Threat Pilot
Software Design Document

First Edition

By Sara Shikhhassan 101142208,
Sam Al Zoubi 101140949,
Tejash Patel 101131066,
Jatin Kumar 101092120

Table of Contents

1. INTRODUCTION

1.1 Purpose

This software design document (SDD) is used to describe and monitor the information required to properly define the Threat Pilot threat modelling tool's architecture and system design. The SDD will advise the development team and future project teams on the architecture of the system to be developed. Based on the project's specific conditions and the software development technique used to construct the system, the design document will be developed gradually and iteratively throughout the system development life cycle.

Threat Pilot is a new threat modelling tool designed to simplify and accelerate the threat modelling process for software development projects. The tool will give developers an intuitive design for discovering possible risks and vulnerabilities in their software, as well as recommendations on how to resolve these concerns.

The design and architecture of the proposed system will be developed with a focus on simplicity, ease of use, and scalability. The system will be designed to be flexible and adaptable to different types of software development projects and methodologies.

In summary, this SDD will provide a comprehensive guide for Threat Pilot development. The tool's functional and non-functional requirements, system architecture, and user interface design are all outlined in this paper. The SDD acts as a resource for future capstone project students who will be responsible for implementing and enhancing the tool by providing them with the knowledge they need to enable the tool's successful development to effectively handle the threats and risks that organizations face.

1.2 Audience

The software design document is intended for a diverse audience comprising of future project developers, stakeholders, future project maintainers, and potential clients. The primary audience for this document is the future year student teams, who will utilize it to gain insight into the software's architecture, design, and implementation details. Dr. Jason Jaskolka and the second reader are important stakeholders who may also utilize the document to understand the technical aspects of the software and provide feedback on the design. Future project maintainers may use this document as a reference when making changes or updates to the software. Furthermore, certain sections of the document, such as the user interface (UI), may be shared with potential clients and other stakeholders who require input/approval into the UI's development.

1.3 System Overview Summary

Threat Pilot is a threat modelling tool designed to simplify and accelerate the threat modelling process for software development projects. The system's architecture is designed with a focus on simplicity, ease of use, and scalability, allowing it to be flexible and adaptable to different types of software development projects and methodologies.

To create the Python web application, a web framework such as Flask or Django should be used, and version control systems like Git can be used to track changes to the codebase and collaborate with other developers. The web application will require a MySQL database to store and retrieve data, and a web server like Apache or Nginx will be used to serve the Python web application to the internet. The data will be stored in a MySQL relational database, holding the project management, element management, and threat management component data.

The overall system architecture must have characteristics such as extensibility, modifiability, maintainability, installability, upgradeability, robustness, availability, scalability, security, and performance. To ensure these characteristics, discrete domain components will be generated based on the actions of roles, such as the project management component, element management component, reporting engine component, threat management component, and system model management component.

Threat Pilots architecture is presented using a modular monolithic style, with components divided into modules that are loosely coupled and independent within the tool and connected to Threat Pilots interface on one end and to a central SQL Database that all components share. Each module contains layers API, user interface, business logic, services, and data interface. There is synchronous communication amongst the independent modules to work correctly through public APIs, allowing access to the logic of each module from the other using public methods, and not using the internal functions.

1.4 Future Contingencies

A few future contingencies that are relevant to the development of a threat modelling tool are the emergence of new types of threats that could require the tool to integrate with new threat modelling techniques to handle these new threats. Increase information sharing amongst developers that may require the tool to handle changes in requirments. Overall the ability of the threat modelling tool to connect with other tools and systems, adapt to new threats, technologies, and techniques, and identify and reduce potential security concerns will make it more useful to developers.

1.5 Assumptions

The Threat Pilot project is based on certain assumptions, which are as follows:

Firstly, future development teams are expected to possess the necessary technical knowledge and skills to build the system, or they should be willing to acquire them.

Secondly, the current software design document has been created to the best of the team's collective knowledge. However, it is acknowledged that challenges may arise during the actual development process, which is typical for any software development project. In such situations, the development team should carefully evaluate the situation and make appropriate updates to the software design document as necessary.

2. DESIGN CONSIDERATIONS

2.1 Operational Environment

The operational environment for the Threat Pilot application will consist of the following components:

- Django/Flask Python Development: A web framework like Flask or Django will be utilized to build the Python web application. These frameworks provide a comprehensive set of tools and libraries to develop web applications quickly and efficiently.
- Git Version Control: The codebase for the application will be managed using a version control system such as Git, allowing for changes to be tracked and facilitating collaboration among developers.
- Github Repository: The application codebase will be stored remotely in a free and easy-to-use Git repository on Github.
- MySQL Database: The application will require a MySQL database to manage data storage and retrieval for the web application.
- Apache/NGINX Web Server: A web server such as Apache or NGINX will be used to serve the Python web application to the internet, providing a reliable and secure means for users to access the application.

3. SYSTEM ARCHITECTURE

3.1 Project Scope

Threat modelling is an important aspect of the system development life cycle because it elicits and understands the system security needs that must be implemented to safeguard the system's specified assets by recognising which portions of the system are most vulnerable to an attack. Threat Pilot, a novel threat modelling tool, is offered to assist system engineers in better eliciting important security needs for their systems. System engineers must have a tool that provides effective and accurate threat management solutions and analysis on their system designs in order to do so. As a result, a tool with clear objectives and usability is critical to reaching this goal. The goal of this project is to elicit Threat Pilot's requirements and software architecture design.

3.2 Functional Requirements

- The system shall enable users to create a new project or load existing models.
- The system shall enable users to import or export part of a diagram that can be used by other models.
- The system shall enable users to input their data using the Data Flow Diagrams (DFD).
- The system shall provide a number of different stencils and each stencil may have different element properties.
- The system shall be able to link the stencils together using another stencil or edge of stencil and each stencil shall have a corresponding icon related to its function.
- The system shall allow stencils to be modified by name and have a list containing separate categories for stencils.

- The system shall use user's input element properties to predict security threats.
- The system shall identify software assets, security controls, and threats and their locations to create a security model.
- The system shall support STRIDE, LINDDUN and CIA framework to generate threats about the vulnerabilities exposed to attackers that might be used to break in.
- The system shall also have threat intelligence from the publicly maintained threat library, MITRE's CAPEC and from proprietary information collected by the developers to know the granular model about what attackers would do after they break in.
- The system shall be scalable to support any other framework or threat library if needed in future.
- The system shall automatically apply suggested threats to the model.
- The system shall be able generate threats on standalone stencils without any other interaction i.e. no data flow to other stencils.
- The system shall create a traceability matrix of missing or weak security controls along with their impact and probability.
- The system shall be able to prioritize the threats using guidelines for quantifying the likelihood and impact of each threat and rank them as high, medium, low or notice priority.
- On identifying threats, the system shall provide information including confidentiality, integrity, availability and ease of exploitation.
- The system shall allow users to add new threats or edit the existing threats to the model.
- The system shall link threats to their corresponding stencil through icons.

- The system shall allow users to edit threats by clicking on the specific stencil to which the threat applies.
- The system shall enable users to view security recommendation reports based on their input at any point in time.
- The system shall generate reports in PDF format and must contain the date, DFD diagrams, summary table, system description, and detailed list of threats.
- The threat details shall contain the name, priority, status, category, description, justification and mitigation of the threat.
- The system shall allow users to save the generated threat reports and the diagrams.
- The system shall allow users to report issues regarding the Threat Pilot system.

3.3 Non-Functional Requirements

Interface Requirements

- The system shall have a Threat dashboard which will allow users to view severity of each vulnerability and asset-level risk.
- The system shall have a Mitigation dashboard which will allow users to see the countermeasures for security control.
- The system shall have detailed documentation about the usability of the tool.

Performance Requirements

- The system shall easily navigate through the project management dashboard.
- The system shall offer smooth experience to users when managing/modifying threats

Architectural Requirements

- The system shall be a web-based application and support the popular web-browsers on the popular operating systems.
- The system shall be available offline.
- The system shall have a github integration.
- The system shall be scalable to integrate with other existing tools/workflow (for example, draw.io, JIRA) and work in conjunction to make the threat modelling process time-efficient.

Compliance Requirements

- The system shall conform to the established look and feel of the Carleton University's branding guidelines including colors and logos.
- The system shall comply with the requirements listed on Carleton University's Fourth year Project website

Development Requirements

- The system's derivables shall follow the deadlines listed on Threat Pilot's proposal timeline.
- The system does not require any other special component, since all of the work can be done on members' personal computers.
- The system shall be free to use for the users.

