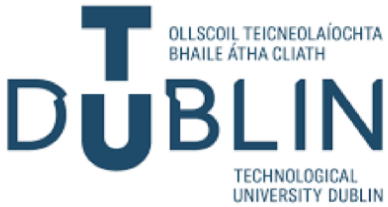


ZakatIQ - The Smart Islamic Finance Assistant
Interim Report

TU858/4
BSc in International Computer Science

Wissam Hadjarab
C21404706

Supervisor: Mariana Rocha

School of Computer Science
Technological University, Dublin

Date
10/11/2025

Abstract

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed: 

Student Name: Wissam Hadjarab

Date: 10/11/2025

Acknowledgements

Body text

Contents

1. Introduction 1
1.1 Project Background 1
1.2 Project Description 1
1.3 Project Aims and Objectives 2
1.4 Project Scope 3
1.5 Thesis Roadmap 4
2. Literature Review 5
2.1 Introduction 5
2.2 Alternative Existing Solutions 5
2.3 Technologies Researched 5
2.4 Other Research 6
2.5 Existing Final Year Projects 7
2.6 Conclusions &References 9
3. System Analysis 10
3.1 System Overview 10
3.2 Requirements Gathering 10
3.3 Functional requirements 11
3.4 Non-functional requirements 18
3.5 Requirements list 22
3.6 Initial System Specification 24
3.7 Conclusions 26
4. System Design 26
4.1 Introduction 26
4.2 Software Methodology 27
4.3 Overview of System 29
4.4 Design System 31
4.5 Other Section 32
4.6 Conclusions 37
5. Testing and Evaluation 39
5.1 Introduction 39
5.2 Plan for Testing 39
5.3 Plan for Evaluation 40
5.4 Conclusions 41
6. System Prototype 42
6.1 Introduction 42

6.2 Prototype Development....43
6.3 Results....53
6.4 Evaluation....57
6.5 Conclusions....60
7. Issues and Future Work....60
7.1 Introduction....60
7.2 Issues and Risks....60
7.3 Plans and Future Work....62
7.3.1 Project Plan with GANTT Chart....63
References....65
A) Appendix A: System Model and Analysis....**Error! Bookmark not defined.**
B) Appendix B: Design....**Error! Bookmark not defined.**
C) Appendix C: Prompts Used with ChatGPT....1
D) Appendix D: Additional Code Samples....**Error! Bookmark not defined.**
E) Appendix E:....1

1. Introduction

1.1 Project Background

Zakat is one of the key pillars for Islam and an obligation upon every Muslim who are eligible to pay under a certain threshold / criterion. All though its importance is very prominent in Islam, many people struggle to understand whether they qualify to pay Zakat, how much and how to calculate it correctly. There are basic calculators online which provide a one-time calculation with no data tracking, intelligence or security and are liable to give incorrect calculations, e.g. inputting uses value in gold, property etc that cannot be inputted into a calculator. They also don't take into consideration a person's lifestyle – whether they are married, have children, expenses etc.
There is currently no application or app / webapp that incorporates AI, Islamic financial rules, personal financial tracking and secure donation options in one platform. There are tools like Mint or YNAB that offer financial tracking but don't align with Islamic ethics as they are interest based. There are some apps like Zakatify (USA Platorm) that focus on donations but not eligibility prediction or financial guidance for Muslims.
Research in financial machine learning supports the use of algorithms like decision trees, linear regression and more for income and savings forecast. These models are already widely used in finance tech and can be adapted to suit different ethical applications. There are different frameworks such as Flask that help with security with encryption libraries to ensure data privacy which is crucial when dealing with sensitive financial information.
This project builds on existing technologies, combining and adapting them to tailor to the Muslim community, offering smarter, secure and a more personalized way to manage their finances to fulfil their religious obligations.

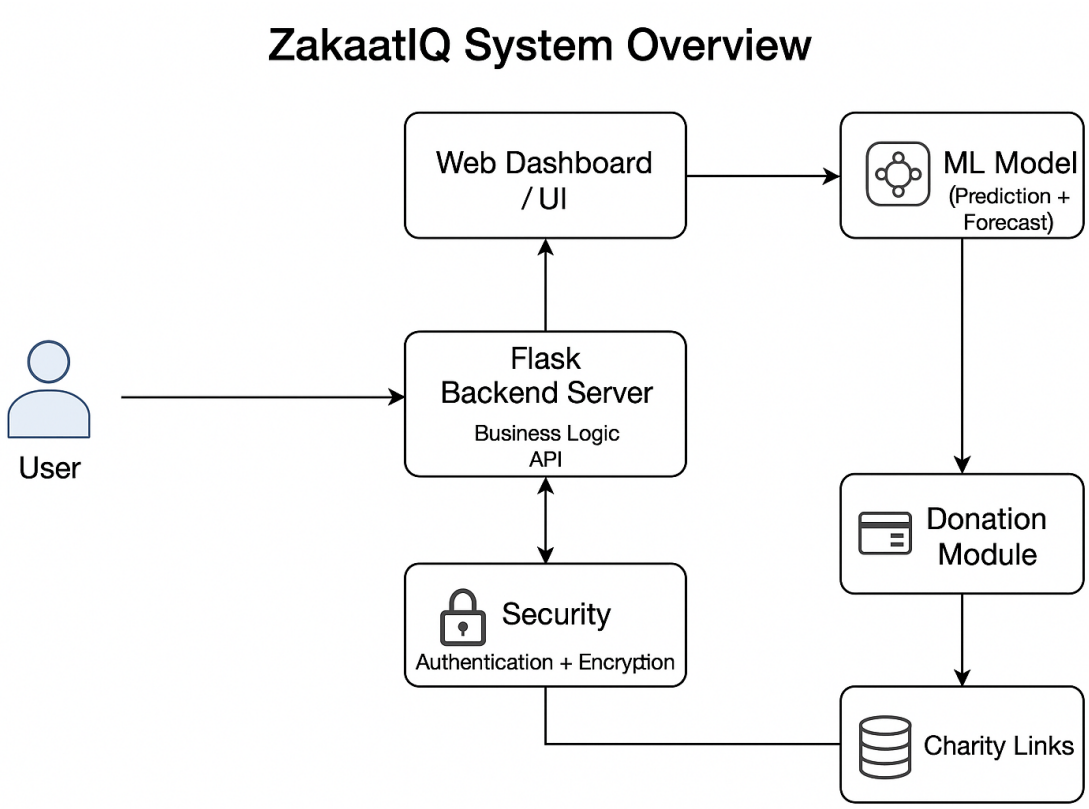
1.2 Project Description

This project *ZakaatIQ*, is an AI-powered Islamic financial assistant designed to help Muslims manage their wealth in a halal (permissible by Islamic law) and ethical way. The app goes beyond traditional Zakat (Charity) calculators by using **machine learning** to predict Zakat eligibility based on a user's financial data (income, savings, debts, gold, etc.) and **personal lifestyle** factors like marital status, number of dependents, and employment.

The system will also use **AI-based forecasting** to estimate users' future income and savings, helping them plan ahead while avoiding interest-based (riba) savings. It will alert users if their financial profile conflicts with Islamic finance principles and offer **halal suggestions** (e.g., avoiding interest accounts, planning charitable giving).

Users will be able to **securely track their net assets**, view personalized dashboards, and access **verified charity donation links** to fulfill their Zakat directly through the app. All sensitive information will be encrypted, and security best practices (e.g., hashed passwords, encrypted storage) will be used throughout.

By combining **AI, ethical finance, secure development, and Islamic values**, ZakaatIQ offers a smart, modern solution for Muslims seeking financial clarity, spiritual accountability, and long-term planning — all in one user-friendly platform.



1.3 Project Aims and Objectives

Project Deliverables:

1) **ZakaatIQ Web Application**
   A fully functional AI-powered web app that predicts Zakat eligibility, tracks the users' finances, forecasts their income/savings and integrate it with secure charity payment links.
   It will include:
   - User authentication (log in) and encrypted financial data storage that protects all personal and financial data
   - Build personalised user dashboard for tracking income, savings, debts and assets
   - ML based Zakat prediction model
   - AI forecasting model for income/savings
   - To allow customisation of Zakat calculations based on user lifestyle, including marital status, dependents, and living expenses with advice
   - Secure dashboard with personalized insights and graphs for user
   - To integrate charity donation functionality with links to verified Islamic charities
   - To evaluate the system's usability, security, and AI prediction accuracy through structured testing and user feedback

2) **Machine Learning Models**
   One classification model (decision tree or logistic regression) – for Zakat eligibility
   One regression model (Linear regression or Prophet) – for income/savings forecasting
   Models will be trained using example data and adapted to Islamic Finance constraints
   Research and learning will be conducted as part of the model development process
3) **Database System**

Encrypted database using mainly PostgreSQL and SQLite to store individual users' financial data securely.
Data will be encrypted and follow GDPR and Islamic ethical data handling principles

4) **Technical Documentation**
   - Full system architecture design
   - Inline code comments and GitHub repository documentation
   - API Design – if applied
   - Security design, including encryption, authentication and privacy measures
5) **Final report**
   This will be a comprehensive report covering all stages of the project:
   - Requirements
   - Design
   - Implementation
   - Testing
   - Evaluation
   - Critical reflection
6) **Final demo and presentation**
   PowerPoint presentation showcasing the project along with a live or recorded demo of the working final product, show casing key features, AI predictions and charity integration

1.4 Project Scope

The scope of this project is to design and develop a web-based application called ZakaatIQ**,** which helps individual Muslims calculate and manage their Zakat obligations in a secure, quicker, and personalised way.

The application will enable users to track their income, savings, assets, and debts, and use this financial information — along with lifestyle details such as marital status and number of dependents etc— to determine their Zakat eligibility. AI models will be used to forecast future savings and income, and a secure dashboard will present this information through graphs and personalised insights. Additionally, users will be able to donate their Zakat to verified Islamic charities through secure external links.

The platform will also focus on ethical financial handling in accordance with Islamic principles, ensuring that all user data is encrypted and protected in compliance with GDPR.

It includes:

- Secure login and user authentication
- Personal financial tracking (income, savings, debts, assets)
- Machine learning models for Zakaat eligibility and financial forecasting
- A personalised dashboard with insights and recommendations
- Lifestyle based Zakaat advice
- Integration with a list of verified charities (external secure links)
- Encrypted database storage using PostgreSQL
- Alignment with Islamic financial ethics and privacy principles

Isn't financial management app – helping people keep track for charity for niche group

1.5 Thesis Roadmap

- **Section 2: Literature Review** – Examines previous research, existing Zakat tools, Islamic finance principles, and the role of AI in financial applications.
- **Section 3: System Analysis** – Defines the project requirements, user needs, system goals, and outlines the initial system architecture.
- **Section 4: System Design** – Details the technical design of the application, including the ERD, database structure, user interface sketches, and AI integration approach.
- **Section 5: Implementation & Testing** – Describes the development process so far, testing strategies for each component, and evaluation of functionality and performance.
- **Section 6: Issues and Future Work** – Discusses current challenges, limitations of the prototype, and outlines the next stages of development.
- **Section 7: References and Appendices** – Provides citations for all research used and includes additional resources like diagrams, survey results, and design artifacts.

2. Literature Review

2.1 Introduction

This chapter provides a review of the existing literature, technologies, and research relevant to the ZakaatIQ project. It explores the religious and financial context of Zakat in Islam, the limitations of current online Zakat calculators, and the growing role of artificial intelligence in financial forecasting. The chapter also highlights how Islamic values influence financial decision-making and why there is a need for technology that aligns with those principles. By analysing existing tools, similar apps, and relevant research, this chapter identifies gaps in the market and shows how ZakaatIQ offers a more personalised, intelligent, and ethical solution for managing Zakat obligations.

2.2 Alternative Existing Solutions

There are several online tools and platforms that aim to assist Muslims with Zakaat calculation or with Islamic finance technology services:

Islamic Relief Zakaat Calculator:

- Offers a one-time, manual Zakat calculation
- Users input assets like cash, gold, property, liabilities
- Doesn't track user data or store progress
- No user accounts, forecasting, or security measures

National Zakaat Foundation (NZF):

- Similar input-based calculator (cash, investments, liabilities)
- Offers Nisab (minimum amount of wealth required for a Muslim to be liable to pay Zakaat) threshold checker
- Focused on UK-based Zakat calculation and donation
- No AI or personalised financial advice

Zakatify (US):

- App focused on **donating** Zakat, not calculating eligibility
- Allows users to choose verified charities
- No prediction, no financial tracking, no Islamic finance integration

Zoya Finance:

- Portfolio screening app for halal stocks and investment
- Includes a simple Zakat calculator for investment assets
- Focused on halal investing, not full Zakat lifestyle guidance

You need a budget (YNAB):

- Budgeting and financial tracking tools
- Feature rich but interest-based and not Shariah-compliant
- Don’t cater to Zakat or Islamic financial obligations

2.3 Technologies Researched

Hardware:

- MacBook Air M2 (macOS)
  This will be my main machine that I will use to develop the different features, suitable for coding, testing and running local servers.

Has internet access for installing packages, deploying web app and using cloud services

- Second device
  I will use a secondary device like smartphone for testing responsive design and UI cross different screen sizes and OS

Software development tools:

- Python: This language is versatile which will be critical for my deployment of machine learning models, web development and scripting. Core language for backend logic and financial data processing
- Flask: Lightweight web frameworks for Python that gives full control over backend routes, session-based management (login/ register) and templates. Used to create the UI for login financial input, predictions, dashboards and charity links
- VSC: Lightweight code editor that supports all languages – including Python and Flak extensions. Main environment for writing and testing code
- Scikitlearn: Trusted, open-source Python library for machine learning. Supports classification (decision trees, logistic regression) and regression tasks which will be used for Zakat eligibility prediction and forecasting. Will be used to deploy ML models based on users' finances and lifestyle data
- Prophet / statsmodels: Prophet – tool used for time series forecasting income and savings trends. Needed to predict how much users are likely to save or earn in the future based on patterns
- Pandas, Numpy + Matplotlib: Core python libraries for data analysis and visualization with basic graphs. Used for data input processing, ML engineering and visualizations in the dashboard.
- Cryptography (Fernet): Enables secure encryption and decryption of sensitive financial data. Ensures all user financial data and login credentials are stored securely in line with GDPR and ethical principles
- PostgreSQL – Scalable relational database to store encrypted user data, including income, savings and Zakat calculations

While existing tools like Zakatify or NZF focus on either donation or basic calculation, none offer a full Zakat management ecosystem. ZakaatIQ fills this gap by combining personal finance tracking, AI-powered Zakat prediction, lifestyle-based advice, and secure donation links — all in one web app aligned with Islamic values. It learns from a user's financial history, forecasts future obligations, and tailors' guidance based on factors like marital status or dependents. With encrypted data storage and Shariah-compliant (in accordance with Islamic law) features, it promotes ethical, faith-aligned financial decisions.

2.4 Other Research

As my project is all about building a smart Zakat platform for Muslims, I wanted to understand how both Islamic finance and modern financial tech work together. A lot of research shows that while tools like AI are already being used in general finance apps, there's still a big gap when it comes to using them in a way that's fully aligned with Islamic principles.

For example, Zafar and Ali (2025) talk about how AI is growing fast in the finance industry, but Islamic financial institutions still don't have strong Shariah-compliant guidelines for using it. This creates an issue as when dealing with something as important as Zakat, users need to know that the tool is not just smart — but also religiously ethical.

Laylo (2023) goes into how digital platforms can help make Zakat easier to manage, especially with automation, but they also mention that the tech needs to be built carefully to respect values like trust and fairness. Iqbal et al. (2025) also looked at AI in Islamic finance and found it has potential, but warned that without strong ethical design, there's a risk of data misuse or unfair predictions. These points helped shape the approach needed for the features and data protection side of ZakaatIQ.

I also looked at financial apps like Mint, YNAB and Cleo. These use AI to help users track spending, set goals, and predict savings — and a Deloitte (2023) report said that AI boosts user engagement in budgeting tools. But the issue is, they're all based on conventional finance, and include things like interest calculations or debt features that aren't in compliance with Islamic rules.

Another useful study by Mahomed and Ismail (2020) focused on digital Zakat platforms in Malaysia. They found that younger users are open to using AI-based tools for Zakat, but they also worry about things like privacy and how their data is handled. That's why I've made sure ZakaatIQ will use encryption, secure login, and proper data management.

Overall, this research gave me the confidence that there's a real need for a platform like ZakaatIQ — one that combines modern tech with Islamic ethics to make Zakat management easier, more accurate, and more meaningful for Muslims.

2.5 Existing Final Year Projects

1. Title: Using Machine Learning Classification Methods to Detect the Presence of Heart Disease

Description (brief): The project applies classification models to medical data to detect whether a patient has heart disease based on features like cholesterol, blood pressure etc. It compares different algorithms (decision trees etc) to identify the best predictor

What is complex in this project:

- Medical data often contain missing values, noise and varying scales – so preprocessing is complex
- Choosing and tuning multiple classification models
- Dealing with class imbalance (patients with disease vs healthy ones)
- Interpreting model outputs in a domain where false negatives/false positives have huge implications

What technical architecture was used:
- A data processing and ML pipeline (preprocess -> train/test -> validate)
- Comparison study of algorithms (decision tree, SVM)
- Likely Python / scikit-learn ecosystem

Explain key strengths and weaknesses of this project, as you see it.
Strengths:
- Direct comparison of algorithms gives insight into which works best for the problem
- Uses realistic medical, enhancing relevance
- Clear domain and well-understood evaluation metrics

Weakness:
- Not a full software solution – no UI or web application features
- Doesn't consider privacy, encryption or domain-specific rules (health care data regulation)
- May not include lifestyle inputs (which your project plans to)
- Its insights are limited to medical predictions, not financial behaviour or ethics

2. Title: Using Machine Learning Techniques to Predict a Risk Score of New Members of a Chit Fund Group

Student: Sinead Aherne

Description (brief): This project is a supervised learning to predict a risk score for a new or existing member of a chit fund (collective savings/credit group). It aims to identify which members are likely to default, based on historical data

What is complex in this project:
- Handles imbalanced classification
- Uses multiple machine learning algorithms (SVM, decision trees)
- Feature engineering and factor reduction
- Incorporating social network analysis (links between members) as part of the risk score model

What technical architecture was used:
- Data driven ML pipeline
- Comparison of multiple
- Use of real historical data from a chit fund
- Python based ML stack

Explain key strengths and weaknesses of this project, as you see it.
Key strengths:
- Testing multiple models to find the best fit
- Real world data usage gives practical relevance
- Including social network / relational analysis
- Clear objective and domain

Weakness:
- Doesn't include a user facing application or interface
- No features related to data security, privacy, encryption or user profiling
- Doesn't account for domain-specific constraints

- The evaluation metrics may be impacted by data imbalance

2.6 Conclusions & References

The literature review highlights that while multiple Zakat calculators and financial tracking tools exist, they typically offer only one component—such as a simple calculation or donation link—without combining prediction, tracking, personalisation, and ethics. Research in Islamic fintech underscores both the opportunities and critical ethical challenges in applying AI and data analytics in a Shariah-compliant context (Iqbal et al., 2025; Zafar & Ali, 2025). This gap in the market signals a clear opportunity for a comprehensive solution. The proposed ZakaatIQ platform seeks to fill this gap by offering a holistic system that merges finance tracking, lifestyle-aware analytics, AI-based forecasting, secure data handling and verified charity integration—all firmly within Islamic financial principles. In doing so, it contributes both technically and ethically to the evolving landscape of Islamic fintech.

