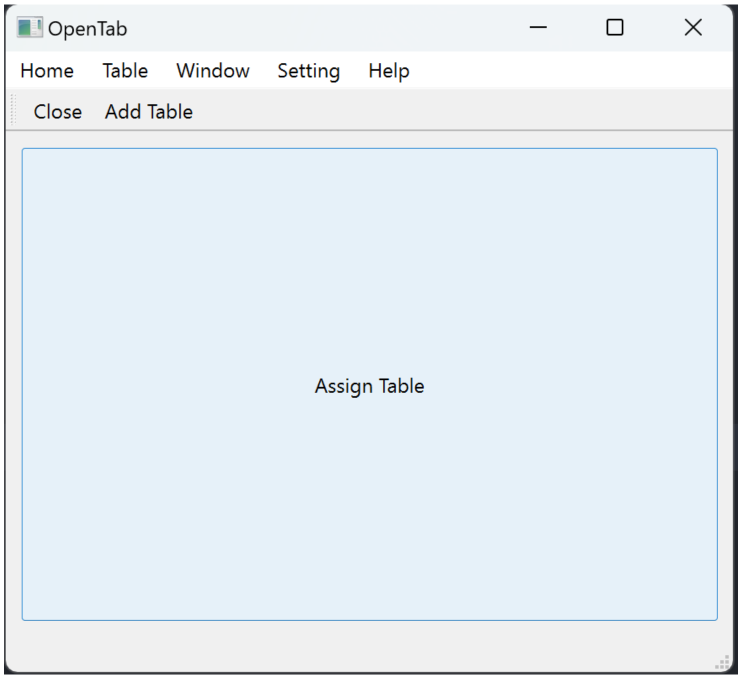
- For waiters/waitresses, after the sign up is finished by the manager/admin, they can login with their own username and password.

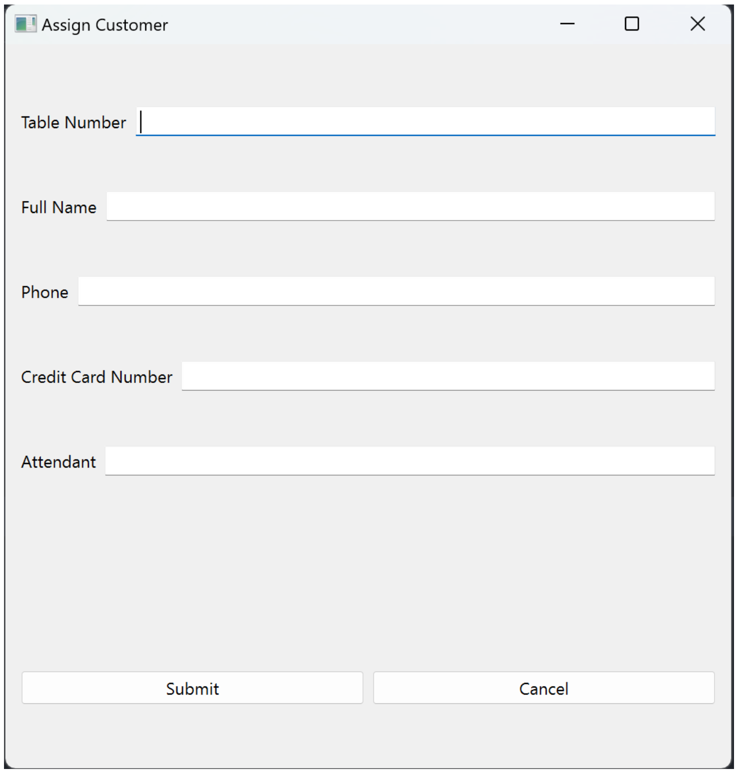


Username: waiter1

Password: ●●●●●●●●

- After login they can add tables, assign customers, store or delete customers' information of the tables. However, they will only be able to see the name and orders of the customer, not any other information of the customers.





- For technical support, please email our team at support@opentab.com