Accuracy Requirements

- The system shall provide accurate threats based on the user input.
- The system shall perform a check to ensure that client input is valid.

Development Methods

The proposed development method for the Threat Pilot project is the Agile methodology. This approach emphasizes collaboration, flexibility, and iterative development, with a focus on delivering working software quickly and continuously improving the product based on feedback from stakeholders.

The Agile methodology should be implemented through the use of Scrum, which involves the use of short development cycles or sprints. Each sprint should be planned, executed, and reviewed, with a focus on delivering a potentially shippable product increment at the end of each sprint. Daily stand-up meetings will be held to ensure that everyone is aligned and working towards the same goals.

3.4 Monolithic vs Distributed Architecture

Monolithic architecture is an architecture approach in which an application is created as a single, self-contained unit. All application components, including the user interface, business logic, and data access, are tightly coupled and executed as a single process. This means that any modifications implemented to one element of the application need the redeployment of the entire application. Distributed architecture, on the other hand, entails the use of several, independent components that collaborate to achieve a shared purpose. Each component may be installed and run independently, connecting with the others over network protocols. As a result, components may be added or deleted from the system without impacting other components, allowing for increased flexibility, scalability, and fault tolerance.

Distributed architecture is more complex than monolithic architecture because of the multiple independent components that need to be designed, deployed, and maintained. It also requires

more effort to ensure data consistency across different components. So it may also require more technical expertise and resources to implement and maintain a distributed architecture compared to a monolithic architecture. Monolithic is simpler in development, in infrastructure, and troubleshooting and allows for code reuse and data consistency.

When deciding between the two types, we wanted to ensure that future students would have the flexibility to adapt the tool to changing requirements over time and to be able to develop the modelling tool in a relatively easier fashion, thus, we opted for a monolithic style that allows for easy maintenance and scalability.

There are several architectural styles within monolithic architecture. We adopted the modular monolithic approach because it offers the benefits of separate modules while still providing the ease of implementation associated with monolithic architecture. Although each module can be worked on alone, they must be integrated for deployment. It enables a high level of parallelism, ordered code, and scalability. A deconstructed database for individual modules is also an option if the requirements for how data is stored vary over time. This design works well for data-intensive applications, such as storing all threats in a database while maintaining the simplicity of a monolith.

System Architecture

The overall system architecture characteristics: (not ordered by priority)

- Extensibility – the tool must be designed to allow the addition of new capabilities and functionalities.

- Modifiability – the tool must be designed to minimize the cost and trade-offs to which it can be modified efficiently. The ability to admit changes due to a new requirement or by detecting an error that needs to be fixed.
- Maintainability – the tool must be designed with the ease to modify/correct defects and their cause, improve performance and other attributes.
- Installability - the tool must be designed to ease the installation of its software onto desktop platforms on popular operating systems such as MAC and Windows.
- Upgradeability – the tool must be designed to easily upgrade from a previous version of its solution to a newer version on platforms by accepting and executing updates efficiently.
- Robustness – the tool must be designed to handle error and boundary conditions while running if the internet connection goes. As the tool is supported as a web application and on web browsers. It must be able to handle these situations and operate offline.
- Availability – the tool must be available 24/7 and handle failures that may occur during its time of use.
- Scalability – as many users shall eventually be using the tool, the tool must perform and operate efficiently and quickly when the number of users or user requests increases.
- Security – models (DFD and all assets, threats, mitigation strategies that are included with it), user information, project details must be encrypted and kept safe in the database. The database must have preventive measures to counter SQL injection attacks.
- Performance - the tool must be responsive through the time required to respond to specific requests/events or the number of requests processed and handled in a given interval of time.

The architectural characteristics ordered by priority:

- Extensibility
- Maintainability
- Modifiability
- Security
- Availability
- Installability
- Performance
- Robustness
- Scalability

We can identify required components of Threat Pilot using the Actor/Action approach. The roles of the tool are the user and the system itself. And the actions they may take that the tool must listen and accept are listed below.

Role: The User

Actions of User:

- create a new DFD
- load an existing DFD
- add a stencil
- edit a stencil (ex: add element property)
- generate threats (by command button)
- edit a threat
- add a threat
- remove a threat

- create custom template
- generate threat report
- download threat report
- commit to GitHub

Role: The System

Actions of System:

- track user activity
- run rules engine
- integrate with GitHub (allow user commits)
- export threat report

Discrete Domain Components generated based on actions of roles:

Project Management Component – Functionality: add/create/load existing or new data flow diagrams.

Element Management Component – Functionality: adding/removing/updating threats and stencils on the data flow diagram

Reporting Engine Component – Functionality: reporting component for documentation of threats.

Threat Management Component – Functionality: lists of rules and threats that maps to the stencils of the DFD.

System Model Management Component – Functionality: tracks and saves the users input/actions when the user interacts with the tool.

The diagram below shows the actions of each role identified and the components those actions must take place in.

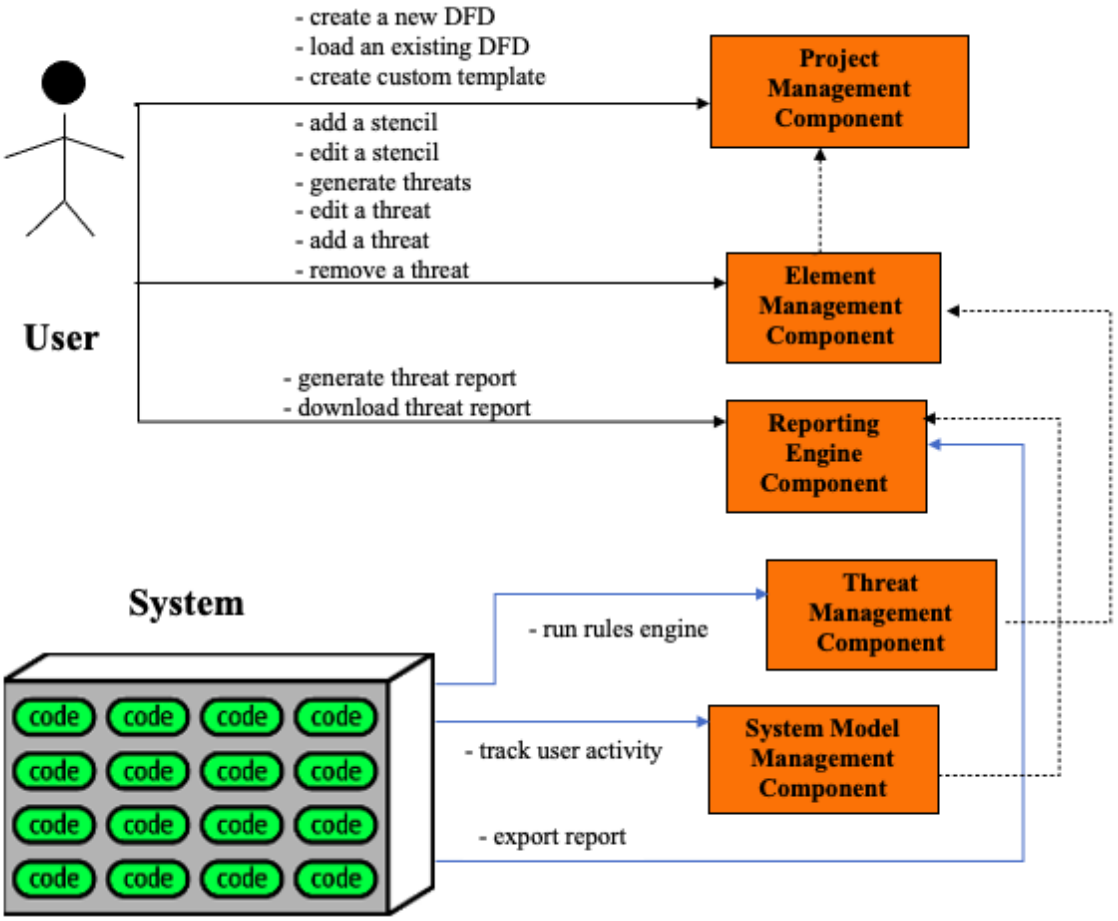

Figure 01: Figure showing the actions of each role identified

3.5 Overall System Architecture

The tool's components (as shown below) are separated into loosely connected and independent modules, each with its own set of layers, which include an API, a user interface, business logic, services, and a data interface. Each component has a specific responsibility. Using their data interface layers, all modules connect with the primary MySQL relational

database, which they all share. This technique promotes synchronous communication across modules to ensure appropriate operation. To guarantee consistency, all Threat Pilot components are created with the same architectural criteria specified earlier in the project.