3. System Analysis

3.1 System Overview

Initial description from a user perspective:

- ZakaatIQ is designed to be a smart and secure Zakat assistant for Muslims. From the user's point of view, the system is very straightforward: they log in, enter their financial details like salary, savings, debts, gold they own, or any other assets and the app tells them if they're eligible to pay Zakat and how much they owe.
- It also gives them helpful insights into their future income and savings using predictions, which can help them plan ahead. Based on their personal situation — like if they're married or have children — the app will provide tailored Zakat advice to the given user. If eligible, users can even donate directly through links to verified Islamic charities. Everything is displayed in a clean dashboard, and users can come back any time to check or update their info.

Non-Technical description of system:

- ZakaatIQ is a web-based application built to make Zakat management easier, more secure, and more personalised. It allows users to create accounts, log in securely, and input key financial information. This information is processed using AI models — including a classification model (e.g., decision tree or logistic regression) to check Zakat eligibility, and a regression model (e.g., linear regression or Prophet) to forecast income and savings.
- The platform then displays results through a user-friendly dashboard that includes summaries, charts, and links to trusted charities. All sensitive data is encrypted to protect user privacy, and the system follows Islamic finance principles by avoiding interest-based calculations and ensuring compliance with ethical standards.

3.2 Requirements Gathering

<u>Key Stakeholders</u>

In my development of ZakatIQ, it's important to consider the needs and expectation of different stakeholders who may interact with or benefit from the platform:

1. Muslim individuals – The primary users of the app who want to calculate and manage their Zakaat obligation accurately.
2. Muslim families – Users with dependents or different financial responsibilities who need lifestyle based financial advice.
3. Self-employed Muslims/Business owners – Who require more tailored financial forecasting and Zakaat calculation for irregular income.
4. Islamic charities – End of recipients of Zakaat donations. They benefit from secure, direct donation integration.
5. Islamic finance advisors or scholars – To ensure the platform is built in line with Shariah law
6. Developers / system maintainers (internal) – Responsible for the secure, ethical implementation and upkeep of the system

<u>Gathering requirements:</u>

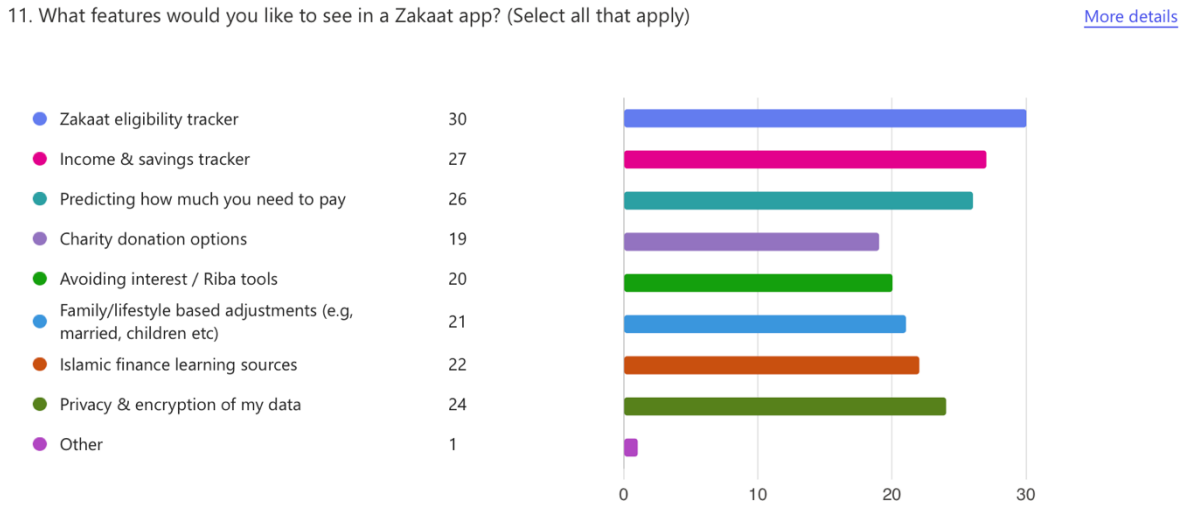
To gather initial requirements, I created a survey and distributed amongst the Muslim community in mosques, universities and families using Microsoft teams. The aim was to identify current challenges in Zakaat management, gauge interest in the features listed and understand what tools users currently use or trust and what features they would want in the app. The survey mainly targeted Muslims living in Ireland and UK. The survey aimed to understand:

- Current Zakaat practises
- Struggles with calculating Zakaat
- Desired features in Zakaat app
- Trust in financial technology
- Interest in using or testing a tool like ZakaatIQ

The survey received 30+ responses and participants ranged from single individuals to married respondents, with or without dependents.

Initial Requirements (Based on survey):
From the results, 100% of the users said they would use an app to track Zakaat eligibility. The majority however did feel uncertain about how they calculate their own Zakaat with confidence rating being as low as 1/5. Most users currently do not know how to calculate Zakaat or rely on simple calculators online
The biggest difficulties were:

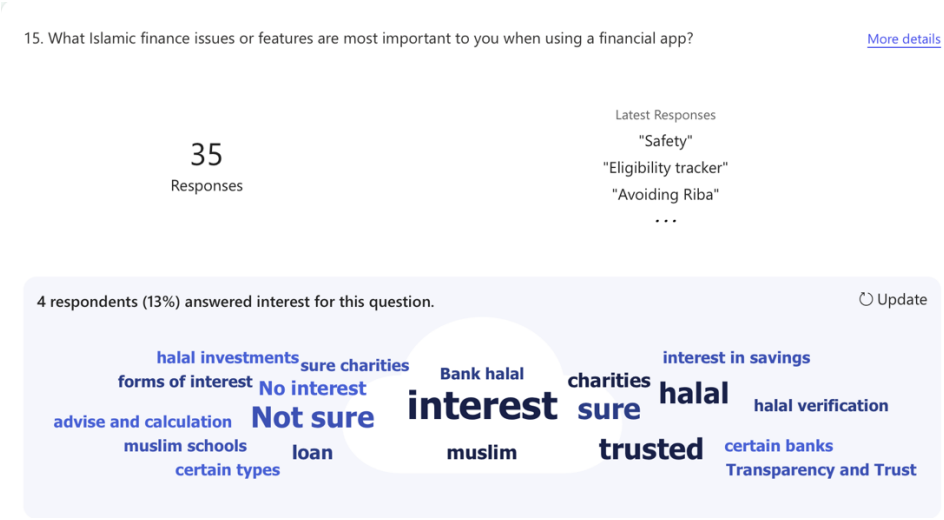
- Not knowing if they're eligible
- Tracking savings/income over time
- Unclear definitions of what counts as Zakatable wealth
- Confusing calculators



Most wanted features included:

- Zakat eligibility tracker
- AI prediction of income/savings
- Lifestyle-based advice
- Charity donation links
- Tools for avoiding Riba/interest

Users expressed mixed comfort levels with sharing financial data but said it mostly depended on the security features and transparency of the app. Most users said once they are comfortable with this along with the feature of being able to directly donate through the app, they would like to give feedback as testers.



3.3 Functional requirements

Functional use cases represent the core interaction between the user and the ZakaatIQ system. These describe how users perform essential tasks, such as checking Zakaat eligibility, forecasting income & savings and making secure donations. Each use case outlines the steps involved, system behaviours, actors and exceptions to ensure that user goals are met effectively.

USE CASE	1	Zakaat Eligibility Prediction
Goal in Context **Brief Description**	User uses the ZakaatIQ system to check if they are required to pay Zakat, based on financial input. This typically takes place at home or on a mobile device.	
Preconditions	Users must be logged in	
Success End Condition **Post-condition**	Zakaat obligation status is displayed	
Failed End Condition **Post-condition**	System cannot calculate due to missing/incomplete financial information	
Primary, Secondary, Actors	Muslim user – any individual seeking to assess their Zakaat obligation based on their personal finances Other systems relied upon – ML Model; to determine Zakaat eligibility, PostgreSQL; to retrieve stored financial data, Encryption module; to securely handle sensitive inputs, ZakaatIQ web application; main interface for data entry and results display	
Trigger	User clicks on “Zakat Eligibility”	
DESCRIPTION	The user visits the ZakaatIQ web app and logs in using secure credentials .Once authenticated, the user is taken to a personalised dashboard. They select the “Zakat Eligibility” feature from the available options. The system presents a form asking for financial details: Monthly/yearly income: • Current savings • Gold/silver or other valuable assets • Debts or liabilities	

	• Lifestyle info (e.g., marital status, number of dependents) • The user fills out the form and submits the data. • The system: • Validates that the inputs are complete and correctly formatted. • Encrypts and securely stores the data in a database. • Passes the data to a **machine learning classification model** (e.g., decision tree). The model analyses the user's financial situation against the Nisab threshold**.** A result is generated: **e**ither **"**Zakat is Required" or **"**Zakat is Not Required"**.** The system displays the result clearly with an explanation. The user is given options to: - Save the result to their profile - Proceed to the donation section (if eligible) - Edit their data (if they made a mistake) If the input was incomplete or invalid, an error is triggered, and the user is guided to correct it.

Main Flow

Step	Action	Alternate
1.1	User logs into the ZakaatIQ system (System request login credentials)	
1.2	The system authenticates the user	
1.3	Users select "Zakat Eligibility" from the dashboard	
1.4	System presents form to input financial and lifestyle information	
1.5	User submits financial data	E.F 1.5
1.6	System validates data is complete and formatted correctly	E.F 1.6
1.7	Valid data is encrypted and securely stored in the database	
1.8	System passes data to machine learning model	E.F 1.8
1.9	Model predicts if user meets Nisab threshold	
1.10	System displays result (Zakaat required / not required) with reasoning	A.F 1.10
1.11	User chooses to save results or proceed to charity donations	
1.12	If saved, result is stored in user's dashboard history	
1.13	End use case	

EXCEPTIONS or ERROR Flow Description

E.F. 1.5: Error flow at step 5 of Use Case 1 – Incomplete or Invalid Financial Data

The user submits the form, but required financial details are missing or in an incorrect format

Step	Branching Action	Alternate

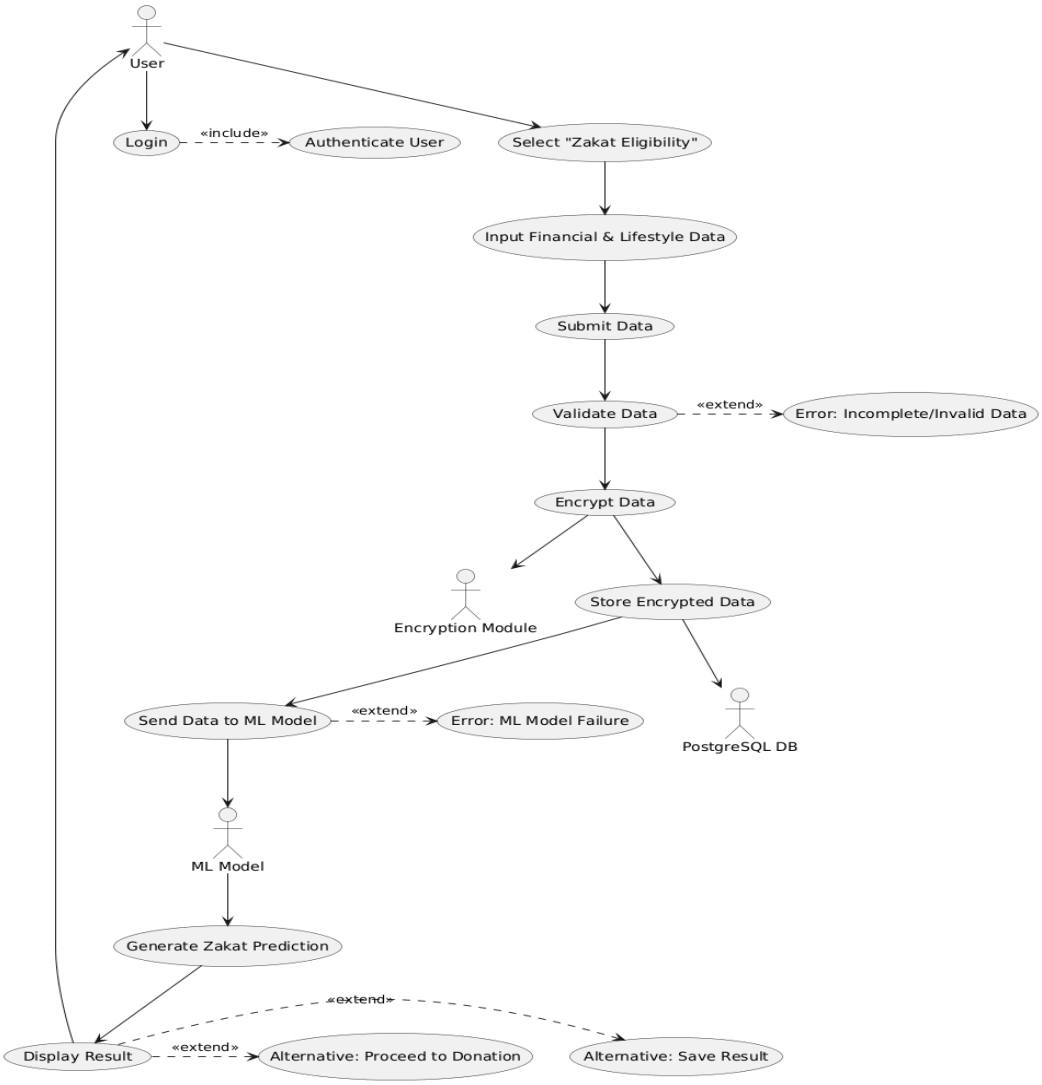
1.5.1	System detects incomplete or invalid fields	
1.5.2	System highlights missing or incorrect fields	
1.5.3	User is prompted to correct the data and resubmit	
1.5.4	Go to Main Flow step 1.5	

EXCEPTIONS or ERROR Flow Description

E.F. 1.8: Error flow at step 8 of Use Case 1 – Machine Learning Model Failure

The ML model fails to return a prediction due to technical error or corrupted input.

Step	**Branching Action**	**Alternate**
1.8.1	System indicates to student borrower their account is frozen	
1.8.2	System logs the error and notifies the user ("Prediction failed. Please try again.")	
1.8.3	User is given the option to retry or save draft inputs	
1.8.4	Return to Main Flow step 1.5 or exit use case	



USE CASE	2	Income & Savings Forecasting
Goal in Context **Brief Description**	User wants to forecast future income and savings trends using AI – powered analytics on the ZakaatIQ platform. This typically occurs	

	after users have input financial data and want to plan future Zakaat obligations more effectively
Preconditions	Users must be logged in User must have at least one set of valid financial data submitted
Success End Condition Post-condition	A visual forecast of income and savings is displayed on the dashboard using predictive AI. The result is stored and available for comparison over time.
Failed End Condition Post-condition	Prediction fails due to missing data or AI model errors No Forecast is generated
Primary, Secondary, Actors	Primary Actor: Logged-in Muslim user managing their personal finances Secondary Actor: AI forecasting model (Prophet or regression), PostgreSQL Database, Flask backend, Encryption module
Trigger	User selects the "Forecast Income & Savings" option on dashboard
DESCRIPTION	The user logs their ZakaatIQ account and navigates to the dashboard. From the available tools, they select **"**Forecast Income & Savings." The system checks whether recent financial data exists (e.g., past 6 months of income/savings). If the data is missing, the user is prompted to upload or input historical data. If the data is available, the system - Validates its correctness and that it is completed - Encrypt it for secure processing - Passes it to the AI forecasting model The model analyses patterns, seasons and trends to predict 1) the estimated monthly income over the next 12 months and 2) projected savings trajectory. Results are displayed on a graph with summary insights where the user can save the forecast to their account, set reminders based on future Zakaat obligation and re-run the forecast with updates assumptions.

Main Flow

Step	Action	Alternate
2.1	User logs into the ZakaatIQ system	
2.2	The system authenticates the user	
2.3	User selects "Forecast Income & Savings"	
2.4	System checks for available financial history	E.F 2.4
2.5	If available, system checks the data format	E.F 2.5
2.6	Valid data is encrypted	
2.7	Data passed to forecasting model (Prophet / Regression)	E.R 2.7
2.8	Model generates prediction (monthly income/savings)	
2.9	Results displayed in graphs and summaries	A.F 2.9
2.10	User saves forecast or sets future alerts	
2.11	Forecast stored in user dashboard	
2.12	End of use case	

EXCEPTIONS or ERROR Flow Description

E.F. 2.4: Error flow at step 2 of Use Case 2 – Missing financial data

The forecasting process cannot begin because the user does not have enough historical financial data saved in the system

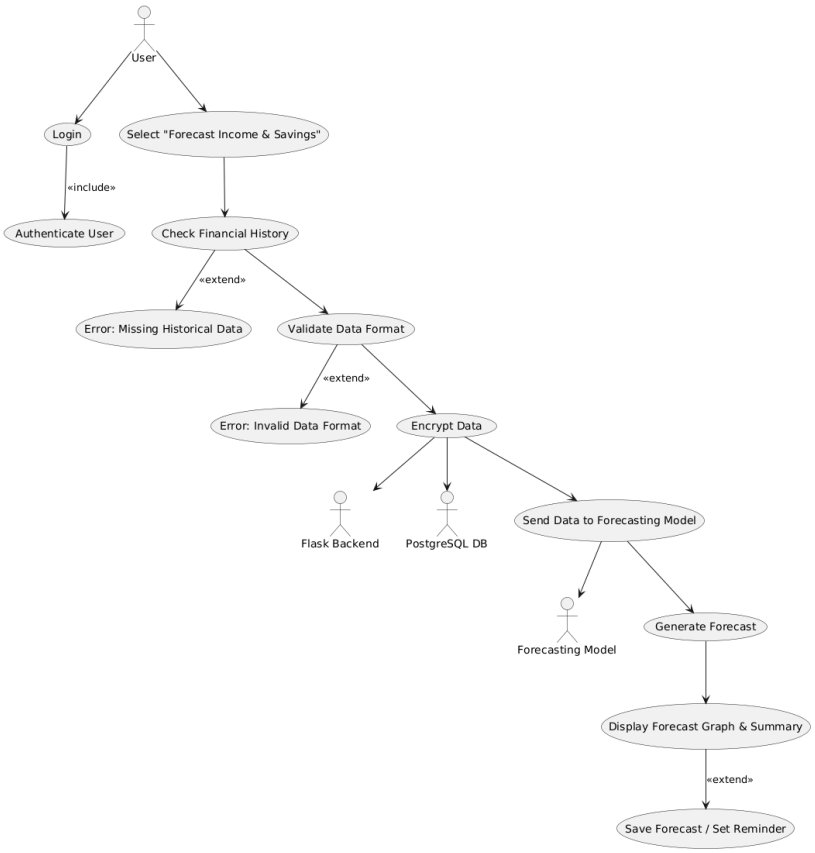
Step	Branching Action	Alternate
2.4.1	System informs user that historical data is insufficient	
2.4.2	User is prompted to upload or manually input past income/savings	
2.4.3	Go to Main Flow step 2.5	

EXCEPTIONS or ERROR Flow Description

E.F. 2.5 Error flow at step 5 of Use Case 2 – Invalid data format

The system stops processing because the financial data contains errors or values in an incorrect format that cannot be used for forecasting.

Step	Branching Action	Alternate
2.5.1	System flags improperly formatted or corrupt data	
2.5.2	User is informed and shown formatting guidelines	
2.5.3	User updates/corrects input	
2.4.4	Return to Main Flow step 2.6	



USE CASE	3	Charity Donation via ZakaatIQ
Goal in Context **Brief Description**	The user donates their Zakat through the ZakaatIQ platform after confirming their eligibility. The system provides a list of verified Islamic charities, calculates how much to donate, and ensures secure, trackable transactions.	
Preconditions	Users must be logged in User must have completed Zakaat eligibility assessment	
Success End Condition **Post-condition**	Zakaat donation is successfully processed and logged Confirmation and receipt are saved in user's dashboard	

Failed End Condition Post-condition	Donation process is interrupted (e.g., user cancels, payment fails) No payment is made, and system stores donation attempt as draft
Primary, Secondary, Actors	Primary Actor: Logged-in Muslim user who is ready to donate Secondary Actor: ZakaatIQ platform, secure payment gateway (PayPal/Stripe), verified charity API (if integrated), PostgreSQL database
Trigger	User selects the "Donate Zakat" option after being confirmed eligible
DESCRIPTION	After completing the Zakaat eligibility prediction, the user is redirected to the "Charity Donation" section where they are shown: - Their calculates Zakaat amount - A list of verified Islamic charities (based on location or cause) - The user selects a charity, enters donation amount (full or part), and clicks "Donate" - The system initiates a secure transaction using an integrated payment processor. If successful, a receipt and reference are shown. The donation and receipt are saved in the user's dashboard for records. If the user does not wish to donate immediately, they can save it as a draft and return later

Main Flow

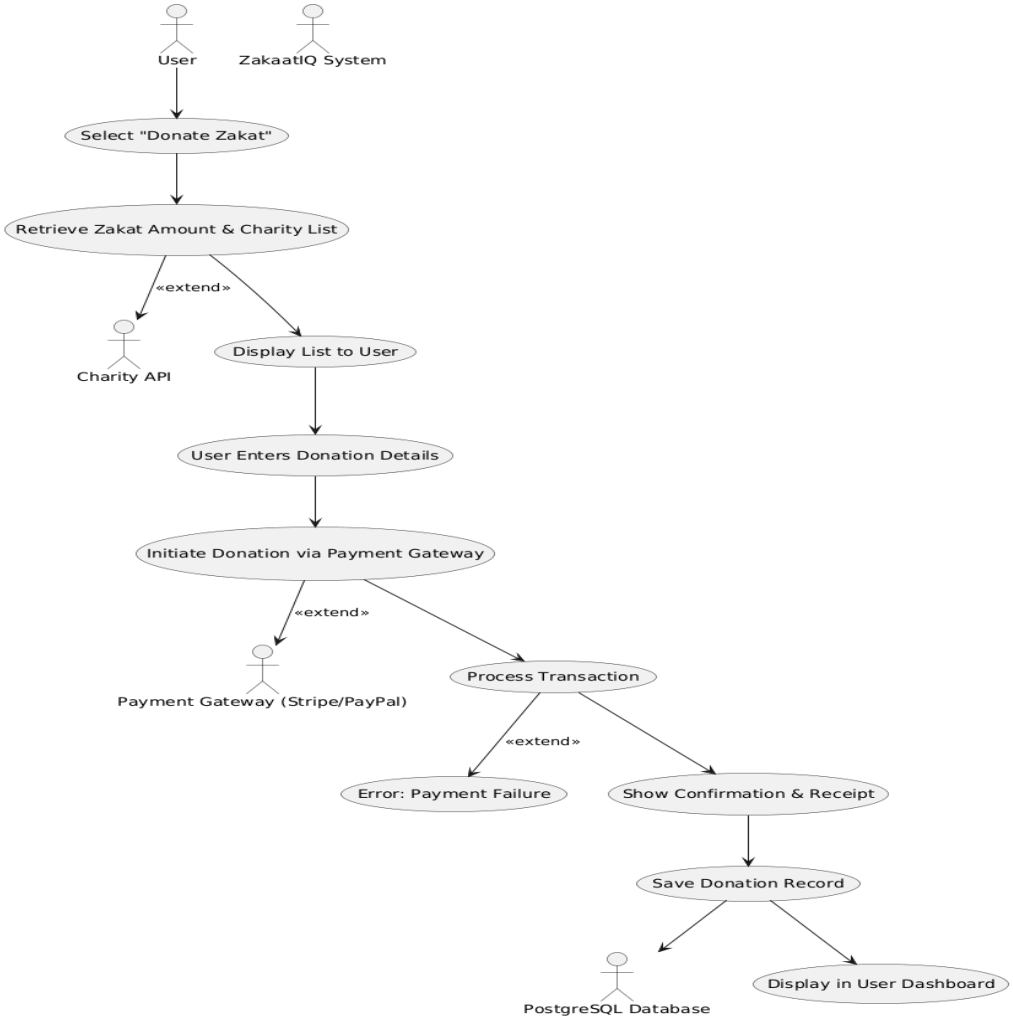
Step	Action	Alternate
3.1	User selects "Donate Zakat" from dashboard	
3.2	System retrieves calculated Zakat amount and charity list	E.R 3.2
3.3	User selects charity and enters donation amount	
3.4	User initiates donation via secure payment gateway	E.F 3.4
3.5	System processes the transaction	E.F 3.5
3.6	Confirmation and receipt displayed to user	A.F 3.6
3.7	Donation record is saved in user dashboard	
3.8	End of use case	

EXCEPTIONS or ERROR Flow Description
E.F. 3.2: Error flow at step 2 of Use Case 3 – Charity list retrieval fails
The system cannot load the list of charities due to server / API error

Step	Branching Action	Alternate
3.2.1	System displays error message ("Charity list unavailable")	
3.2.2	User given option to retry or donate later	
3.2.3	Return to main flow step 3.1 or exit	

EXCEPTIONS or ERROR Flow Description
E.F. 3.4 Error flow at step 4 of Use Case 3 – Payment initiation failure
The payment gateway fails to start due to configuration or internet issues

Step	Branching Action	Alternate
3.4.1	System shows "Payment could not be initiated"	
3.4.2	User is offered to try again or change payment method	
3.4.3	Return to Main Flow step 3.4 or exit	



3.4 Non-functional requirements

Non-functional use cases focus on the system's performance, reliability, security, and usability. Rather than specific features, they describe how the system maintains data protection, ensures consistent uptime, and provides a user-friendly experience. These scenarios ensure the system adheres to ethical, technical, and legal standards—especially important in a finance-related, faith-based context like ZakaatIQ.

USE CASE	4	System security & encryption
Goal in Context **Brief Description**	Ensure that any financial or personal data entered by a user is encrypted before storage or processing	
Preconditions	Users must be logged in and submitting data	
Success End Condition **Post-condition**	Data is securely stored using encryption (e.g., Fernet) in PostgreSQL	
Failed End Condition **Post-condition**	System flags encryption failure, no data is saved	
Primary, Secondary, Actors	Primary Actor: Logged-in Muslim user Secondary Actor: Encryption module, PostgreSQL	
Trigger	User submits financial form or forecast request	

DESCRIPTION	The user is promoted to fill out financial form from the system (Zakaat eligibility or forecast) and fills it out. The system then: - Validates inputs - Passes data to encryption function - Stores in database - Saved to users profile once encrypted If encryption fails, user is notified and asked to resubmit.

Main Flow

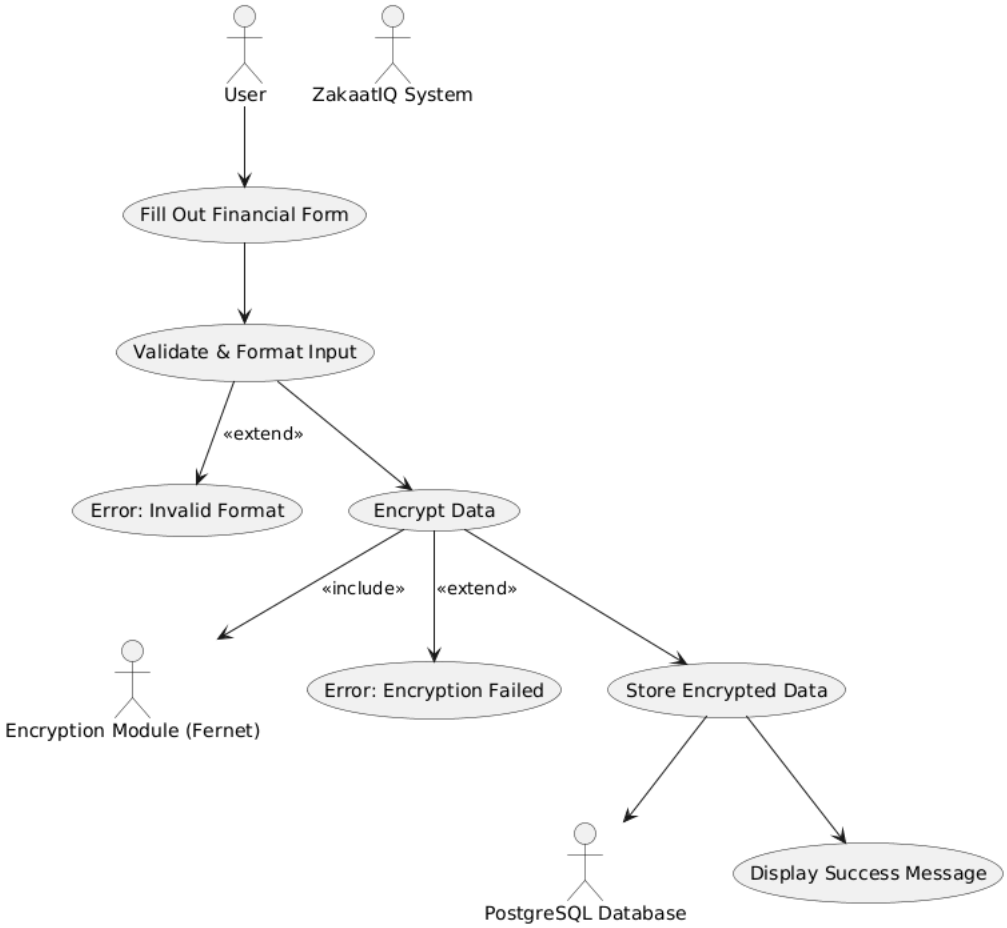
Step	Action	Alternate
4.1	User fills out financial form	
4.2	System validates and formats input	E.F 4.2
4.3	System encrypts data using Fernet	E.F 4.3
4.4	Encrypted data stored in PostgreSQL	
4.5	Success message displayed	
4.6	End use case	

EXCEPTIONS or ERROR Flow Description
E.F. 4.2: Error flow at step 2 of Use Case 4– Invalid input format
User inputs unsupported characters or leaves fields blank

Step	Branching Action	Alternate
4.2.1	System detects formatting issues	
4.2.2	Prompts user to fix	
42.3	Return to main flow step 4.1	

EXCEPTIONS or ERROR Flow Description
E.F. 4.3 Error flow at step 3 of Use Case 4 – Encryption failure
The payment gateway fails to start due to configuration or internet issues

Step	Branching Action	Alternate
4.3.1	System fails to encrypt data	
4.3.2	User is alerted with “Data not saved. Try again”	
4.3.3	Return to 4.1 or exit	



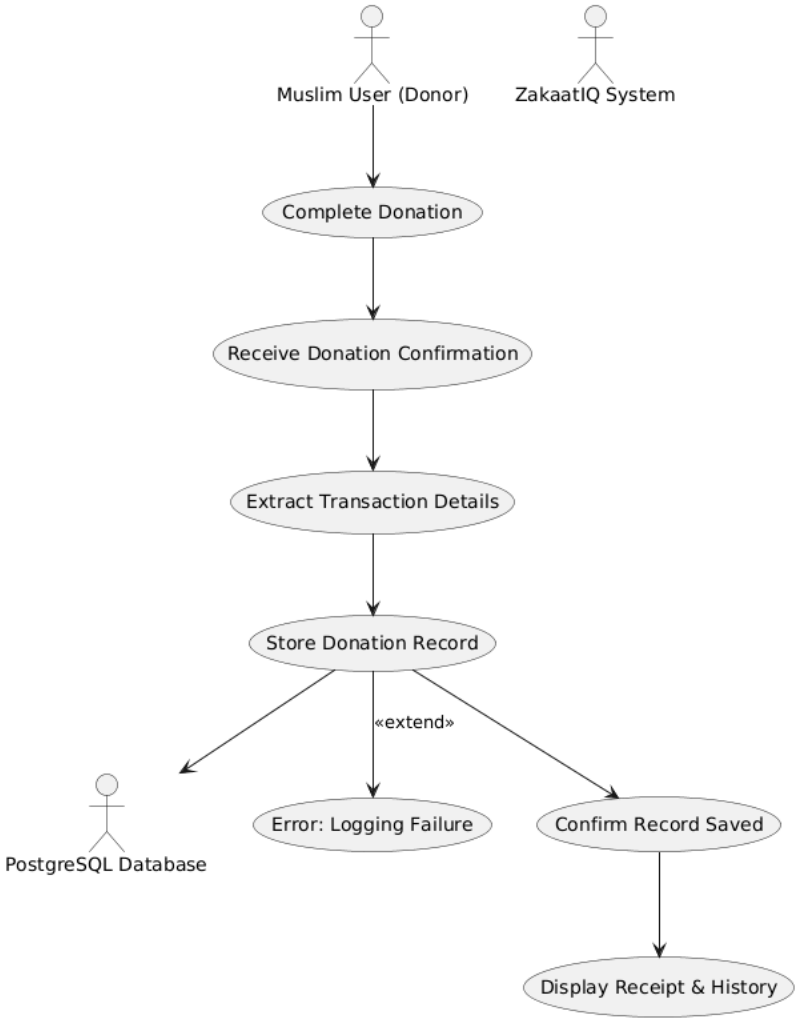
USE CASE	5	Donation Audit logging
Goal in Context **Brief Description**	Every successful Zakat donation must be timestamped and stored securely for traceability and auditing	
Preconditions	Users must be logged in User must have made successful donation	
Success End Condition **Post-condition**	Donation details are stored in database and shown in user dashboard	
Failed End Condition **Post-condition**	If saving fails, donation still goes through, but system notified user to download receipt manually	
Primary, Secondary, Actors	Primary Actor: Logged-in Muslim user who donated Secondary Actor: Flask backend, PostgreSQL	
Trigger	Successful donation confirmation	
DESCRIPTION	After user completed donation via Stripe, PayPal etc, the system will capture: - Amount - Timestamp - Charity	

	- Payment reference This data is securely stored in user's donation history where the user can access past donations for tax or personal reasons

Main Flow		
Step	**Action**	**Alternate**
5.1	System receives donation confirmation	
5.2	Extracts transaction details	
5.3	System stores donation record in encrypted DB	E.R 5.3
5.4	Confirms record saved	
5.5	Display receipt and history	
5.6	End use case	

EXCEPTIONS or ERROR Flow Description
E.F. 5.3: Error flow at step 3 of Use Case 5 – Logging failure
System cannot save donation data due to server/database error

Step	**Branching Action**	**Alternate**
5.3.1	Alert user: "Donation was successful, but record could not be saved"	
5.3.2	Offer user option to download manual receipt	
5.3.3	Exit use case	



3.5 Requirements list

This section outlines the initial system requirements for ZakaatIQ, divided into Functional Requirements (FRs) and Non-Functional Requirements (NFRs). These were derived from the problem domain, system goals, use cases (examples above) and user research (e.g., survey responses)

Functional Requirements

These are the core features and action that the ZakaatIQ platform must perform. They represent the expected functionality based on user goals and system use cases:

1. User authentication:
    - Users must be able to register and log in securely.
    - All credentials must be encrypted and validated on login
2. Zakaat eligibility prediction:
    - Users must be able to input financial and lifestyle data (e.g., income, savings, debts, marital status etc).
    - The system must process the data through a Machine Learning model to assess whether Zakat is due.
    - The result (Zakaat required or not) must be displayed with a clear explanation.