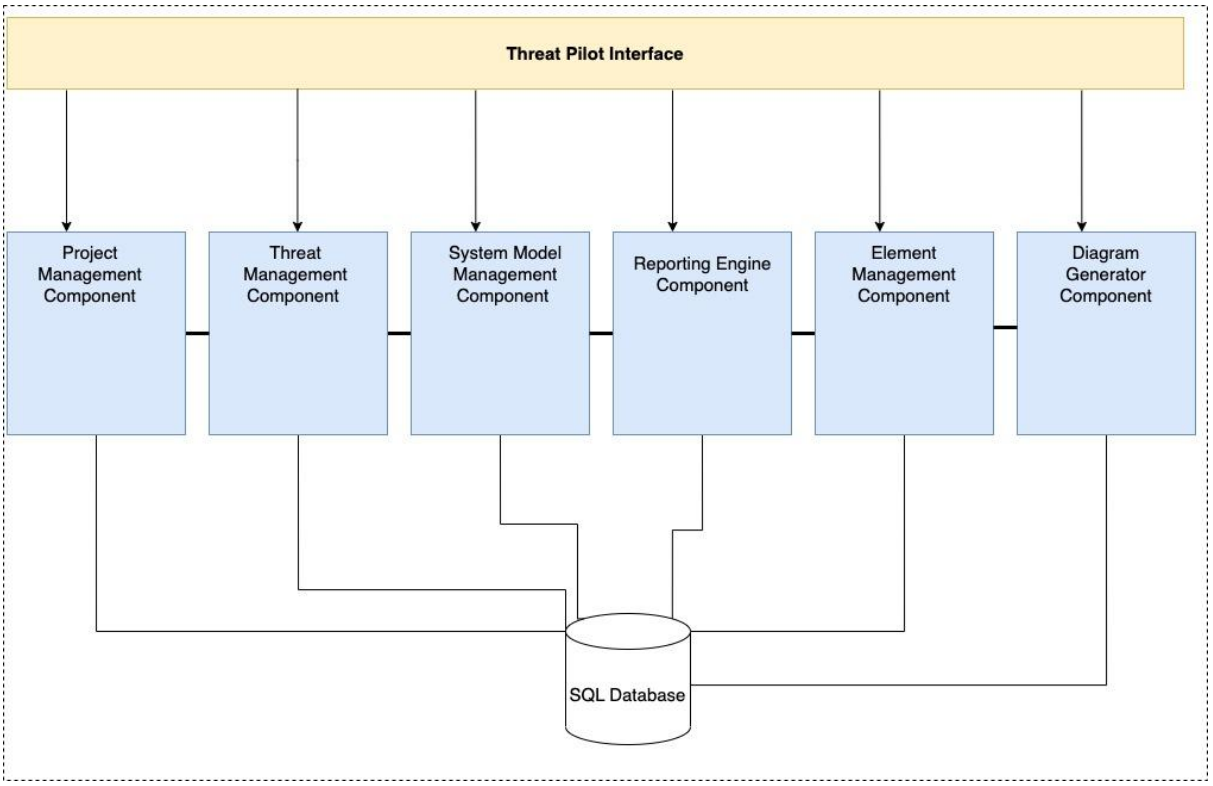

Figure 02: Components of Threat Pilot Application Server

We chose an architecture in which each module has its own set of layers, including an API layer that enables public access to each module's functionality and a data interface layer that permits data manipulation inside the central MySQL database. The modules can synchronously communicate with one another without relying on each other's internal operations or logic by using public APIs, guaranteeing loose coupling and modularity. This is shown by the black arrow connectors that are between the modules as shown in the figure above. Overall, this technique provides a strong and scalable architecture for Threat Pilot. This technique combines the benefits of modular architecture with the simplicity of a monolithic architectural style. The characteristics of extensibility, modularity, low coupling,

high cohesion in the modules, and ease of deployment in a modular monolithic architecture style allows for easy future extension and customization.

The visual representation of the system components of the complete threat pilot can be seen in figure 6 below. The complete project would include external integration such as github, bitbucket, and slack, HTML and pdf report generation, web browser and desktop user interface. The main Threat Pilot application is shown in the middle which includes application logic such as rules engine, threat management, and project management. The API will allow the user interfaces to interact with the Threat Pilot application. The web browser and desktop user interfaces could be a whole project in itself which could be done as a separate project.

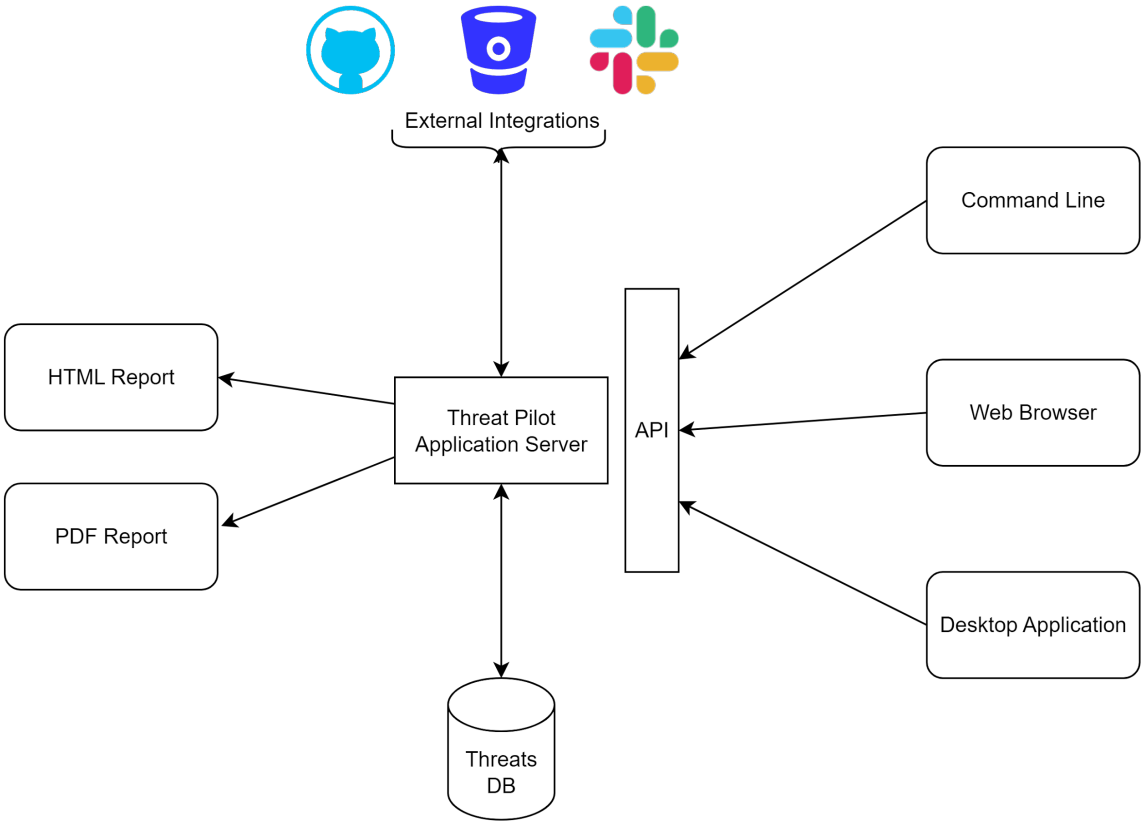

Figure 03: Overall System architecture of Threat Pilot

3.6 Application Server

The Application Server of Threat Pilot represents the core of the system and would handle the processing of data and business logic. It shall communicate with the presentation tier (web application, and desktop application) via the API layer to retrieve user inputs and provide the necessary data for the user interface. The Application tier will be implemented using server-side frameworks such as Ruby on Rails, Django, or Express. These frameworks make it easier to write, maintain, and scale the application. It also has wide user adoption and a huge online community to discuss any framework related questions.

The Application Tier would contain the following components: the Project Management Component, the Element Management Component, the Threat Management Component, the System Model Management Component, the Diagram Generator Component, and the Reporting Engine Component. Each component will have a respected API that can interact with the APIs of the other components within the Application Server using RESTful API calls. The API layer can be implemented using the Express.js, Flask, or ASP.NET frameworks. These frameworks are lightweight, require minimal setup, and make write endpoints intuitive and user-friendly.

3.6.1 Project Management Component

The Project Management Component handles the creation and management of projects, including loading existing models and importing/exporting parts of a diagram. Threat Pilot will be designed to enable users to create new data flow diagrams for their system to model and to load existing models to continue operation. It will also allow users to create and open templates. This component encompasses those services into modules.

Field	Type	Mandatory Field	Description
projectID	INT	YES	Unique Identifier
dateOfCreation	Date	YES	Date the project was created
lastModifiedDate	Date	YES	Date the project was last modified
title	String	YES	Title of the Project
ownerName	String	YES	Name of the project owner
description	String	NO	Description of the Project
modelList	List<System Model>	YES	List of all System Models in the Project

Table 1: Object description table for the Project object

Functionality	Path	Return Codes
Get list of all the projects	GET /api/projects/	200 OK 404 Not Found

Get the project details of the project with given project ID	GET /api/projects/*{projectID}*	200 OK 404 Not Found
Create a new project	POST /api/projects/	200 OK 404 Not Found
Update a project with given project ID	PUT /api/projects/*{projectID}*	200 OK 404 Not Found
Deletes a project with given project ID	DELETE /api/projects/*{projectID}*	200 OK 404 Not Found

Table 2: RESTful API endpoints for the Project Management component

Implementation Instructions

- No application logic should be handled in the API for Project Management as its only task is to receive the messages and pass them to the Project Management Component.
- Unit tests shall be added to make sure the correct response code is returned for each endpoint.
- Every step in the service must be logged for easy debugging of issues in the application. Since there is no UI for this service, logging is the only way of finding the root cause of any issues in the past.

3.6.2 Element Management Component

The Element Management Component is responsible for the creation and management of elements, including the different element properties, linking of elements, and editing of elements by name and categories. Some default elements are already created by the application. Default elements include Datastore, Boundary, Dataflow, and Actor. An Element Management Component is essential in threat modelling tools to allow users the flexibility to edit and view their modeled system and select the important element/entities of their system to be modelled.

Field	Type	Mandatory Field	Description
elementID	INT	YES	Unique Identifier
elementImage	BINARY	NO	Optional image of the element
name	String	YES	Name of the element
description	String	NO	Description of the element
additionalPropertiesList	List<AdditionalProperties>	YES	List of all additional properties applicable to the element
category	CATEGORY	NO	Category where the element belongs to

Table 3: Object description table for the Element object

Functionality	Path	Return Codes
Get list of all the elements	GET /api/elements/	200 OK 404 Not Found
Get the element details of the element with given element ID	GET /api/elements/{elementID}	200 OK 404 Not Found
Create a new element	POST /api/elements/	200 OK 404 Not Found
Update an element with given element ID	PUT /api/elements/*{elementID}*	200 OK 404 Not Found
Deletes an element with given element ID	DELETE /api/elements/*{elementID}*	200 OK 404 Not Found

Table 4: RESTful API endpoints for the Element Management component

3.6.3 Threat Management Component

The Threat Management Component uses the user's input data, including the system model, to identify security threats and identify software assets, security controls, and threats and their locations to create a security model. It shall support STRIDE, LINDDUN, and CIA threat categories and have threat intelligence from publicly maintained threat libraries to generate threats and apply suggested threats to the model. It also provides the ability to users to create, delete and update threats and to get a list of all generated threats.

Field	Type	Mandatory Field	Description
threatID	INT	YES	Unique Identifier
framework	FRAMEWORK	YES	Indicates which Framework the threat belongs to
description	String	NO	Description of the threat
condition	String	YES	Boolean condition needed for the threat activation
details	String	NO	Details of the threat
severity	PRIORITY	YES	Severity of the threat (Low, Medium, High)
mitigations	String	NO	Mitigation strategies of the threat
example	String	NO	Some examples of threat

references	String	NO	References for threat
Target	List<Element>	YES	List of elements the threat applies to

Table 5: Object description table for the Threat object

Functionality	Path	Return Codes
Get list of all the threats	GET /api/threats/	200 OK 404 Not Found
Get the threat details of the threat with given threat ID	GET /api/threats/{threatID}	200 OK 404 Not Found
Create a new threat	POST /api/threats/	200 OK 404 Not Found
Update a threat with given threat ID	PUT /api/threats/{threatID}	200 OK 404 Not Found
Deletes a threat with given threat ID	DELETE	200 OK

	/api/threats/{threatID}	404 Not Found

Table 6: RESTful API endpoints for the Threat Management component

3.6.4 System Model Management Component

The System Model Management Component describes the system model administratively and holds details such as model id, elements list, threats list, boundaries list, and dataflow lists during a threat analysis. The details are stored into a database for the user to retrieve when requested and to generate threat reports on runs.

Field	Type	Mandatory Field	Description
systemModelID	INT	YES	Unique Identifier
elementList	List<Element>	YES	List of all elements in the system model
threatsList	List<Threat>	YES	List of all threats in the system model
boundariesList	List<Boundary>	YES	List of all boundaries in the system model
dataflowList	List<Dataflow>	YES	List of all dataflow in the system model

Table 7: Object description table for the System Model object

3.6.5 Diagram Generator Component

The diagrams will be generated using a package called GraphViz. Graphviz is an open source tool for creating graph visualizations. It uses a simple text language called DOT to describe the graphs, and then generates diagrams in various formats, including PNG, PDF, SVG, and

more. This component will consist of two basic sub components responsible for converting graph data to DOT language, and DOT language to PNG format respectively.

Input Processing Sub-Component

The input processing module will receive the Graph Data Structure in an object oriented format and convert it into a DOT language format, which is the language used by Graphviz to generate diagrams. This module will be responsible for ensuring the Graph Data Structure is in the correct format and has all necessary information for generating the diagram.

Graphviz Integration Sub-Component

The Graphviz Integration module will receive the DOT language code generated by the Input Processing module and use the Graphviz package to generate a diagram image in the desired format. This module will be responsible for configuring the Graphviz package and executing it to generate the diagram.

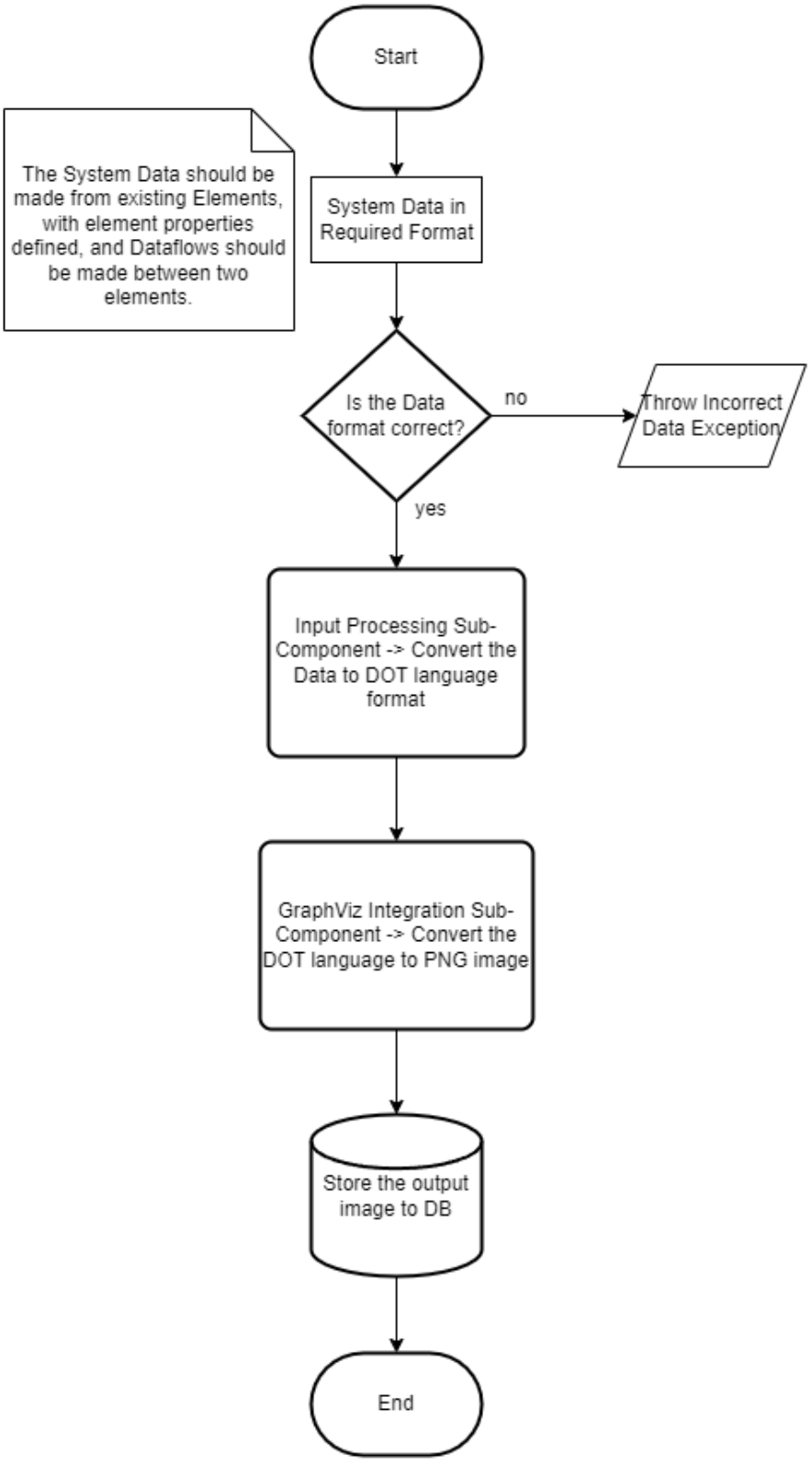

Figure 04: Flowchart for the Diagram Generator Component

3.6.6 Reporting Engine Component

This Reporting Engine Component will generate PDF reports, containing the date, DFD diagrams, summary table, system description, and detailed list of threats, including the name, priority, status, category, description, justification, and mitigation of the threat.