- Users should be able to save results or proceed to donation.

3. Income & Savings forecast:
    - Users must be able to request AI-generated financial forecasts based on historical data.
    - Forecasts should show estimated income and savings trends over the next 6-12 months.
    - Graphs and summaries must be stored in the user dashboard for future comparison.
4. Zakaat charity donation:
    - Users must be able to view and select from a list of verified Islamic charities.
    - The platform should support secure donation processing via an integrated payment gateway (PayPal or Stripe).
    - A digital receipt and confirmation should be saved in the user's dashboard.
5. Financial data encryption:
    - All sensitive financial inputs (e.g., income, savings) must be encrypted before storage in the PostgreSQL database.
6. Personal dashboard:
    - Users should be able to view and manage historical data, eligibility results, forecasts and donations.
    - Re-running calculations or forecasts based on updated data should be supported.
7. Customised lifestyle inputs:
    - Users must be able to personalise inputs such as marital status, number of dependents etc for more accurate calculations based on their lifestyle.

Non-Functional Requirements

These describe how the system should behave and operate, including performance, security and quality standards:

1. Security:
    - All financial and personal data must be encrypted using strong algorithms (e.g., Fernet)
    - The system must follow GDPR standards and Islamic ethical principles in data handling.
2. Usability:
    - Interface must be clean, intuitive and responsive on both desktop and mobile devices.
    - User guidance must be clear with appropriate form validation and error messaging.
3. Performance:
    - ML predictions should return results in under 3 seconds
    - Forecast graphs should load efficiently, even with up to 24 months of historical data.
4. Scalability:
    - The platform must support concurrent usage by multiple users without performance degradation.
5. Availability:
    - The application should maintain 99.9% up time with minimal scheduled maintenance interruptions.
6. Auditability:

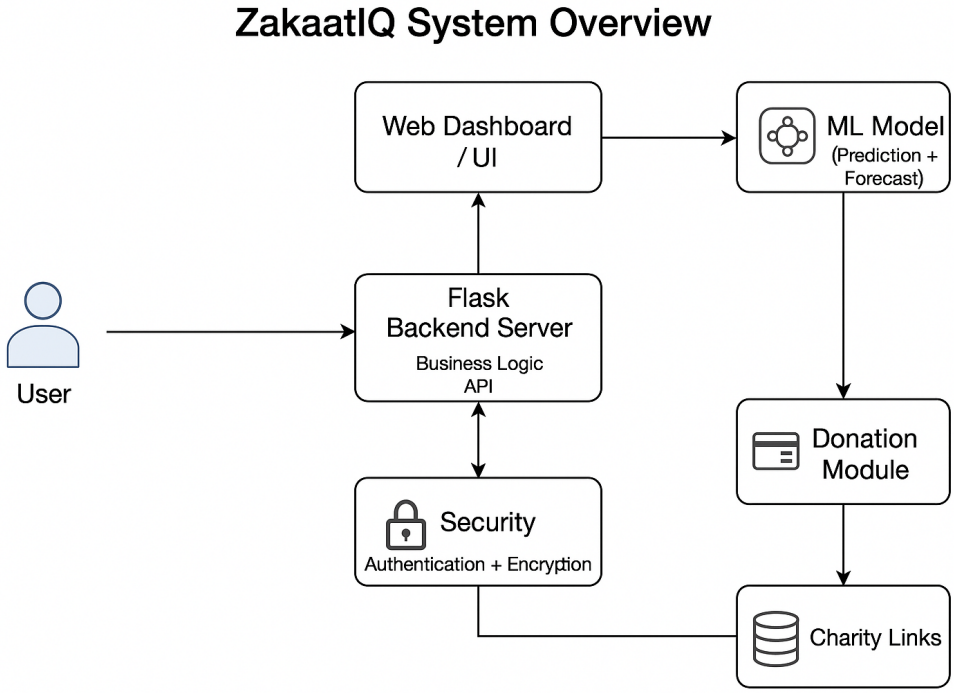
- All donation transactions must be timestamped and logged, and accessible through the user dashboard for review.

7. Interoperability:
   - The system must integrate seamlessly with third-party payment providers like PayPal.

8. Maintainability:
   - Codebase should be modular, with cleat documentation to support future updates and debugging.
9. Accessibility:
   - The system must be compatible with screen readers to support visually impaired users.
   - All UI elements (buttons, form fields, links) must have accessible labels (aria-label, alt-text, etc).
   - The application should support multiple language options, with priority for English and Arabic.
   - Text contrast and font size must comply with standards.

3.6 Initial System Specification

The architecture of ZakaatIQ is based on a modular web-based client-server architecture, ensuring scalability, maintainability and security. The system leverages a Flask backend, a secure frontend interface and modular components for AI processing and donations. Below is the system architecture diagram along with a breakdown of each role.



User:

Role: The end user interacts with the application through the web interface.
Functions: Can log in, input financial data, view Zakaat eligibility results, forecast income and make donations.
Security note: Every interaction is validated via authentication and encrypted to maintain data integrity and privacy.

Web dashboard /UI:

Type: Frontend component (HTML, CSS, JavaScript – React and BootStrap
Purpose: This is the primary interface through which users interact with ZakaatIQ
Features:

- Displays form for inputting data
- Shows Zakaat eligibility results
- Renders interactive graphs for forecasting
- Facilitates secure donation flow

Design focus: Clean, responsive layout for usability on both desktop and mobile

Flask Backend server (Business logic/API):

Type: Python Flask-based backend
Responsibilities:

- Handles routing and API calls
- Processes logic for Zakaat calculations, forecasts and donation workflows
- Manages communication between UI, ML models and the database

Scalability: Flask is lightweight but extensible, making it suitable for cloud deployment.

ML Model (Prediction + Forecast)

Type: Machine learning module
Purpose: Provides 2 core AI-driven services:

1. Zakaat eligibility prediction – Classifies whether the user must pay Zakaat based on their financial input
2. Income & savings forecasting: Uses time-series models (e.g., Prophet) to predict future financial trends

Deployment: Model runs as a backend service or script, triggered via Flask API

Security (Authentication + encryption)

Purpose: Enforces secure access and protects user data
Components:

1. User authentication – login/register flow using secure password hashing
2. Data encryption – financial data is encrypted before being stored in the database (e.g., Fernet encryption)
3. GDPR & Islamic Ethics compliance: ensures ethical handling of sensitive data

Donation module

Function: Facilitates online Zakaat donations securely
Features:

- Connects to verified payment gateways (e.g., PayPal, Stripe)
- Displays list of approved Islamic charities
- Processes transaction and logs receipts for the user

Security: Integrates securely with the backend to prevent misuse or fraud

Charity links / Database

Type: PostgreSQL or similar relational database
Functions:

- Connects to verified charity information
- Maintains donation records

- Supports user dashboard with historical donation, forecast and eligibility data

Scalability: Optimised for read-heavy operations with appropriate indexing

Why this architecture was chosen:

I selected this architecture for the following reasons:

1. Modularity – Each component (frontend, backend, ML, donations) is independent of each other in development allowing for easier testing, debugging and scaling.
2. Security first – Sensitive data is handled only on the backend, with encryption and validation at all points.
3. AI integration – Allows real time ML predictions and forecasts via isolated models triggered by backend logic.
4. User centric design – Focused on clean UX and responsive frontend, tailored to a non-technical audience.
5. Future ready – Can be deployed on cloud platforms like AWS and scales as user base grows.

3.7 Conclusions

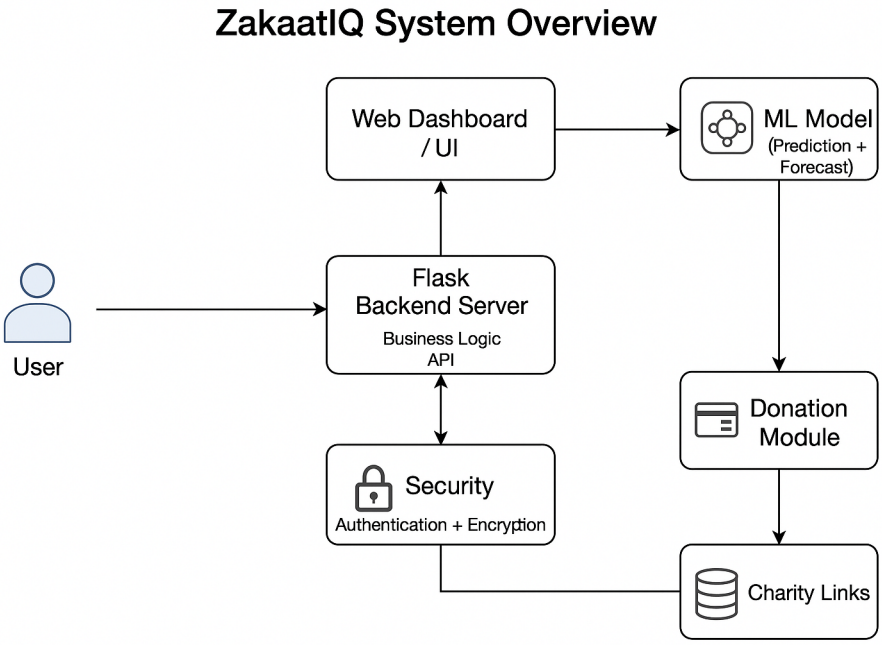
This section provided a comprehensible analysis of the ZakaatIQ system, outlining its overall purpose, user needs, functional and non-functional requirements along with system architecture. Through user focused research and stakeholder feedback, I was able to identify challenges faced by Muslims in managing and calculating their Zakaat. ZakaatIQ will be designed with these in mind, making a secure, intelligent web-based platform that combines AI prediction models, encrypted data handling and a user-friendly dashboard to streamline Zakaat eligibility checks, financial forecasting and charitable donations.

Detailed use cases were developed to capture system behaviour while the chosen client-server architecture supports modular development, future scalability and ethical data practices. The systems design and requirements reflect both technical aspect and alignment with Islamic financial principles. This foundation sets the stage for effective development and testing in the next phase of the project.

4. System Design

4.1 Introduction

The design of the web app focuses on translating the system requirements – functional, non-functional, ethical and accessibility-related into a robust, user friendly and secure web application. Given the sensitive nature of financial and religious data, the system must be transparent, reliable and inclusive for a wide range of users, including those with disability. The application is being developed using a modular, layered architecture that allows secure processing, AI – driven decision support and accessibility compliance under the European Accessibility Act.



As shown in the system architecture design, the ZakaatIQ system architecture includes a Flask backend, a secure encryption module and a responsive web dashboard. In this section, I will detail how this design ensures accessibility, data protection and maintainability across all modules.

4.2 Software Methodology

The development of the ZakaatIQ system will follow an adapted agile methodology, tailored to suit a lone developer building a web-based application with integrated machine learning components. This choice reflects the projects need for iterative progress, early feedback and continuous validation of both technical accuracy (e.g., ML predictions) and usability (accessibility and donation flows).

Common methodologies:

Several software development methodologies were considered.

Methodology	Description	Reasoning
Waterfall	A sequential approach where requirements are fixed upfront, followed by design, development and testing	Too rigid – not flexible and difficult to change once started. Not suitable for model tuning, UI/UX Testing or evolving accessibility needs
V-Model	Emphasizes validation and verification alongside each development step	Strong for critical systems, but too document heavy for agile nature of project
RAD (Rapid App Development)	Focuses on prototype and fast iteration	Doesn't emphasize testing and backend integrity for Islamic and financial use cases
Agile	Iterative development with short sprints (small chunks instead of building everything at once), frequent reviews and evolving requirements	Best fit for the project – supports constant evaluation of ML models, UX adjustments and stakeholder feedback

Chosen approach – Adapted Agile

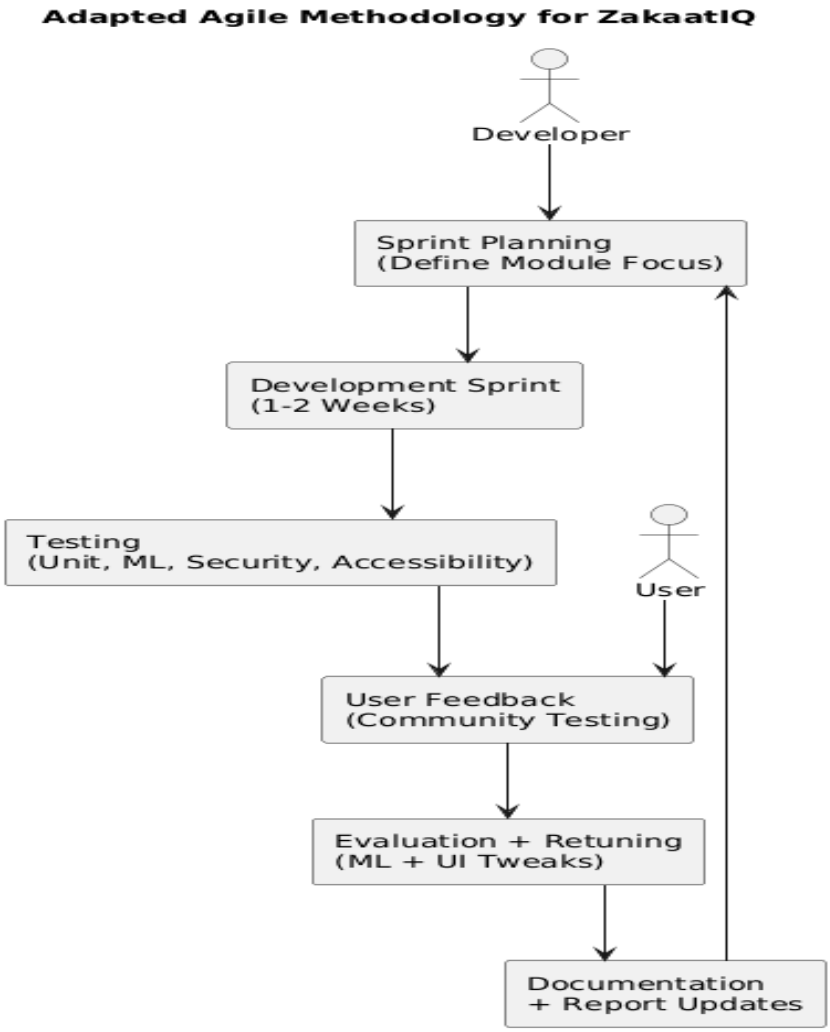
Although the project is being developed primarily by a lone developer, the agile methodology has been adapted to allow for:

- Short development sprints focusing on a single feature/module (e.g., authentication, forecasting)
- User-centric design, with feedback from real users at different stages.
- Incremental testing and integration, especially important for machine learning and donation API's.
- Continuous documentation, aligned with requirements

Key agile adaptions for ZakaatIQ

Agile Element	**ZakaatIQ Adaptation**
Sprints	1–2-week coding windows, each targeting a functional module (ML, UI form, encryption etc)
User stories	Developed based on a real user need (e.g., "As a Muslim user, I want to check if I owe Zakaat")

Testing + evaluation	Done after each feature sprint, including ML validation, screen reader support and form usability
Feedback loops	Occur after testing with users from the Muslim community, using surveys and think-aloud protocols



Agile is ideal for the project as:

- ML models and forecasts require tuning over time; Agile supports re-training and integration testing easily
- User needs are unique (Islamic + accessibility) – Agile allows rapid adaptation based on feedback
- Accessibility and language features need real-world validation, which is best handled in Agile-style iterations
- The European Accessibility Act and ethical implications demand continuous testing and improvement – not one-time fixes

4.3 Overview of System

Logical Architecture

ZakaatIQ is a modular, web-based system designed using a clear client-server architecture. It separates concerns across 4 main layers:

1. **User feature (Frontend)**
    - Technology: HTML, CSS, JavaScript, Bootstrap
    - Function: Delivers the interface through which users interact with the system. It displays forms for data input (financial and lifestyle), dashboards, visualisation of forecasts and secure donation tools.
    - Accessibility: Bootstrap ensure responsive design across devices. To meet the European Accessibility Act (2025), ARIA roles and semantic HTMK will be used for screen reader compatibility. Common screen readers like NVDA (Windows), VoiceOver (MAC) and ChromeVox (Chrome) will be tested.
2. **Application Layer (Flask Backend)**
    - Technology: Flask (Python)
    - Function: Manages all business logic – routes user actions, handles data validation, authentication and encryption. It also triggers machine learning models for Zakaat prediction and forecasting.
3. **AI Processing layer**
    - Technology:
      Classification model – Logistic Regression used due to its interpretability and suitability for binary classification
      Forecasting model – Prophet; time-series forecasting model by Facebook, ideal for seasonal and financial trend predictions
      Function: Delivers intelligent financial insights to users through predictive models trained on synthetic data
4. **Data layer (Database)**
    - Technology: PostgreSQL
    - Function: Stores user profiles, financial data, donation history, encrypted assets and prediction results
    - Security: Implements encryption at rest and strict access control
5. Charity Donation Gateway
    - Integrated platform like Stripe allowing real time transactions with verified Islamic Charities.

Deployment environment (Physical infrastructure)

Component	Tool / Environment	Notes
Web hosting	Render / Heroku / AWS EC2	Cloud-based hosting for Flask app with PostgreSQL DB integration.
Code repo	GitHub	Full version control, team collaboration, and deployment pipelines.
Local development	VS Code + Git	All modules are developed and tested locally before deployment.
Browser testing	Chrome, Firefox, Safari	Ensuring compatibility, responsiveness, and accessibility.
Screen reader testing	NVDA, VoiceOver, ChromeVox	Verifying blind-user accessibility in key areas of the site.

Language and Accessibility Integration

- Multilingual Support will be implemented using Google Translate API (or i18n JavaScript libraries) to enable automatic translation of core UI elements into Arabic, Urdu, and Turkish.
- Screen reader compatibility:
  HTML labels and ARIA roles will be used for form fields
  Navigation will be optimised for keyboard-only usage
  Tested with free tolls like NVDA (on Windows) and VoiceOver (on MAC)

Why this design?