The reports will be generated using the Superformatter template engine. The Python Superformatter engine by ebrehault is a powerful tool that can be used to generate reports in various formats such as PDF, HTML, and Microsoft Word. The engine allows you to define templates that can be populated with data dynamically to generate the final report.

This component will first load the template from a file called template_engine.md. Then the data to be populated in the template needs to be defined as a dictionary. Finally, the superformatter engine will be used to populate the template with the data. Additionally, the weasyprint package can be used to generate the report in PDF format.

Class Diagram

The Class Diagram for the Threat Pilot application model is shown below. The various fields for Objects were inspired from existing threat modelling tools of Microsoft Threat Modelling Tool, and Threat Management Studio.

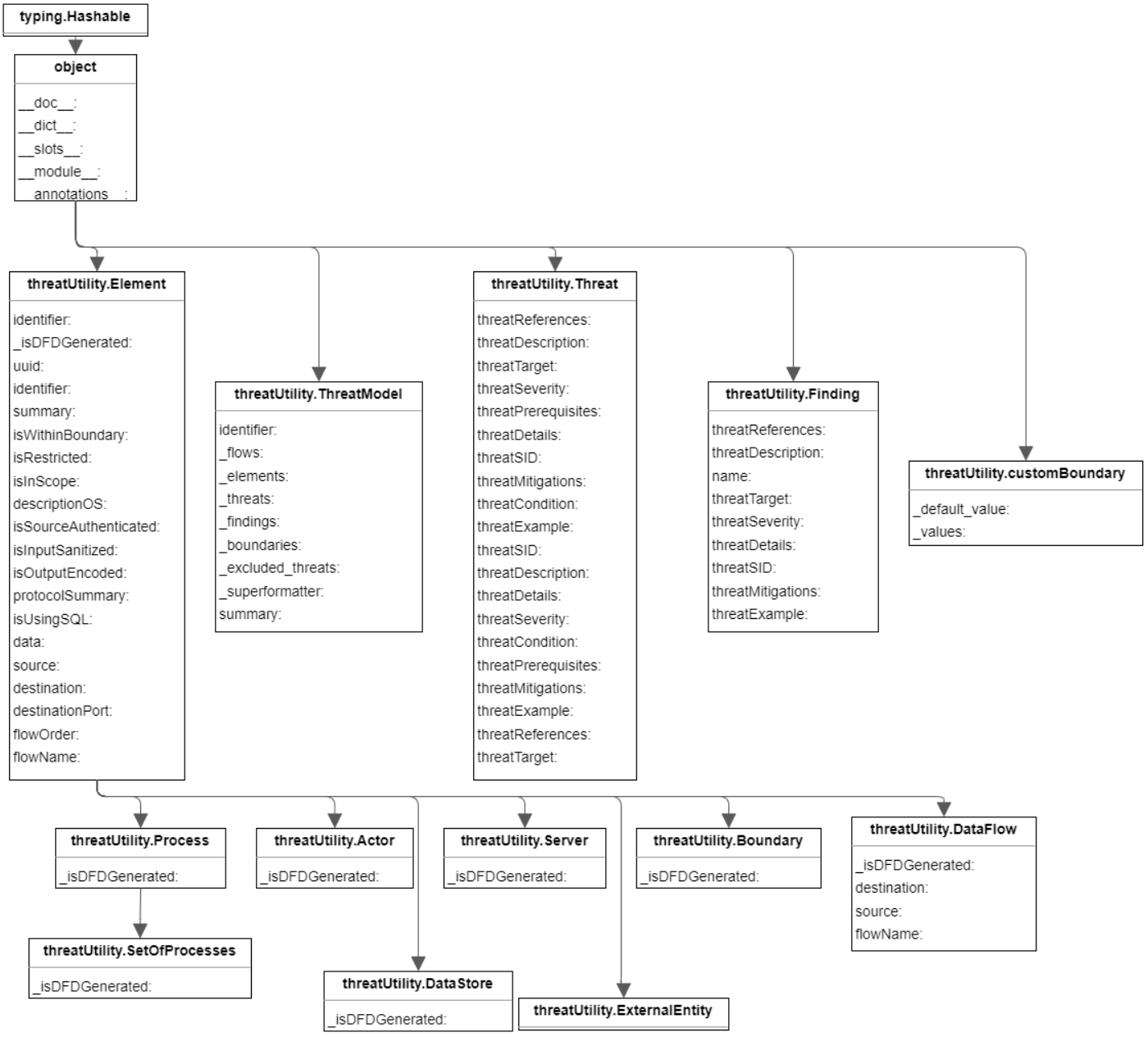

Figure 05: Class Diagram for the Threat Pilot Model

4. DATABASE STRUCTURE

The Threat Pilot data will be stored in a MySQL relational database, holding the project management component data, element management component data, and threat management component data. The components will be stored in the MySQL database as separate tables. The data will be retrieved using SQL queries. Users might then access and change the data as needed to assist their threat modelling operations.

MySQL is a well-known open-source relational database management technology that will complement the Threat Pilot web application. MySQL supports the SQL language, which maintains relational databases and offers capabilities such as data encryption and replication for high availability and scalability, allowing for large and complex databases to grow over time. It is also interoperable with a wide range of web application frameworks and programming languages, which contributes to its scalability. MySQL transactions are ACID-compliant due to the usage of the InnoDB storage engine. MySQL's ACID features are important in this threat modelling tool for the following reasons:

1. Atomicity: Since Threat Pilot modelling sometimes requires many phases, such as adding or deleting assets, recognising threats, and describing countermeasures, ACID-compliant transactions ensure that these activities are handled as a single, indivisible unit of work. To ensure data consistency and correctness, either all or none of the modifications are stored.
2. Consistency: It is critical in Threat Pilot to verify that the data is consistent before and after a transaction. For example, whenever a user adds a new asset to the application, the asset must be stored to the database consistently and correctly. MySQL can assist

guarantee that the data in the threat modelling tool is reliable and correct by enforcing consistency through the use of ACID transactions.

3. Isolation: Many people may access and alter the data at the same time. Concurrent transactions, if not properly isolated, can result in data inconsistencies, such as when one user's modifications are overridden by another's. ACID-compliant transactions guarantee that transactions are executed independently of one another, ensuring that modifications made by one user do not conflict with those made by others.
4. Durability: Threat Pilot offers vital information regarding system's security. The data saved in the database must be durable to avoid loss in the event of system failures or faults. MySQL may assist ensure that the data in the threat modelling tool is permanently stored in the case of a system failure by enforcing durability through the use of ACID-compliant transactions.

4.1 Threat Management Component Table

Table 9: SQL database column types table for Threat Management Table

Column	Type	Mandatory Data	Description
ThreatID*	INT	NOTNULL	Unique identifier for each threat
Framework	ENUM		Framework the threats belong to, could also be a mixture. I.e (CIA) or (CIA, LINDDUN)
Description	VARCHAR		Description of the threat
Condition	VARCHAR	NOTNULL	Boolean condition under which the

			threat can occur
Detail	VARCHAR		Details of the threat
Severity	VARCHAR	NOTNULL	Severity of the threat (“Low”, “Medium”, “High”)
Mitigation	VARCHAR		Mitigation strategies of the threat
Example	VARCHAR		Examples of the threat
Reference	VARCHAR		References for the threat
Target	ENUM	NOTNULL	List of elements the threat applies to

Columns with the asterisk (*) symbols are unique and thus a primary key.