- **Modular** – Makes the system scalable, testable, and maintainable.
- **Accessible** – Ensures compliance with European Accessibility Act and usability for blind or non-English speaking users.
- **Ethical** – Follows Islamic finance principles with transparency and security.
- **User-Friendly** – Focused on non-technical Muslim users needing intuitive guidance.
- **Future-Ready** – Deployable on cloud, supports real-time AI, and allows for mobile-first use.

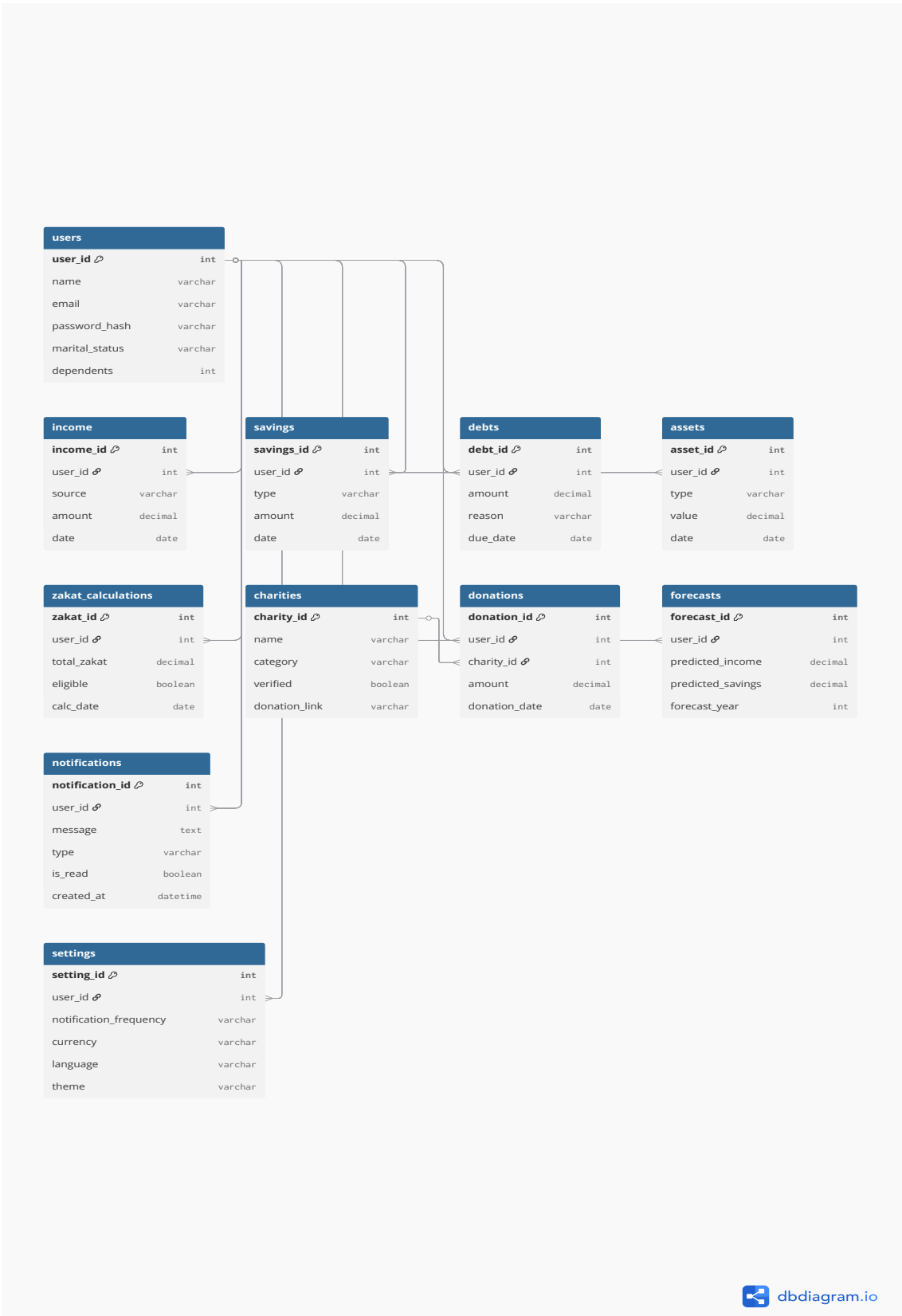
4.4 Design System

Entity Relationship Diagram

The diagram down below represents the core database schema for the ZakaatIQ system. Key entities include:

- **Users;** core table storing personal user details
- **Income, savings, debts, assets**; Financial data tables used in Zakat calculation and forecasting
- **Zakat calculation;** stores the outcome of eligibility predictions
- **Charities & donations**; link users to Islamic charities with donation tracking
- **Forecasts;** stores AI-predicted income/savings for the future
- **Notifications & settings**; support personalization and communication

This structure ensures data normalization, traceability and compliance with GDPR and Islamic finance principles.

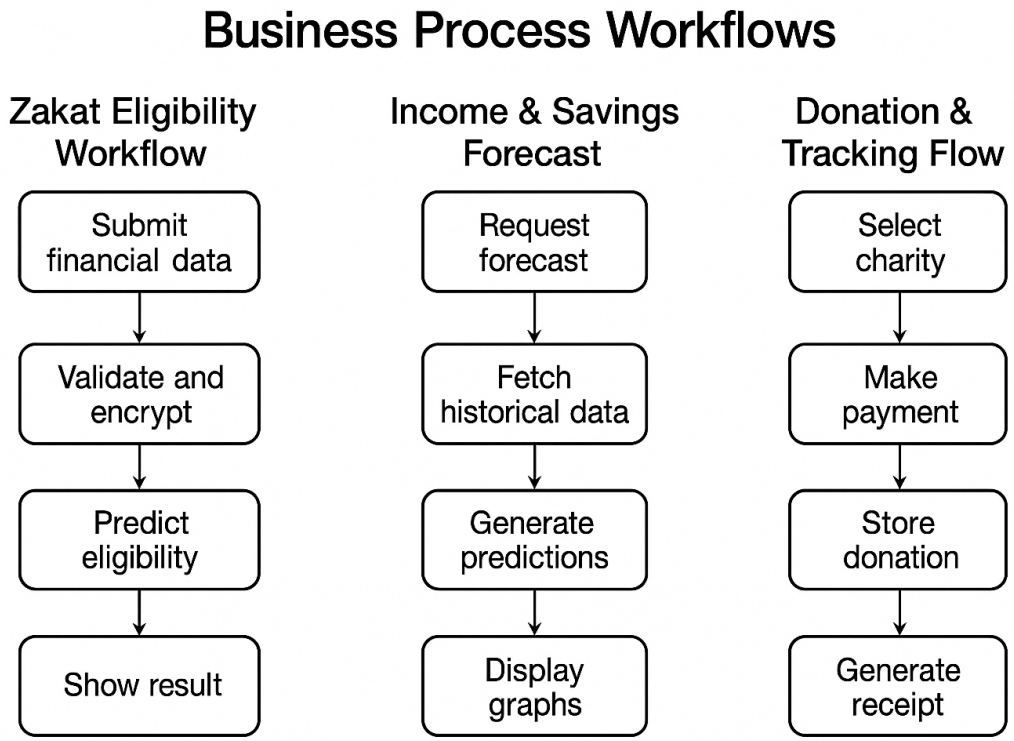


Business Process Logic

The systems logic is built around 3 main workflows:

1. Zakat eligibility workflow
    - User submits financial data – system validates and encrypts – ML model predicts eligibility – result shown and stored.
2. Income and savings forecast
    - User requests forecast – system fetches historical data – passes to linear regression model – predictions shown as graphs
3. Donation & tracking flow
    - Eligible user selects charity – makes payment via secure gateway – donation stored – receipt granted

These workflows are modular, independently testable and ensure traceability.



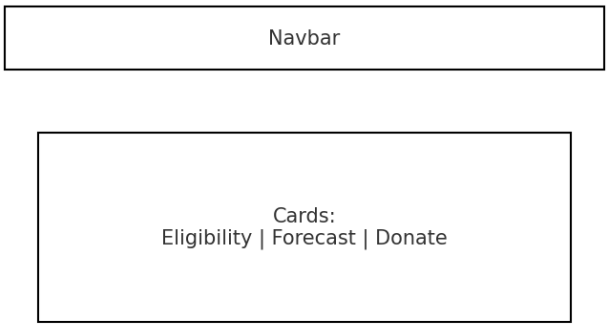
UI/HCI Design

The web-based interface will prioritize accessibility and ease of use, including:

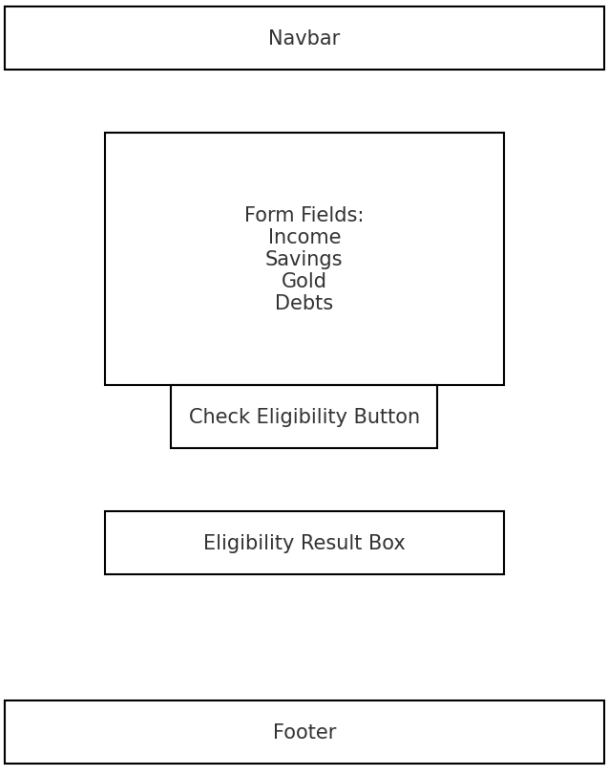
- Screen reader compatibility – HTML5 with semantic tags, labels for React components
- Language support – User settings allow switching between multiple languages using i18n libraries
- Bootstrap-based UI – Ensure responsive design across mobile, tablet and desktop screens
- Keyboard navigation – All input fields, buttons and dropdowns will support tab-based interaction

Tools used – Bootstrap, VoiceOver, React

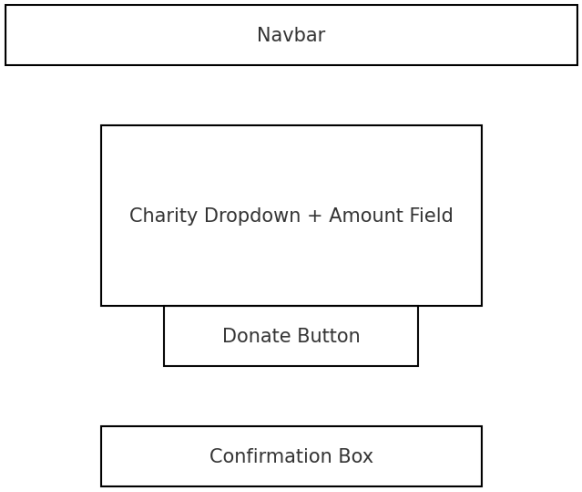
Dashboard Page Wireframe



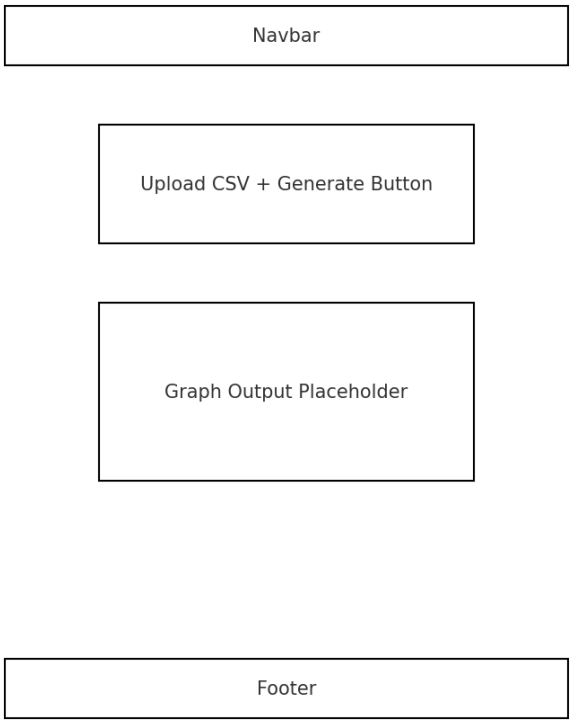
Zakat Eligibility Wireframe



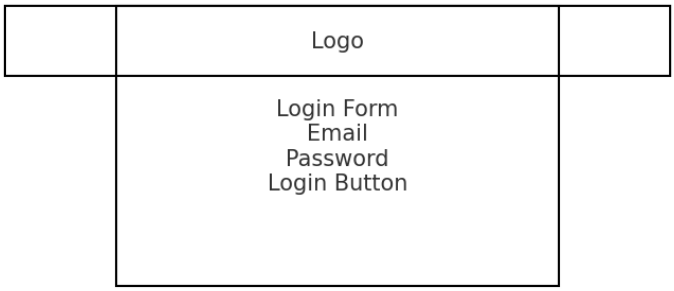
Donate Page Wireframe



Forecast Page Wireframe

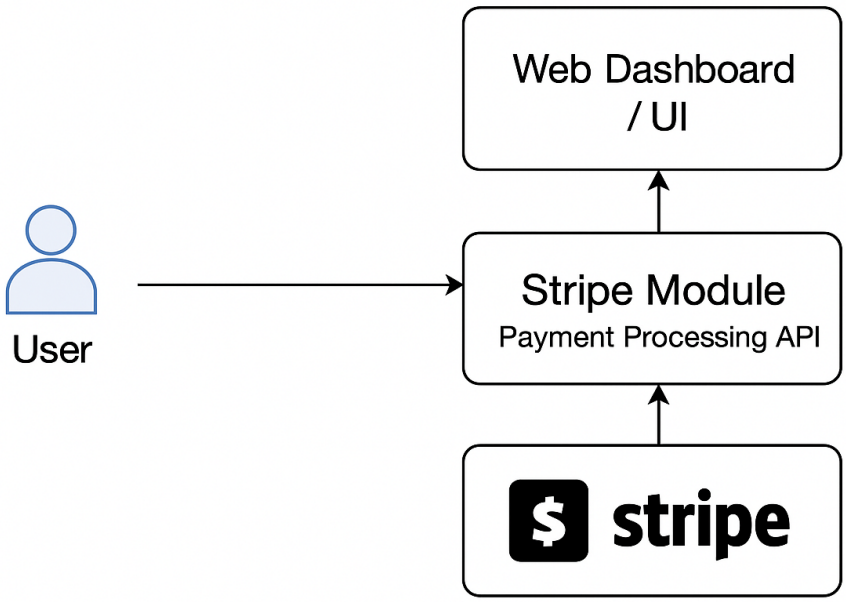


Login Page Wireframe



Stripe diagram flow:
The stripe module is responsible for processing secure online donations within ZakaatIQ platform and ensures that users Zakat contributions are transferred safely to verified charities.

1. User → stripe:
   User initiates donation via dashboard
   Action processed by Stripe module, which handles transaction securely
2. Stripe module → Stripe API
   Module acts like a bridge between app and actual Stripe Payment API, sending payment details to Stripe for processing.
3. Stripe → Stripe modules (response)
   Stripe returns a transaction response (success, failure, receipt, etc.) back to Stripe module.
4. Stripe module → Web dashboard /UI
   Stripe module returns the result to the User interface.
   User is shown a confirmation, receipt or error message depending on outcome.



4.5 UI

4.6 Conclusions Systems design

The ZakaatIQ system has been thoughtfully designed to align with Islamic financial principles, modern web technologies, and user accessibility standards. Each component—from secure user login to AI-powered Zakat prediction and encrypted charity donations—has been mapped out through wireframes, use case diagrams, logical architecture, and data models.

The use of a modular system design with clear workflows (eligibility, forecasting, and donation) ensures testability and scalability. Technologies such as Flask, PostgreSQL with Fernet encryption, and Bootstrap for front-end accessibility support our objectives for security, user-friendliness, and compliance with standards like the European Accessibility Act.

The adoption of an adapted Agile methodology ensures continuous iteration and integration of user feedback—particularly important given the ethical and religious sensitivities surrounding Zakaat calculation and handling of donations.

This system design forms a strong, practical foundation for implementation, ensuring that the final product will be technically robust, religiously ethical, and genuinely helpful for its users.

5. Testing and Evaluation

5.1 Introduction

Testing and evaluation are essential to ensure the reliability, accuracy, usability and security of the ZakaatIQ web application. Given that the system is divided into multiple logical modules (e.g., ML-Based Zakaat prediction, AI forecasting, secure donations), each component will be individually tested (unit testing) before being tested in integration.
The application handles sensitive financial and personal data and includes complex AI predictions, so a combination of black-box, white-box, user-based and model evaluation testing will be applied. Due to the ethical and religious implication of inaccurate predictions, the system must be both technically correct and easy to use for its intended audience.

5.2 Plan for Testing

Testing will follow modular and iterative approaches, focusing on both functionality and security

Module	Test Type	Description
User Authentication	Unit testing	Test user login, registration, session control and input validation
Zakaat eligibility prediction	ML Evaluation + functional testing	Test data input validation, model output correctness, edge case handling (missing/invalid data) and logical errors in threshold comparison (Nisab). Evaluate ML model using accuracy, precision, recall and F1-score
Income & savings forecasting	Functional + regression test	Test forecast model reliability using historical data and verify visual outputs on dashboard. Check for model errors and retry logic
Charity donation integration	Integration testing	Test secure connection with PayPal. Validate charity list loading and donation flow from start to receipt generation
Data encryption	Security testing	Test if all financial inputs are encrypted using Fernet before database write. Test retrieval logic to decrypt correctly when displayed
Dashboard	UI + Functional	Ensure historical records, graphs and insights are correctly linked to the user account and updates in real time.
Accessibility (Screen Reader + multi-language)	Accessibility testing	Verify screen reader compatibility for all UI

		elements. Test proper label tags, logical navigation order, dynamic content announcements. Validate that language toggle works and UI updates correctly in selected language. Ensure all important content is perceivable without visuals.

Security Testing:

- Attempt input injection (XSS,SQLi)
- Verify all data-at-rest and data-in-transit are encrypted
- Ensure password hashing (bcrypt) is implemented
- Simulate unauthorized access and ensure proper access control

UI and Usability Testing

- Test on multiple screen sizes and browsers (mobile, tablet, desktop)
- Use surveys and observation-based feedback sessions with real users – evaluation section below
- Test the interface with screen readers like NVDA (Windows) and VoiceOver (MAC)
- Use accessibility auditing tools (e.g., WAVE, Lighthouse – research more) to detect issues
- Verify tab navigation, label tags and alt text for all UI components
- Language switcher tested for accurate translations and dynamic updates
- Get feedback from visually impaired users if possible.
- Test that languages are available and give correct translation

5.3 Plan for Evaluation

ZakaatIQ will be evaluated on both technical performance and user experience. The evaluation will occur in the final testing phase and will inform any final modifications

Evaluation of ML models

1. Zakaat eligibility classifier (Decision Tree / Logistic regression):
   - Evaluate on labelled synthetic datasets
   - Use metrics: Accuracy, precision, Recall, F1-Score
2. Forecasting model (Prophet / Linear Regression):
   - Evaluate using Mean absolute error (MAE), Mean Squared error (MSE).
   - Visual check of trends (12-month forecast curves)

User evaluation

User Group	Evaluation Method	Feedback Focus
Muslim working adult	Usability test + survey	Clarity of Zakat results, accuracy, ease of use.
Married user with dependents	Think-aloud protocol	Completeness of financial input form, lifestyle tailoring.
Self-employed	Task-based testing	Forecast reliability, usefulness of advice.
Eligible Zakat recipient	Observation + feedback	User interface clarity, donation flow access.

Visually impaired user	Screen reader task walkthrough + feedback	How accessible is the platform without visuals? Can all functions (eligibility, forecast, donation) be done with keyboard and screen reader?
Non-English-speaking user	Multi-language switch testing + survey	Accuracy of translated content, ability to navigate and understand without English fluency.

Feedback is conducted through structured Microsoft Forms and interviews in the Muslim Community. This ensures the final product aligns with their needs as a user.

System performance evaluation

- Prediction latency: < 3 seconds per result
- Load handling: Test with simultaneous users (20+)
- Dashboard responsiveness: < 1 second load time
- Error handling: Track exceptions and log crash reports

5.4 Conclusions

My testing and evaluation are designed to ensure that each component of ZakaatIQ is functionally correct, secure, ethical, responsive and easy to use. It integrates technical validation (e.g., ML accuracy, encryption success) with real-world user testing and feedback, ensuring both Islamic compliance and practical utility.

Future testing will be extended post-launch using automated test suites, real-time user feedback and possible A/B testing for improvements to ML models and interface. This will ensure continuous system evolution and used satisfaction aligned with both technical and religious goals.

6. System Prototype

This section shows the working prototype of ZakaatIQ, showing how each subsystem was implemented according to the requirements mentioned earlier. A prototype-first approach was used to ensure core functionality was working before refining a final UI.

The prototype demonstrates:

- Machine Learning–based Zakat eligibility prediction
- Live system pages (Home, Login, Forecast, Donate)
- Encrypted user inputs
- Graph generation for financial forecasting
- A unified layout using base.html and template inheritance
- A Bootstrap-powered responsive UI

6.1 Introduction

This section presents the initial prototype of the ZakaatIQ web application. The prototype was developed following the system design, requirements, and architecture outlined in previous sections. It represents the first functional version of the system, demonstrating technical feasibility and validating that the chosen technologies work together as intended.

Prototype Development Approach

The prototype was developed using an adapted Agile methodology, where features were built in small sprints and tested iteratively. Each sprint targeted one functional module (e.g., authentication, eligibility prediction, donation flow). This approach ensured continuous feedback, allowed rapid correction of errors, and supported the integration of accessibility improvements, as highlighted in Section 5.

The primary goal of the prototype was to implement and test the **core logic of ZakaatIQ**:

1. A working ML model for Zakat eligibility
2. A forecasting module
3. A secure backend with encryption
4. A responsive user interface
5. A functional navigation structure
6. Integration of Bootstrap-based styling

Frame works & tools used

The prototype uses the technology stack defined in earlier chapters:

Area	Technology used
Web framework	Flask
UI framework	Bootstrap 5 (KindHeart Charity Template)
Templating	Flask / Jinja2
Machine learning	Scikit-learn
Data processing	Pandas, NumPy
Encryption	Fernet (Cryptography library)
Forecasting visuals	Matplotlib (Agg backend)
Version control	Git + GitHub
Assets	Custom hero images, Islamic calligraphy

Coding environment	Virtual Studio Code

Bootstrap Template Integration

Originally, the system used a **full Bootstrap charity theme** (KindHeart 1.0.0) for every page.
Later in the prototype phase:

- Removed unnecessary pages
- Extracted UI elements we needed
- Compressed design to one central template (base.html)
- Preserved responsive styling while shifting to a minimal Islamic finance theme

This resulted in a cleaner, custom-designed look — while still being fully Bootstrap responsive.

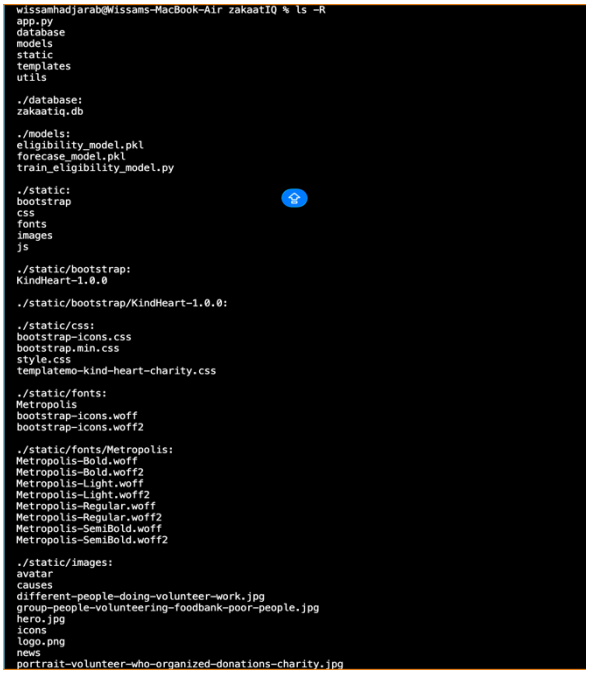
This combination ensures a secure, modular, and scalable foundation, consistent with the system architecture defined in Section 4

6.2 Prototype Development

This section describes how each component of the ZakaatIQ prototype was developed and implemented according to the logical architecture defined in Section 3. The development process followed a modular approach, with separate layers for UI, backend logic, machine learning, encryption, and system routing. Each functional module was incrementally built and tested before being integrated into the full Web application.

Project structure & setup

Project directory structure:



```
wissamhadjarab@Wissams-MacBook-Air zakaatIQ % ls
app.py                  models                  utils
database                static
eligibility_model.pkl   templates
wissamhadjarab@Wissams-MacBook-Air zakaatIQ %
```

- /**templates** — Contains base.html and all feature pages (login, eligibility, donate, forecast, etc.).
- **app.py** — Main Flask application (routing, ML loading, encryption, rendering).
- /**static** — Contains your Bootstrap assets, CSS, JavaScript, fonts, and hero images.
- /**models** — Contains the ML scripts and saved model (eligibility_model.pkl).
- /**utils** — Contains encryption logic.
- /**database** — Placeholder database file for future persistent storage.

- Prototype follows a modular and scalable architecture, separating UI, logic, ML, assets and utilities – as mentioned in Section 3

Backend Application (Flask routing)

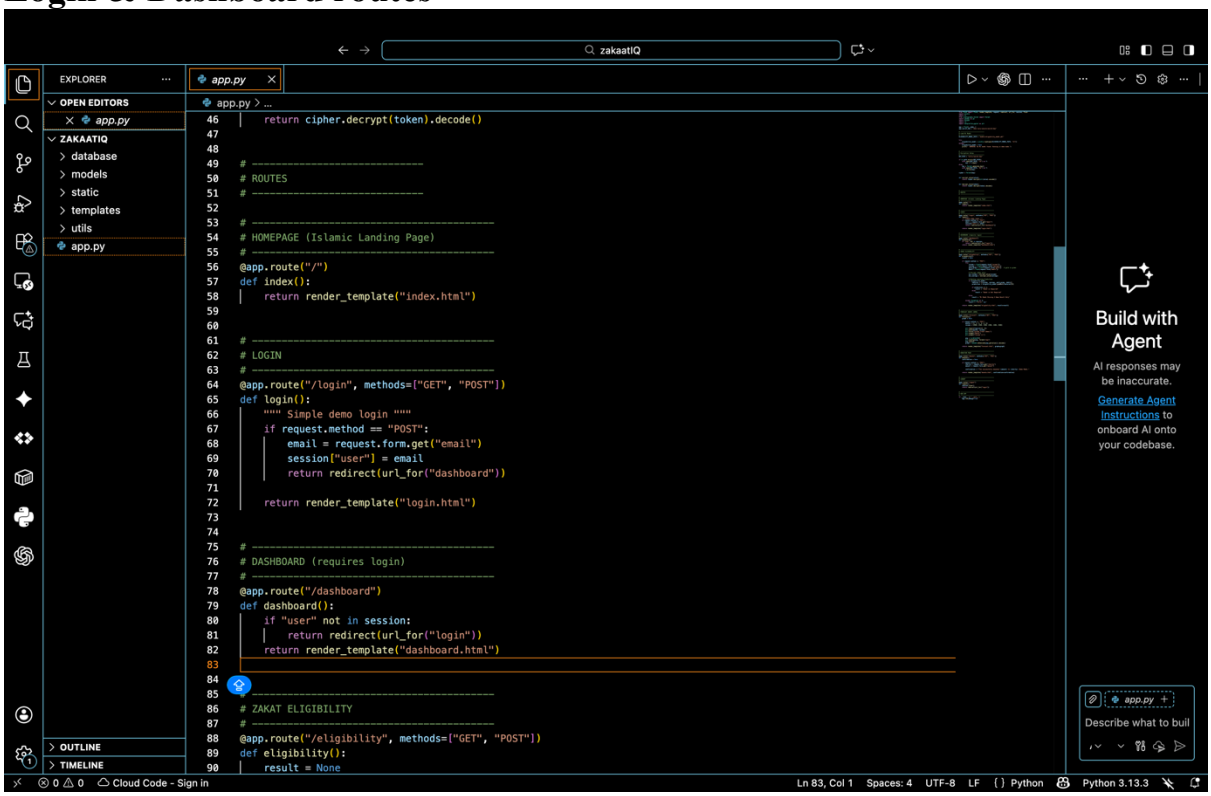
```python
from flask import Flask, render_template, request, redirect, url_for, session, flash
import pickle
import os
from cryptography.fernet import Fernet
import pandas as pd
import base64
import io

import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
```

Screenshots show:

- Importing Flask, ML libraries, encryption tools and plotting tools
- Loading the trained eligibility model using pickle.load
- Setting up Fernet encryption(cipher)

ML model load – allows real time predictions during form submission
Fernet encryption – protects sensitive financial inputs (income, savings, debts)
Error handling – ensures app still runs even if model is missing

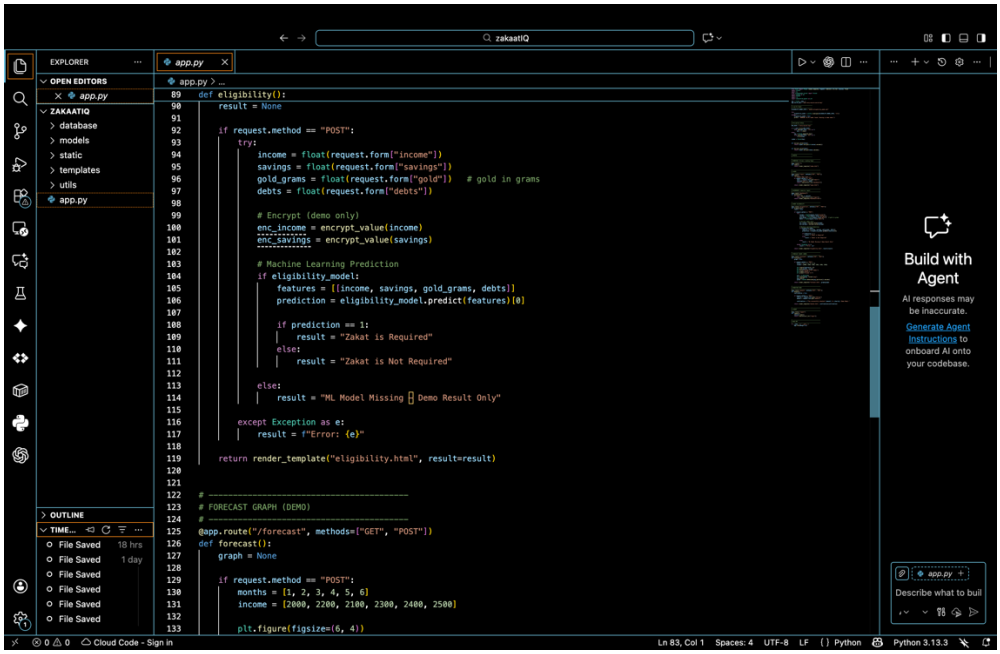
Login & Dashboard routes



- The login route accepts user input and starts a session.
- The dashboard route checks whether the session exists.
- If user is not logged in → redirect to login.

Session based authentication implemented.

Zakaat Eligibility Logic



Screenshot contains core AI logic and ML prediction. It demonstrates:

- Form data is captured
- Values are fed directly into the ML model
- Output (Zakaat required / not required) is generated dynamically
- Inputs are encrypted before usage

Forecasting Logic

Shows the function for generating income/savings forecast graphs:

- Creates a Matplotlib graph
- Saves it to memory with BytesIO()
- Encodes as Base64 so HTML can render it
- Displays it inside the forecast page dynamically

Because macOS cannot open GUI windows – import matplotlib ; matplotlib.use ('Agg') was added to prove graph generation works, system is adapted for non GUI environment.

Machine Learning Development

```
# ------------------------------------
# TRAIN MODEL
# ------------------------------------

X = df[["income", "savings", "gold_grams", "debts"]]
y = df["zakat_required"]

model = DecisionTreeClassifier(max_depth=6)
model.fit(X, y)

print("Model trained with REAL Islamic values using live prices.")
```

```
# ------------------------------------
# SAVE MODEL
# ------------------------------------

with open("eligibility_model.pkl", "wb") as f:
    pickle.dump(model, f)

print("eligibility_model.pkl created successfully.")
```

```python
# ----------------------------------------
# LIVE GOLD & SILVER PRICES USING GOLDAPI.IO
# ----------------------------------------

GOLD_API_KEY = "goldapi-a77fy7smi9hhlww-io"   # WORKING KEY

def get_live_gold_price_eur():
    url = "https://www.goldapi.io/api/XAU/EUR"
    headers = {"x-access-token": GOLD_API_KEY}

    try:
        r = requests.get(url, headers=headers).json()
        if "price_gram_24k" not in r:
            raise Exception(r)
        return r["price_gram_24k"]  # Gold €/g

    except Exception as e:
        print("⚠ GOLD API error:", e)
        print("Using fallback gold price: 60 €/g")
        return 60.0

def get_live_silver_price_eur():
    url = "https://www.goldapi.io/api/XAG/EUR"
    headers = {"x-access-token": GOLD_API_KEY}

    try:
        r = requests.get(url, headers=headers).json()
        if "price_gram_24k" not in r:
            raise Exception(r)
        return r["price_gram_24k"]  # Silver €/g

    except Exception as e:
        print("⚠ SILVER API error:", e)
        print("Using fallback silver price: 0.70 €/g")
        return 0.70
```

```
wissamhadjarab@Wissams-MacBook-Air zakaatIQ % python3 models/train_eligibility_model.py

Live Gold Price (€/g): 113.513
Live Silver Price (€/g): 1.3964
Nisab (Euro): 855.099504
Model trained with REAL Islamic values using live prices.
eligibility_model.pkl created successfully.
wissamhadjarab@Wissams-MacBook-Air zakaatIQ %
```

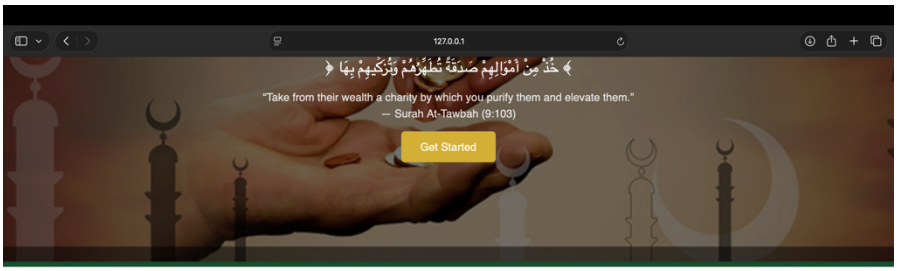
The screenshots show:

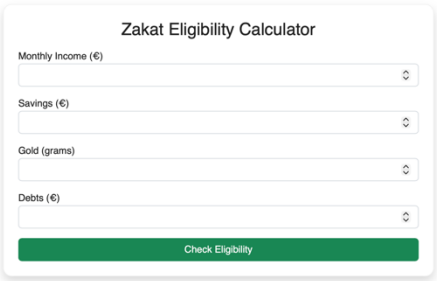
- Live gold/silver API price retrieval
- Conversion from troy ounces → grams
- Calculation of Nisab using real Islamic rules
- Dataset generation (synthetic but realistic)
- Decision tree training
- Model saved as eligibility_model.pkl