4.2 Element Management Component Table

Table 10: SQL database column types table for Element Management Table

Column	Type	Mandatory Data	Description
ElementID*	INT	NOTNULL	Unique Identifier
ElementImage	BLOB		Optional image of the element
Name	VARCHAR	NOTNULL	Name of the element
Description	VARCHAR		Description of the element
AdditionalProper	ENUM	NOTNULL	List of all additional properties

tiesList			applicable to the element
Category	VARCHAR		Category where the element belongs to (i.e AWS)

Columns with the asterisk (*) symbols are unique and thus a primary key.

4.3 Project Management Component Table

Table 11: SQL database column types table for Project Management Table

Column	Type	Mandatory Data	Description
ProjectID*	INT	NOTNULL	Unique Identifier
DateOfCreation	DATE	NOTNULL	Date the project was created
LastModifiedDate	DATE	NOTNULL	Date the project was last modified
Title	VARCHAR	NOTNULL	Title of the Project
OwnerName	VARCHAR	NOTNULL	Name of the project owner
Description	VARCHAR		Description of the Project
ModelList	ENUM	NOTNULL	List of all System Models in the Project

Columns with the asterisk (*) symbols are unique and thus a primary key.

5. USER INTERFACE DESIGN

5.1 UI Design for the Threat Pilot Web-Application

The User Interface design shall adopt an MVC (Model-View-Controller) design pattern. This design is taken as it shall help in achieving a clear separation between the application's data (Model), user interface (View), and application logic (Controller). By using the MVC design pattern, the Threat Pilot web applications will be easy to develop, maintain, and scale. The Model shall handle the data management, the View shall handle the user interface, and the Controller shall handle the logic that connects the Model and View. This separation allows for independent development and modification of each component. The UI consists of a Home Page, Create a Threat Model Page, Create New Template Page. Open Template Page. Additionally, the UI shall have a link to the documentation. The UI throughout the web application shall have a consistent look and feel to enhance user experience and reduce confusion.

5.2 Web Pages

The following are the templates for the main web pages of the Threat Pilot application.

Home Page

The primary interface of the web application shall present the following options on its home page:

- Create Threat Model: This option shall allow users to create a new threat model.
- Load Existing Model: This option shall enable users to load an existing model.
- Create New Template: This option shall allow users to create a new template.
- Open Template: This option shall allow users to open an existing template.

- Documentation: This button shall be present at the top right of every page, and it shall redirect the user to the Threat Pilot's documentation.

Create a Threat Model Page

- The create threat model page shall feature the following components:
- A list of available Data Flow Diagram (DFD) elements shall be displayed on the left-hand side of the page.
- Available threat frameworks shall be presented on the top right-hand side of the page.
- Each element shall display a list of associated threats.
- A modification panel for threat items shall be located at the bottom of the page.
- The page shall contain the following buttons at the top: Save, Undo, Redo, Export, and Generate Threats, respectively.

Create New Template Page

The create new template page shall feature the following components:

- New Category button: This button shall enable users to create a new category.
- New Threat Type button: This button shall allow users to create a new threat type.
- Delete button: This button shall provide users with the ability to remove a category or threat type.

Open Template Page

The create new template page shall feature the following components:

- Template List: This section shall display all saved templates.
- Template Selection: This section shall allow users to select a template to open.

- Modification Panel: This section shall provide the user with options to make modifications to the selected template.

5.3 UI Diagrams

The following diagrams illustrate the UI design for the desktop application, they are made using Figma (https://www.figma.com) and only the screenshots were available using the free to use service.

HomePage UI Design

The HomePage of the web application shall serve as the primary landing page for the user. It shall provide the user with an intuitive interface that shall allow the user to navigate the various functionalities of the application with ease.

The Threat Pilot's home page will serve as the landing page after the user logs in. It will feature a link to the application's documentation. A logo will be placed in the center of the page, while buttons to load, create a new model, and load templates will be located at the bottom. Users can also open pre-existing templates from this section. Additionally, a hyperlink will be available for users to reach developers through GitHub if they have any queries.

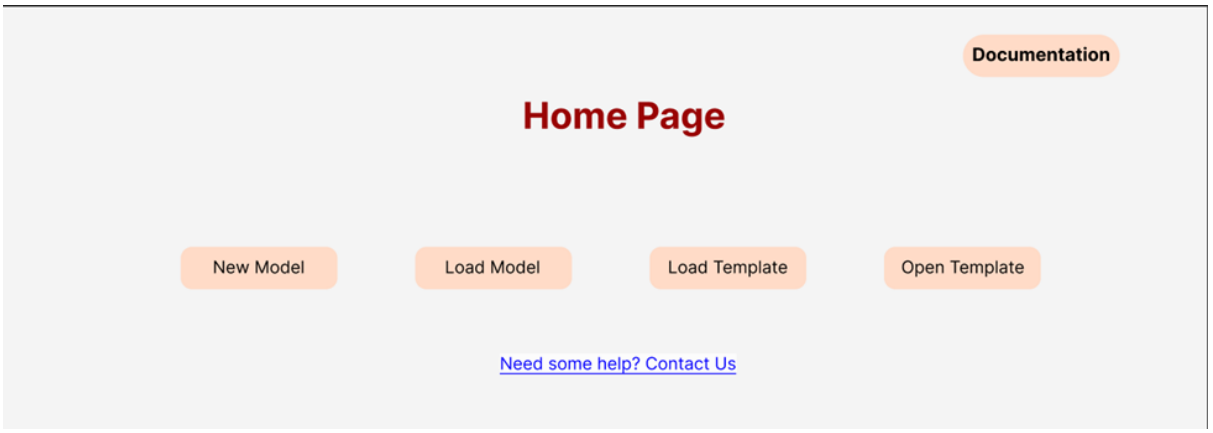

Figure 06: UI design for Home Page of web application

The threat model page shall present a list of all the data flow diagram (DFD) elements along with their respective threats, as well as the available frameworks that can be applied to each element. The page shall also provide functionality for saving, undoing, and redoing changes made to the threat model, as well as generating reports based on the current state of the model.

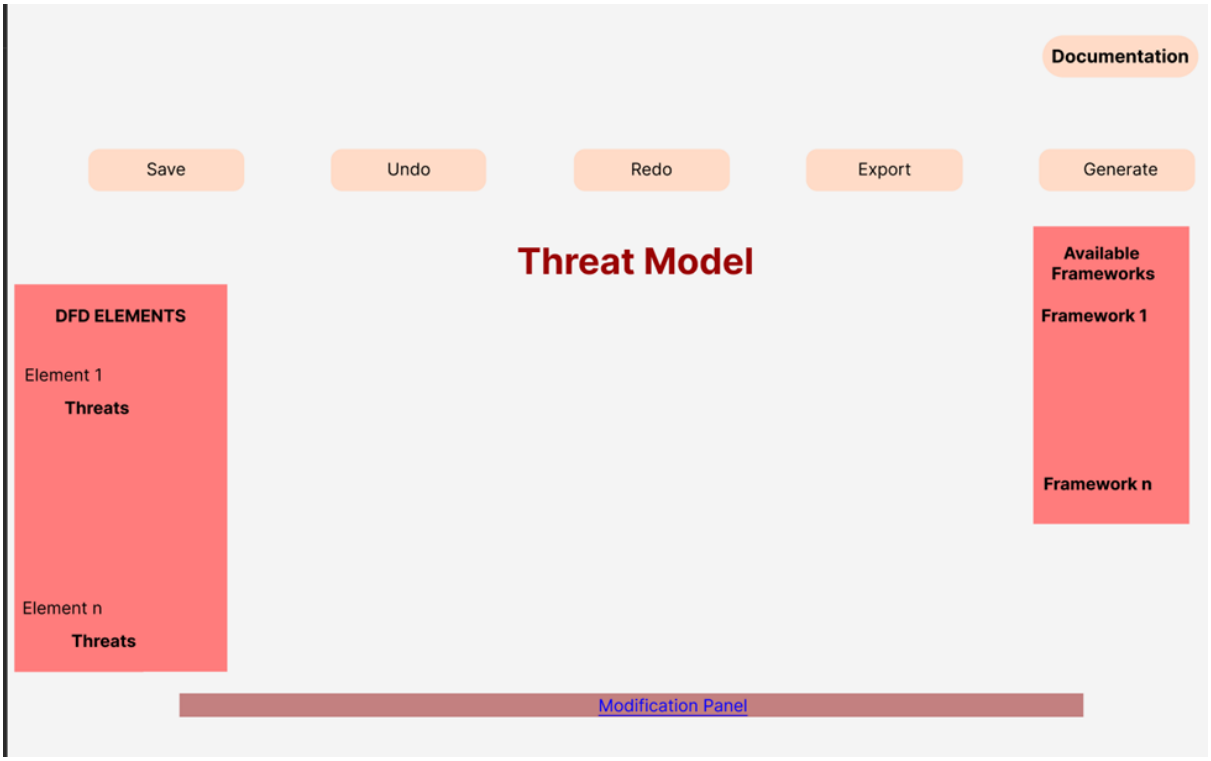

Figure 07: UI design for Threat Model Page of web application

In case of a new template, the Threat Pilot shall have the option to either create a new threat from scratch or modify an existing one. This includes creating new threat categories or deleting existing templates. As a web application, users also have the option to publish and make their custom templates available to others.