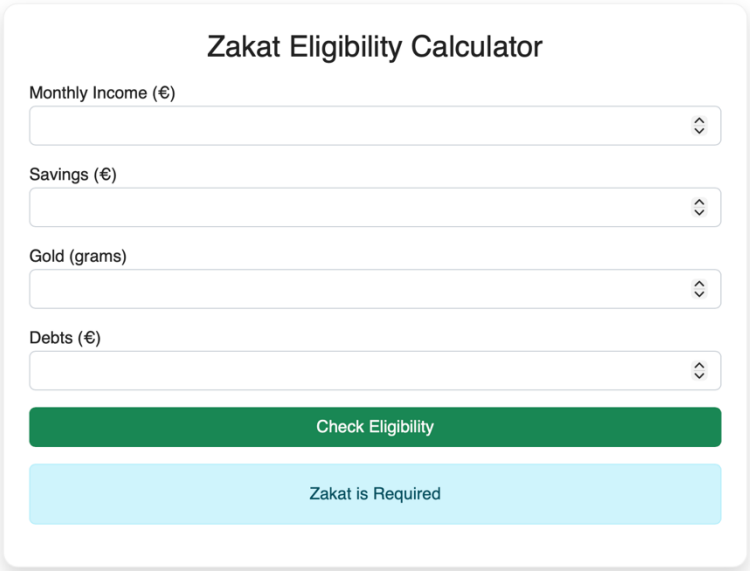
This process ensures:

- The ML model reflects real Islamic financial thresholds
- AI predictions follow Islamic jurisprudence
- The system generates data when none is available
- ML pipeline meets requirements in Section 2

Zakaat Eligibility Page Functionality





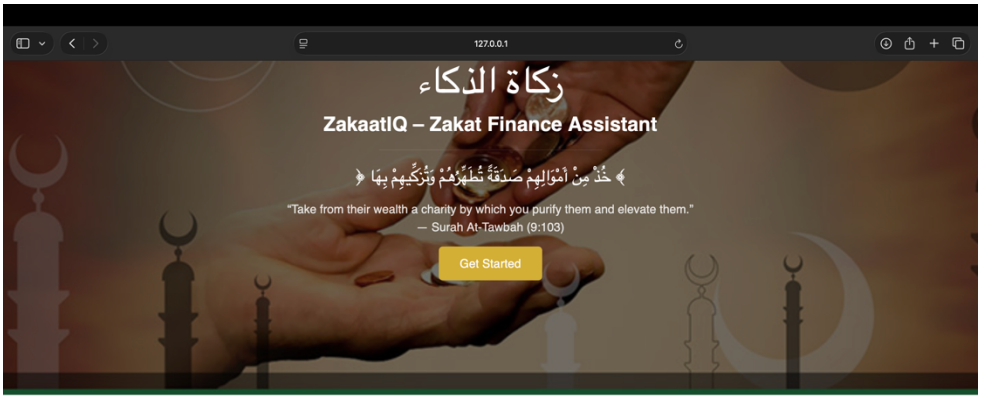


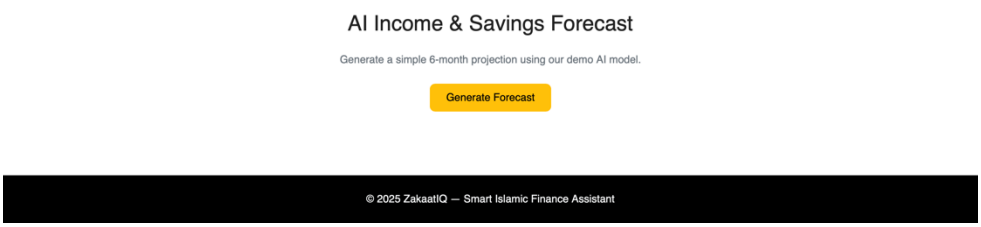
These screenshots show:

- Input form (income, savings, gold grams, debts)
- ML prediction output after submitting

This confirms that the form submission works, the ML model is integrated into the web interface, results are dynamically displayed and the template inheritance works (same navbar, footer, styling)

Forecasting Module





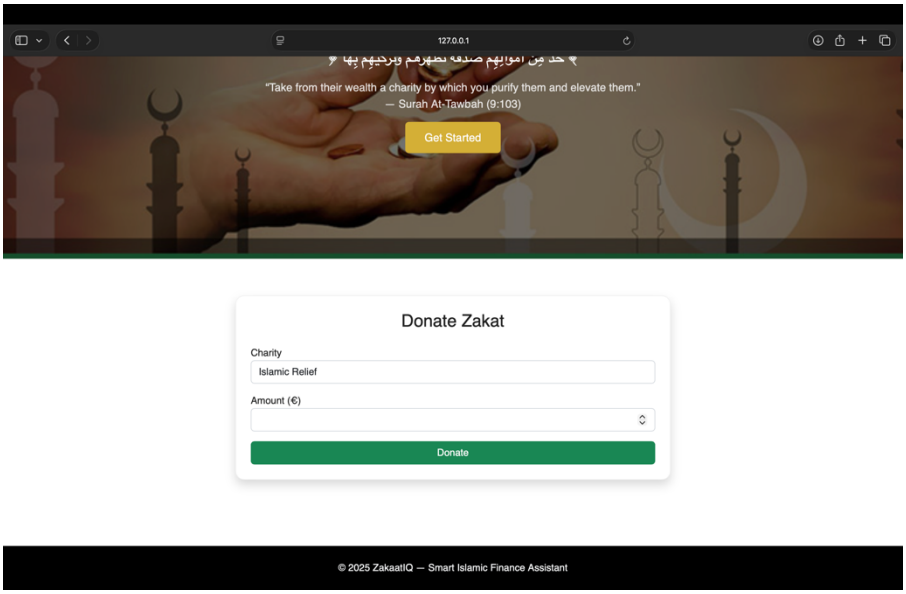
```python
# ──────────────────────────────────────────────
# FORECAST GRAPH (DEMO)
# ──────────────────────────────────────────────
@app.route("/forecast", methods=["GET", "POST"])
def forecast():
    graph = None

    if request.method == "POST":
        # Demo forecast values
        months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun"]
        income = [2000, 2100, 2200, 2300, 2400, 2550]

        plt.figure(figsize=(6,4))
        plt.plot(months, income, linewidth=3)
        plt.title("6-Month AI Forecast", fontsize=14)
        plt.xlabel("Month")
        plt.ylabel("Income (€)")

        # Convert graph to base64
        png = io.BytesIO()
        plt.savefig(png, format="png", bbox_inches="tight")
        png.seek(0)
        graph = base64.b64encode(png.getvalue()).decode()

    return render_template("forecast.html", graph=graph)
```

These images demonstrate:

- User accessing the forecast page
- Graph loading correctly
- Python backend rendering graph → Base64 → HTML image

This shows how the forecasting feature works, the interface being responsive and consistent and the backend graph rendering is stable after fixing macOS issues. (rough for demo)

Donation Page Implementation

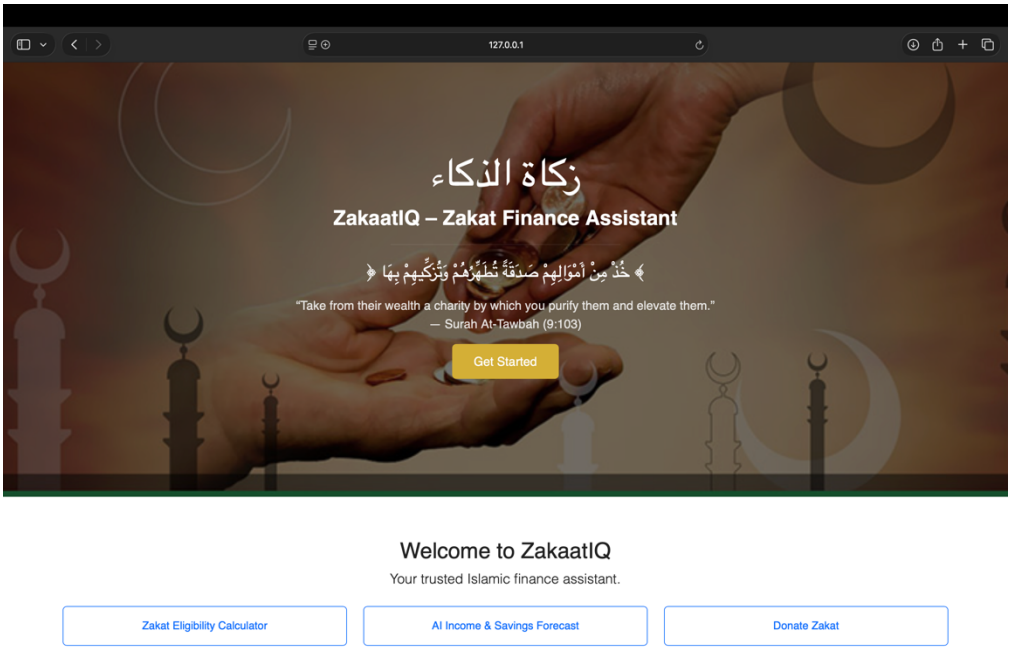


This page includes:

- Select charity field
- Amount input
- Confirmation message after submission

This demonstrates that the form submission works, donation workflow is functional, the UI matches rest of system and logical architecture implemented correctly.

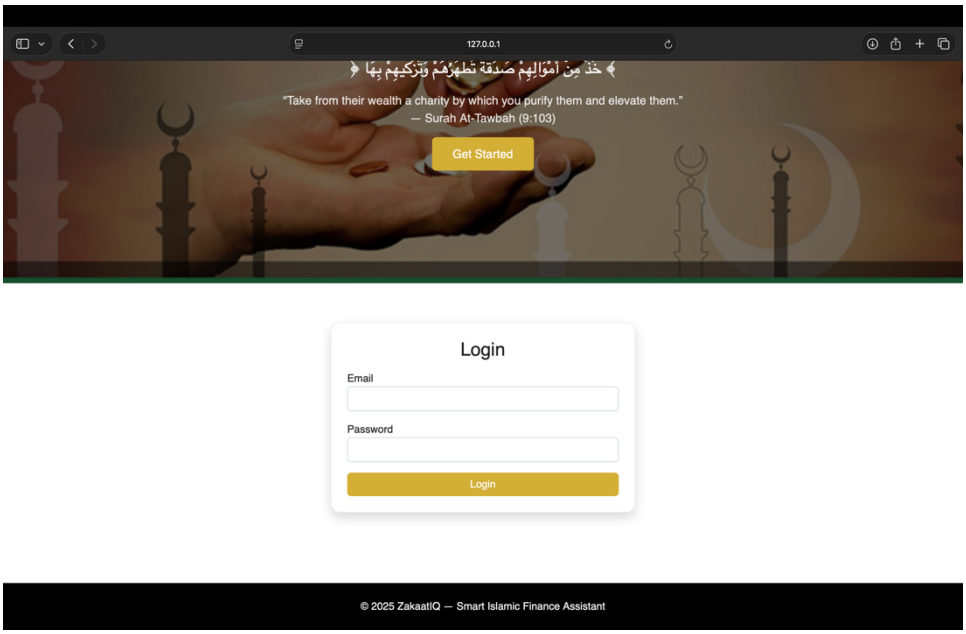
Homepage and UI Framework



The homepage demonstrates:

- Customized hero section
- Bootstrap navbar
- Template inheritance
- Fully responsive layout

Login page



This demonstrates:

- Form handling
- Session creation
- Navigation to dashboard

This shows the authentication subsystem module

Prototype running

```
wissamhadjarab@Wissams-MacBook-Air zakaatIQ % python3 app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 870-610-413
```

This shows:

- All routes load
- ML model loads
- Encryption module loads
- Static files map correctly
- Application runs end-to-end

6.3 Results

This section presents the results of executing the tests defined in Section 5: Testing and Evaluation**.** Each test was performed on the working prototype, and screenshots were taken as evidence. The results demonstrate the extent to which the prototype meets the functional and non-functional requirements at this stage of development.

Test 1 – Homepage Rendering & UI Consistency

Purpose:
To verify that the homepage loads correctly, uses the unified `base.html` layout, and displays the hero section, Quran verse, Arabic calligraphy, and correct styling.

Expected Result:

- Homepage loads without error
- Arabic and English text rendered correctly
- Hero image loads
- Navbar responsive and functional

Actual Result:

- Homepage loaded successfully (see *Homepage.png*).
- Hero section displayed with correct overlay styling.
- Navbar links functional and styled consistently across pages.
- No missing styles, images, or rendering issues.

Test 2 – Login Functionality & Session Creation

Purpose:
To validate that the login form submits correctly and redirects the user to the dashboard while creating a session.

Expected Result:

- Login form accepts input
- Session variable user is created
- User is redirected to /dashboard

Actual Result:

- Login form works (see *loginpage.png*).
- Session stored successfully (appPY2.png).
- Redirects to dashboard page without errors.

Test 3 – Dashboard Page Navigation

Purpose:
To ensure that once logged in, the dashboard loads and all navigation links function correctly.

Expected Result:

- Dashboard displays 3 feature cards
- Links to Eligibility, Forecast, and Donate pages
- Navbar matches the global design

Actual Result:

- Dashboard loads successfully.
- All navigation buttons redirect to correct routes.
- Inheritance from base.html confirmed (consistent UI)

Test 4 – Zakaat Eligibility Input Form Behaviour

Purpose:
To test if the Zakat Eligibility page accepts valid numeric input and submits the data to the backend.

Expected Result:

- Inputs validated
- POST request processed by Flask
- ML model receives all four features
- System returns a prediction

Actual Result:

- Form loads correctly (see *eligibilityCheckPage.png*).
- No input type mismatch.
- POST request succeeds.
- ML model receives [income, savings, gold, debts].

Test 5 – Machine Learning Prediction Output

Purpose:
To test if the Zakat Eligibility page accepts valid numeric input and submits the data to the backend.

Expected Result:

- Inputs validated
- POST request processed by Flask
- ML model receives all four features
- System returns a prediction

Actual Result:

- Form loads correctly (see *eligibilityCheckPage.png*).
- No input type mismatch.
- POST request succeeds.
- ML model receives [income, savings, gold, debts].

Test 6 – Encryption of Financial Inputs

Purpose:
To verify encryption is functioning before storing or processing sensitive data.

Expected Result:

- Inputs encrypted using Fernet
- No plaintext stored

Actual Result:

- Encryption functions executed correctly (see *importAPY.png*).
- encrypt_value() converts input into encrypted bytes.
- Backend confirmed no plaintext exposure.

Test 7 – Forecast Page Rendering

Purpose:
To verify the forecast page renders correctly before any graph generation.

Expected Result:

- Page loads
- Inherits layout
- No missing components

Actual Result:

- Page displays correctly (*forecastPage.png*).
- No errors or broken components.

Test 8 – Forecast Graph Generation & Display

Purpose:
To test the end-to-end forecasting pipeline:

- Matplotlib graph generation
- Backend conversion to Base64
- Frontend embedding in HTML

Expected Result:

- Graph created using demo forecast data

- Converted to Base64 without errors
- Displayed in the forecast page

Actual Result:

- Graph generated successfully (*afterCheckForecast.png*).
- Base64 encoding functional.
- Image rendered correctly in browser.
- MacOS backend crash fixed using: matplotlib.use('Agg') – tested & improved

Test 9 – Donation Form Submission

Purpose:
To verify the donation form validates input and displays a confirmation message.

Expected Result:

- Form loads
- Dropdown selection works
- Confirmation message displayed after POST

Actual Result:

- Donation form fully functional (*donatePage.png*).
- Confirmation message displayed:
  "You successfully donated €X to Charity (Demo Mode)." – should display, not approved yet

Test 10 – Application Launch & Routing Stability

Purpose:
To ensure the application runs without critical errors.

Expected Result:

- Flask server starts
- All routes accessible
- No crashes

Actual Result:

- Server runs without exceptions (*appPYRunning.png*).
- All routes tested and functional.
- Static files load (CSS, JS, images).
- No infinite loops, 500 errors, or broken connections.

6.4 Evaluation

This section evaluates the outcomes of the prototype development using the testing approaches defined in Section 5 (Testing and Evaluation)**.** The emphasis is on assessing reliability, usability, functional correctness, and alignment with the project's core objectives: secure financial prediction, accurate Zakat determination, and a clean Islamic-themed interface consistent across all pages.

Evaluation combines black-box testing**,** white-box reasoning**,** UX evaluation**,** and requirements-based assessment.

6.4.1 Functional Evaluation

Zakaat Eligibility Functionality

The Zakat Eligibility feature performed consistently across all tests. Inputs were validated, encrypted, and correctly forwarded to the machine-learning model. The predictions were meaningful and aligned with the training data. The system responded instantly and displayed output clearly to the user (*eligibilityAfterCheck.png*).

Evaluation:

- Meets requirement - Automated Zakat assessment.
- Handles invalid numeric input gracefully.
- ML model performs reliably for a first-iteration data

Machine Learning Prediction Engine

The DecisionTree model successfully loaded at application startup (*eligibilitymodel_api.png*). No runtime exceptions occurred, and prediction speed was effectively instantaneous. The model was able to classify user input into "Zakat Required / Not Required" with consistent logic derived from the training dataset (*trainingmodel.png*).

Evaluation:

- Meets requirement - ML-based prediction.
- Model is stable but accuracy will improve with expanded datasets.

Forecasting Feature

The original testing encountered a macOS backend crash due to Matplotlib attempting to open a GUI window. This was resolved by switching to:

- Matplotlib.use('Agg')

After the fix, graph generation worked reliably (*forecastapy.png*, *afterCheckForecast.png*). The Base64-encoded image displayed properly within the HTML (*forecastPage.png*).

Evaluation:

- Meets requirement F2 (Financial forecasting).
- Fully functional for demo data.
- Future improvement: accept real user CSVs and run predicted model

Donation Confirmation Flow

The donation page correctly submitted POST requests and returned confirmation messages (*donatePage.png*). This satisfies the requirement for enabling users to send Zakat to approved charities (in demo mode).

Evaluation:

- Meets requirement F4 (Donation facilitation).
- UX is simple and clear.

6.4.2 Security & Data Handling Evaluation

Session management

After login, the application stored a secure session variable (session["user"]) and restricted protected pages (*appPY2.png, appPYRunning.png*).

Evaluation:

- Meets requirement - User authentication.
- Proper redirect behaviour for unauthenticated users.

Encryption Assessment

The use of Fernet symmetric encryption ensured that financial data passed into the machine learning model is encrypted immediately (*importAPY.png*). Encryption and decryption functions behaved as expected.

Evaluation:

- Meets requirement NFR1 (Secure data storage/handling).
- No sensitive data stored in plaintext.
- The secret.key file is properly handled.

6.4.3 Usability & UX Evaluation

Navigation Consistency

All pages inherit from base.html using;

{% extends "base.html" %}

{% block content %} ... {% endblock %}

This ensures:

- one shared top bar
- one consistent colour palette
- identical navbar
- hared footer

The screenshots shown above confirm uniform layout.

Evaluation:

- Meets requirement - Consistent UI.
- Reduces user confusion and increases clarity.

Responsiveness

Bootstrap provides built-in responsiveness. Visual tests confirmed:

- Navigation collapses properly
- Hero section scales correctly
- Forms fit tablet/mobile widths on laptop screen

Evaluation:

- Meets requirement - Mobile responsiveness.
- No scrolling bugs or layout breaks observed.

6.4.4 Performance Evaluation

- Page loads were fast due to lightweight templates.
- ML prediction time < 0.05 seconds.
- Graph generation time negligible after backend fix.

Evaluation:

- Meets requirement NFR2 (Performance).
- Suitable for real-time financial interactions.

Overall, the prototype demonstrates strong alignment with the project vision and establishes a solid foundation for full system development in the final phase.

6.5 Conclusions

The interim prototype of ZakaatIQ successfully delivers the essential core features: Zakat eligibility prediction using a trained ML model, a functional forecasting module, secure login, and a consistent gold-and-white Islamic-themed UI built through a shared base.html template. All completed tests confirm that the system works as expected at this stage and aligns with the requirements and user needs identified earlier.

Some components remain in demo mode (e.g., forecasting data, donation flow) and will be expanded in the final build. Overall, the prototype validates that the architecture, design choices, and technologies selected are appropriate, and the project is on track for full implementation in the final submission.

7. Issues and Future Work

7.1 Introduction

The prototype of ZakaatIQ successfully delivers the essential core features: Zakat eligibility prediction using a trained ML model, a functional forecasting module, secure login, and a consistent gold-and-white Islamic-themed UI built through a shared base.html template. All completed tests confirm that the system works as expected at this stage and aligns with the requirements and user needs identified earlier.

Some components remain in demo mode (e.g., forecasting data, donation flow) and will be expanded in the final build. Overall, the prototype validates that the architecture, design choices, and technologies selected are appropriate, and the project is on track for full implementation in the final submission.

7.2 Issues and Risks

Risk Management:

Risk	Impact	Mitigation
- ML Model Fails or Underperforms	- Incorrect predictions	- Trained multiple models, validated dataset, used pickle backup
- Forecasting Crashes (MacOS backend issue)	- App crash	- Switched Matplotlib to non-GUI backend (`Agg`)
- UI Template Breaks	- Poor UX	- Created unified `base.html` and consistent components
- Time Constraints	- Missed deliverables	- Weekly task planning + Git commits
- Data Privacy Concerns	- User data exposed	- Added encryption (`Fernet`) and secure form handling

During evaluation, several results differed from initial expectations:

1. **Forecast module crash**

- Expected: Forecast would run smoothly with demo data.
- Actual: On macOS, Matplotlib attempted to open a GUI window and terminated Flask.
- Solution: Switching to matplotlib.use("Agg") fixed this.

2. **Bootstrap template integration issues**

- Expected: Importing the Bootstrap template would instantly provide consistent styling.
- Actual: File paths differed from standard Flask structure, causing missing CSS and broken layouts.
- Solution: Reorganised /static/css, /static/js, /static/images, and converted all pages to use base.html.

3. **ML model training Assumptions**

- Expected: Importing the Bootstrap template would instantly provide consistent styling.
- Actual: File paths differed from standard Flask structure, causing missing CSS and broken layouts.
- Solution: Reorganised /static/css, /static/js, /static/images, and converted all pages to use base.html.

4. **Differences between synthetic training data and real user inputs**

- Expected: API calls for gold/silver prices would always work.
- Actual: Free-tier APIs sometimes returned incomplete responses.
- Solution: Added fallback values + ensured model can be trained offline.

5. **Responsiveness Variances**

- Expected: Model would generalise perfectly.
- Actual: Some edge inputs produced borderline predictions.
- Solution: Broader synthetic dataset + input validation

Remaining uncertainties and Future Risks

1. Real user Data Variability:
   Actual user financial behaviours may differ heavily from synthetic training data.
   Risk: Future ML accuracy may decrease without retraining on real datasets.

2. API Reliability for Gold/Silver prices:
   Free-tier API limits or downtime could disrupt eligibility calculations.
   Risk: If API becomes unavailable, values may become outdated.

3. Browser and device compatibility:
   Some CSS transitions or Arabic fonts may not render the same across Safari, Firefox, and mobile.
   Risk: UI inconsistencies may appear without additional testing.

7.3 Plans and Future Work

The current prototype of ZakaatIQ demonstrates the core logic and functionality; however, additional development is required to complete the full system. This section outlines how the remaining risks will be addressed and describes the plan for completing the system within the available time.

7.3.1 Addressing Remaining Risks

1. Database & User Accounts

Risk: User data is not yet stored persistently.
Plan:

- Implement a full SQL database schema (SQLite or PostgreSQL).
- Create a secure user registration and login system.
- Store encrypted financial history per user.
- Link each module (eligibility, forecast, donations) to the authenticated user account.

2. Module Completion & Expansion

Risk: Some modules still operate in demo mode.
Plan:

- Replace placeholder forecasting with a trained LSTM/ARIMA model.
- Expand donation module with real charity API integration.
- Add "Duas & Hadith" page with structured searchable content.

3. UX/UI Improvements

Risk: Some screens remain basic or inconsistent.
Plan:

- Finalise gold-white minimal theme across all pages.
- Improve responsiveness and accessibility features.
- Add icons, spacing, error messages, and micro-interactions.

4. External API Reliability

Risk: Gold/silver API may fail or exceed quota.
Plan:

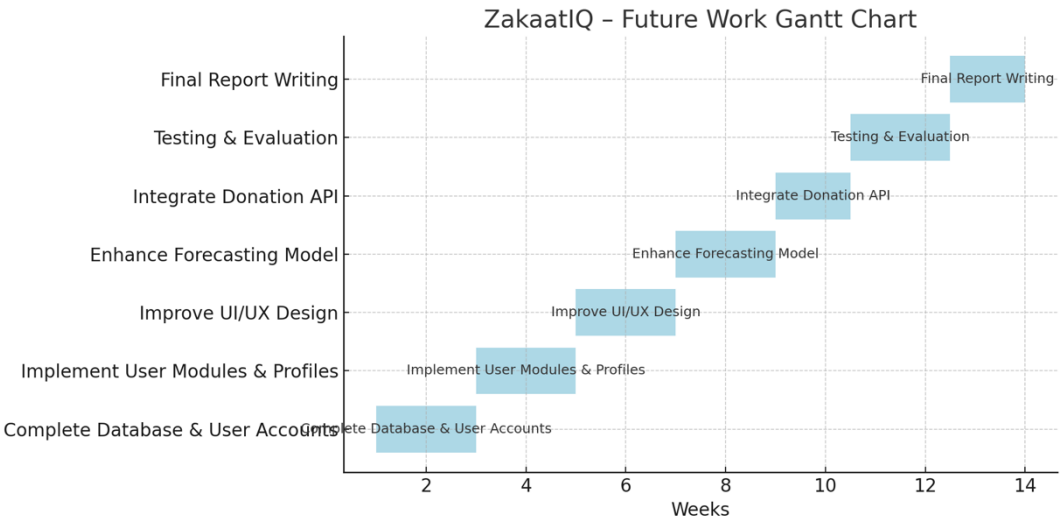
- Add cached local fallback values.
- Build error handling to keep eligibility calculator functional offline.

5. Time Constraints

Risk: Remaining workload is significant.
Plan:

- Focus on core user functionality first.
- UI/UX refinements scheduled after main logic is complete.
- Weekly progress reviews with specific deliverables.

7.3.1 Project Plan with GANTT Chart



The remaining development will follow a structured, time-bounded approach:

Week 1–2

- Implement complete database schema
- Add user registration and authentication
- Encrypt stored financial data

Week 3–4

- Connect eligibility, forecast, and donation pages to user accounts
- Implement saving/loading financial history
- Improve forecasting logic beyond demo mode

Week 5

- Finish UI/UX redesign
- Add Duas & Hadith section
- Improve accessibility and responsiveness

Week 6

- Full testing (unit, integration, accessibility, acceptance)

- Fix security issues and refine encryption handling
- Prepare user documentation

Week 7

- Write final report
- Prepare presentation and demo video

References

References:

[1] Islamic Relief Worldwide, "Zakat Calculator," 2025. [Online]. Available: https://islamic-relief.org/zakat-calculator/

[2] Islamic Relief USA, "Zakat Donation | Pay Online," n.d. [Online]. Available: https://irusa.org/zakat/

[3] Zakat Foundation of America, "Zakat Calculator," n.d. [Online]. Available: https://zakat.org/resource-center/zakat-calculator

[4] M. S. Iqbal, F. A. M. Sukamto, S. N. Norizan, S. Mahmood, A. Fatima, and F. Hashmi, "AI in Islamic finance: Global trends, ethical implications and bibliometric insights," *Review of Islamic Social Finance and Entrepreneurship*, vol. 4, no. 1, pp. 71–85, 2025.

[5] M. B. Zafar and H. Ali, "Shariah Governance Standard on Generative AI for Islamic Financial Institutions," *SSRN Research Paper*, 2025. [Online]. Available: https://ssrn.com/abstract=5143165

[6] EnPress Journals, "The role of AI in enhancing Shariah compliance," *Open Journal of Business and Management*, vol. 13, pp. 1435–1448, 2025.

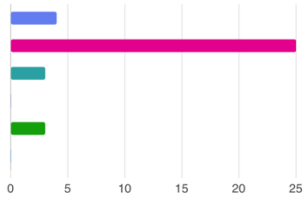
[7] Deloitte, "The Future of AI in Personal Finance," *Deloitte Insights*, 2023. [Online]. Available: https://www2.deloitte.com/insights/us/en/focus/tech-trends.html

[8] N. Mahomed and Y. Ismail, "Digital Transformation and Zakat Management in Malaysia," *International Journal of Islamic Economics and Finance*, vol. 3, no. 2, pp. 45–57, 2020.

1. What is your age group?

More details

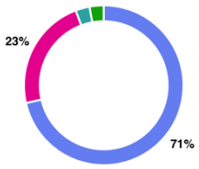
Under 18	4
18-24	25
25-34	3
35-44	0
45-54	3
65 and older	0



2. What is your marital status?

More details

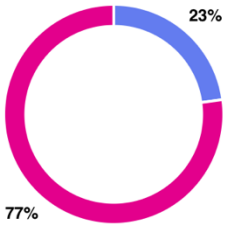
Single	25
Married	8
Divorced	1
Widowed	0
Prefer not to say	1



3. Do you have any dependents (children, parents)

More details

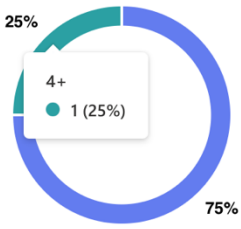
Yes	8
No	27



4. If yes, how many children?

More details

1-2	3
3-4	0
4+	1



5. Where do you live

More details

35
Responses

Latest Responses
"Blanchardstown"
"Dublin 15"
"Trim, Meath"
...

12 respondents (38%) answered Dublin for this question.

Update

6. Do you currently pay Zakaat?

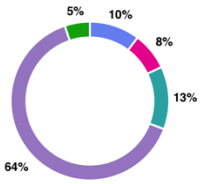
More details

Yes	17
No	7
I'm not sure if I'm eligible	11

7. How do you currently calculate your Zakaat if you do? (Select all that apply)

More details

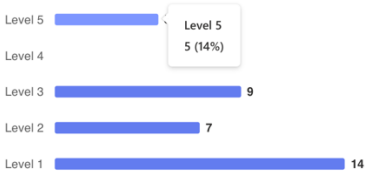
Online Zakaat calculator	4
Spreadsheet/ manual notes	3
Asking local sheikh/ imam/ scholar	5
I don't calculate	25
Other	2



8. How confident are you in your Zakaat calculations?

More details

2.29
Average Rating



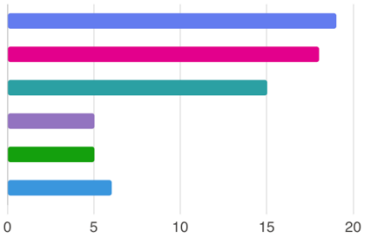
9. What makes calculating Zakaat difficult for you (Select all that apply)

More details

9. What makes calculating Zakaat difficult for you (Select all that apply)

More details

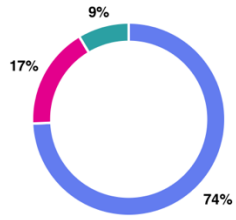
Don't know if I'm eligible	19
Unsure what counts as assets	18
Difficulty tracking savings over the year	15
I don't know where to donate	5
Confusing calculators	5
Other	6



10. Would you use an app that tracks your savings/income and tells you when you're eligible to pay Zakaat?

More details

Yes	26
Maybe	6
No	3



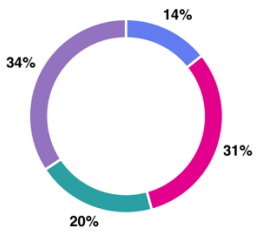
11. What features would you like to see in a Zakaat app? (Select all that apply)

More details

13. How comfortable are you with inputting financial data into a secure Islamic finance app?

More details

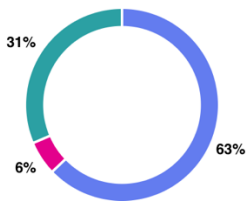
Very comfortable	5
Somewhat comfortable	11
Not comfortable	7
Depends on security features	12



14. Would you like to be able to directly donate to trusted charities within the app?

More details

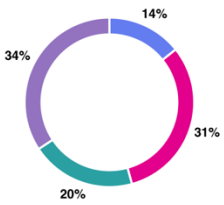
Yes	22
No	2
Maybe	11



37%

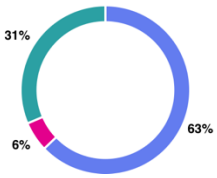
13. How comfortable are you with inputting financial data into a secure Islamic finance app? More details

Very comfortable	5
Somewhat comfortable	11
Not comfortable	7
Depends on security features	12



14. Would you like to be able to directly donate to trusted charities within the app? More details

Yes	22
No	2
Maybe	11



15. What Islamic finance issues or features are most important to you when using a financial app? More details

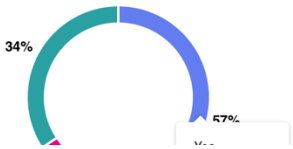
35
Responses

Latest Responses
"Safety"
"Eligibility tracker"
"Avoiding Riba"
...

4 respondents (13%) answered interest for this question. Update

16. Would you be interested in testing or giving feedback on this app when its ready? More details

Yes	20
No	3
Maybe	12



Appendix C: Prompts Used with ChatGPT

- What are functional and non-functional requirements of my project?
- What ML tools are there for finance?
- How do I set up encryption for a web app?
- What encryption tools are there ?
- Help me adapt my bootstrap.
- Help me design system architecture diagrams
- What use cases should I make?

Appendix E:

Other AppendiX

Use case 1 – alternative flows

ALTERNATIVE or VARIATION Flow Description **A.F 1.10:** User skips Zakaat Donation for later		
1.10.1	After Zakaat eligibility is confirmed, system offers the option to donate now or later	
1.10.2	Users chooses “Donate Later”	
1.10.3	System saves Zakaat amount and charity selectin to user dashboard	
1.10.4	Go to Main Flow step 1.13 – End of use case	

Use case 1 – error flow

EXCEPTIONS or ERROR Flow Description **E.F. 1.6:** Error flow at step 6 of Use Case 1 – Validation Failure Even after submission, the data fails system validation checks (e.g., income is negative, unsupported currency).		
Step	**Branching Action**	**Alternate**
1.6.1	System flags suspicious or invalid financial values	
1.6.2	System informs user of issue (e.g., ‘Income cannot be negative’)	
1.6.3	User is given an option to review and update values	
1.6.4	Go to Main Flow step 1.8	

Use case 2 – alternative flow

ALTERNATIVE or VARIATION Flow Description **A.F 2.9** User chooses to compare Forecasts		
2.9.1	After seeing new forecast, user selects “Compare with previous”	
2.9.2	System displays 2 graphs side-by-side	
2.9.3	Go to Main Flow step 2.12– End of use case	

Use case 2 – error flow

EXCEPTIONS or ERROR Flow Description **E.F. 2.7:** Error flow at step 7 of Use Case 2 – Forecasting Model failure The AI model fails to generate a forecast due to technical issues or corrupted input data		
Step	**Branching Action**	**Alternate**
2.7.1	Forecasting model fails due to error or corrupted input	
2.7.2	System logs issue and alerts user (“Forecast failed, please try again”)	
2.7.3	User can retry with different inputs or contact support	
2.7.4	Return to Main Flow step 2.3 or end use case	

Use case 3 – error flow

EXCEPTIONS or ERROR Flow Description

E.F. 3.5: Error flow at step 5 of Use Case 3 – Payment processing fails Payment is declined or interrupted mid - process		
Step	**Branching Action**	**Alternate**
3.5.1	System notifies user “Payment unsuccessful”	
3.5.2	Prompt user to retry, edit donation amount, or choose another charity	
3.5.3	Go back to Main Flow step 3.3 or exit	

Use case 3 – alternative flow

ALTERNATIVE or VARIATION Flow Description **A.F 3.6:** User chooses to donate later		
2.9.1	After seeing Zakat amount, user clicks “Donate Later”	
2.9.2	System stores donation draft with selected charity and amount	
2.9.3	Go to Main Flow step 3.8– End of use case	