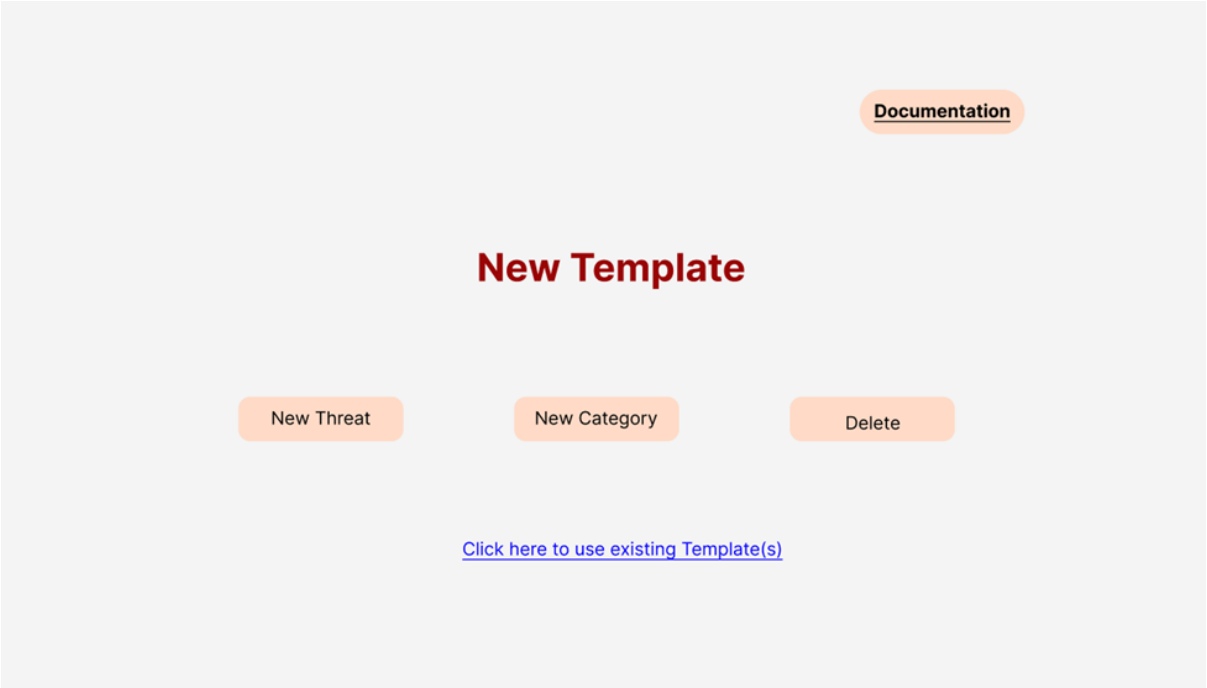

Figure 08: UI design for New Template Page of web application

The Open Template page shall allow users to select from pre-existing templates to speed up the threat modelling process. Users also have the option to modify or create new templates according to their own requirements.

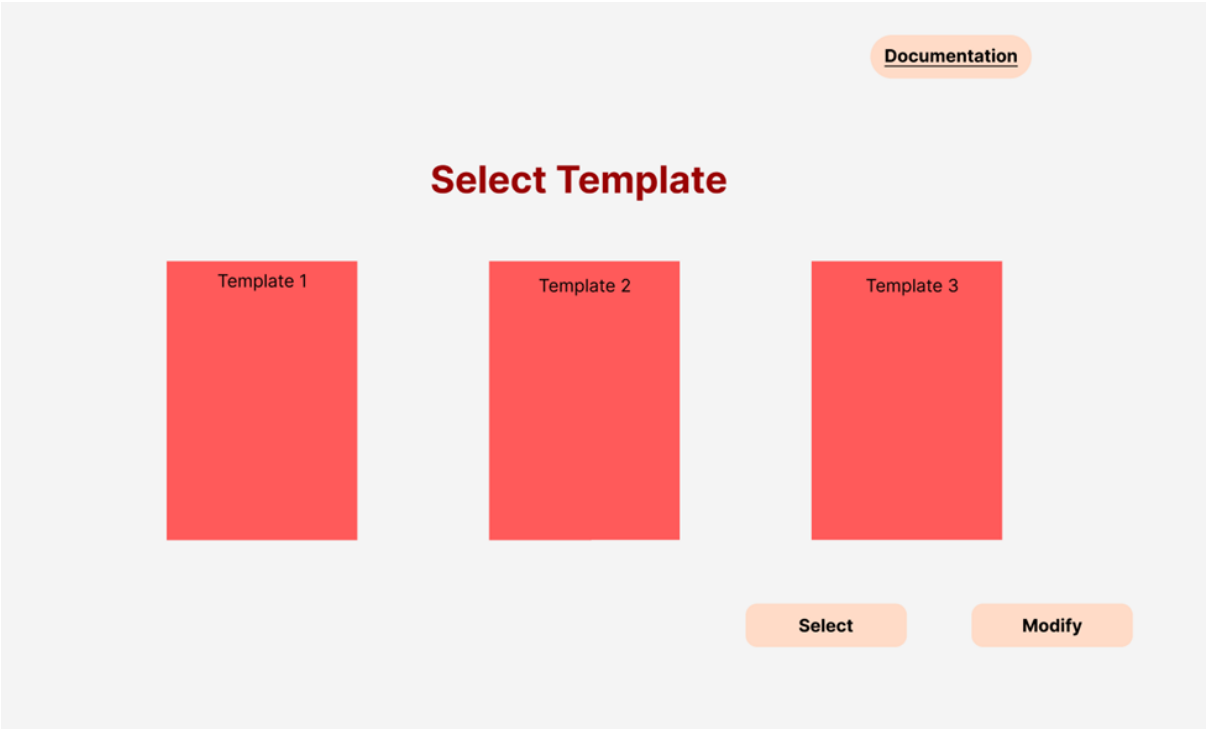

Figure 09: UI design for Select Template Page of web application

This component would handle the creation and management of elements, including the different element properties. In the Threat Pilot web application, there will be multiple elements, and their details will be displayed in the left menu bar. From there, users can link or edit a particular element.

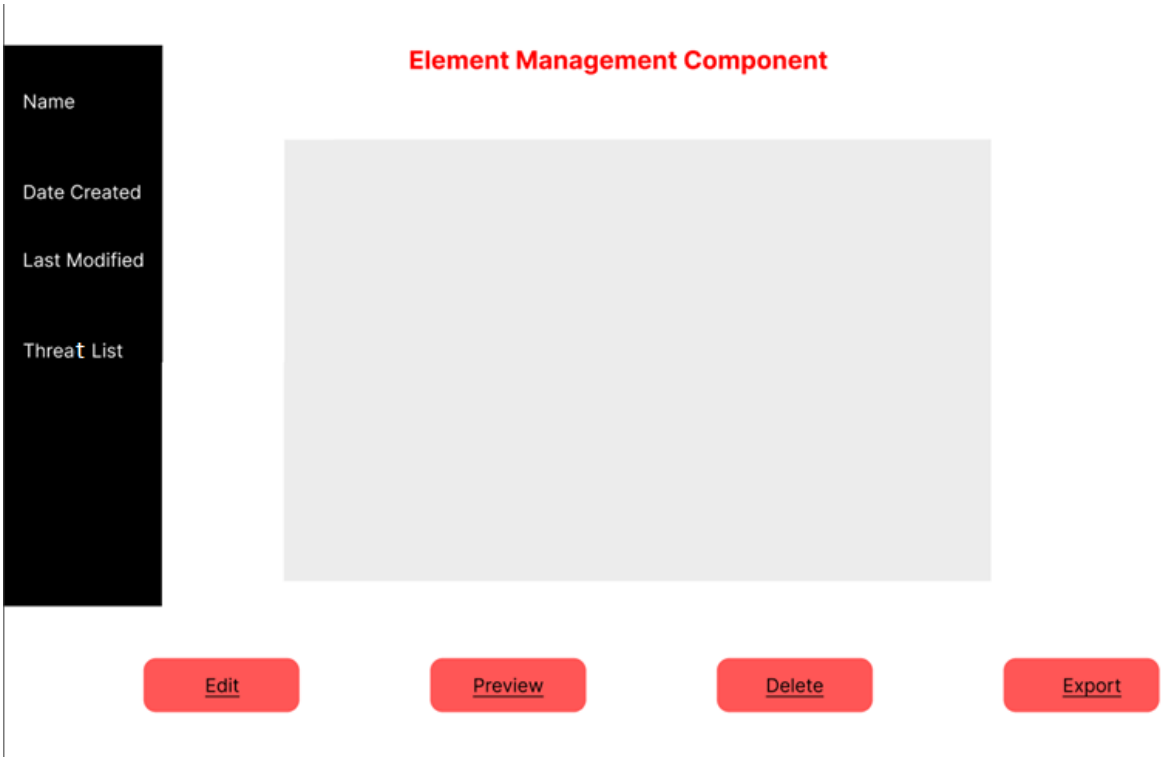

Figure 10: UI design for Element Management Page of web application

The diagram generator in the Threat Pilot comprises a screen for creating diagrams, and the left side contains all the stencils required to make the diagram. On the right side of the screen, there shall be a section that shows details of the element selected. Each element will have specific configurations, and related warnings and errors will also be displayed in the same section. This feature will provide users with better visualization and understanding of the diagrams being created.

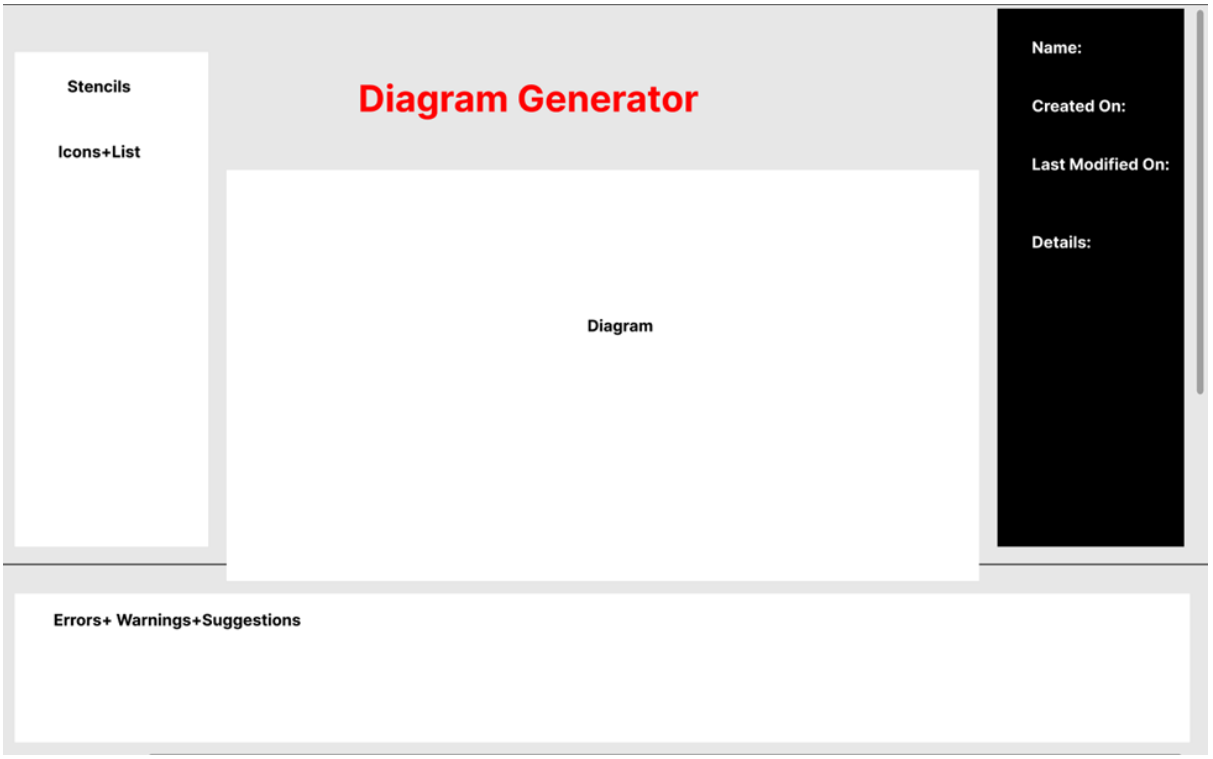

Figure 11: UI design for Diagram Generation Page of web application

The project management feature will provide a list of all the projects available, along with navigation menus. On the right-hand side of the page, the users shall be able view past activities and select a project of their choice, along with the completed project list. This will enable us to manage and monitor the ongoing and completed projects effectively.

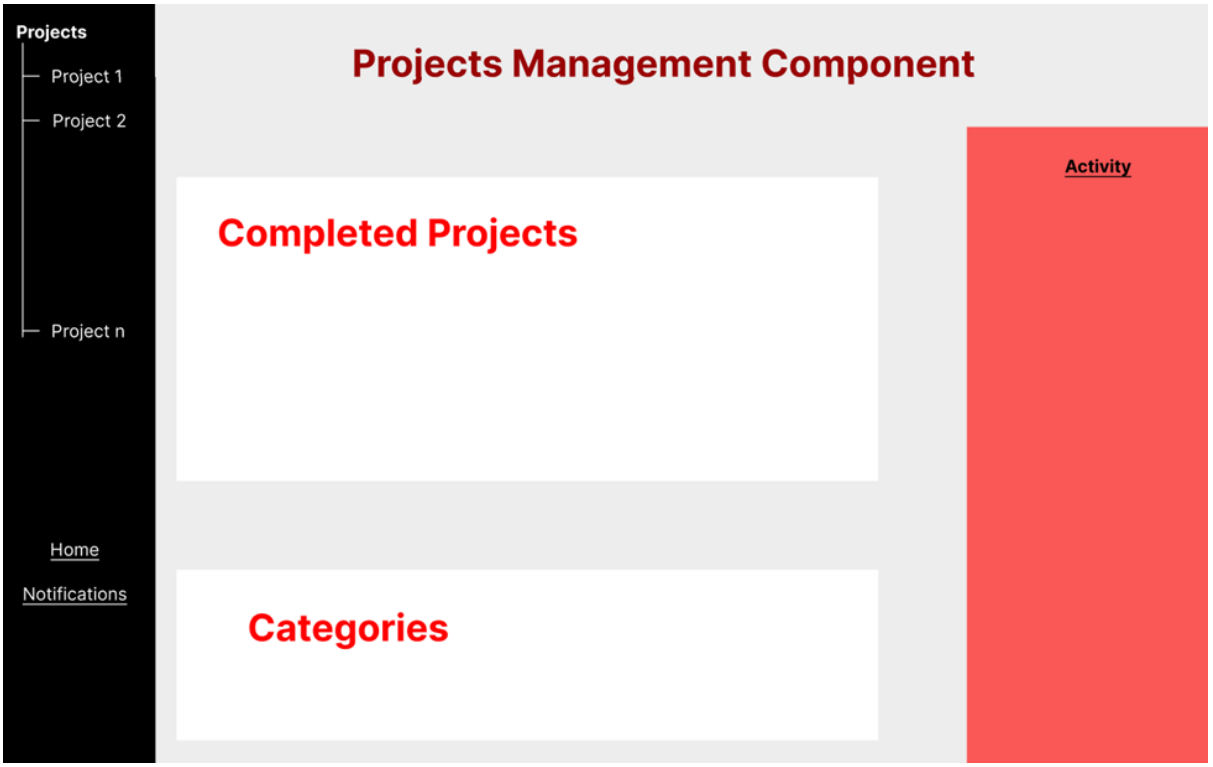

Figure 12: UI design for Project Management Page of web application

5.4 UI Design for the Threat Pilot Desktop-Application

The logic and functionality of the application, as described previously, are generic and applicable to both the web and desktop versions. Therefore, the design documentation for both versions will remain the same. However, there shall be differences in the implementation stage. The UI/UX design will remain consistent between both versions, providing a unified experience for users. Any changes in the implementation stage will be documented accordingly.