DIT826 – Software Engineering for Data-Intensive AI Applications

Team 6

Project Title: *AILoan*

Members:

Akuen Akoi Deng
Cynthia Tarwireyi
Kanokwan Haesatith
Nazli Moghaddam

Table Of Contents

Individual contributions

Individual contribution in Team Work

The contribution of each team member to the overall work in the delivered artefact is stated in the table below (one row per team member). The form is signed by all team members

Individual Name	Description of Contribution	Signature
Akuien Akoi Deng	Login and sign up Welcoming page Setting up CI/CD pipeline Model training Model Evaluation Model performance optimization Project containerisation(docker) Explainable AI Testing System deployment(kubernetes) Documentation Training possible extension models	AK
Cynthia Tarwireyi	• ***ML Model*** **Data Cleaning:** Explore the dataset and perform feature engineering **ML Model Definition:** Determine which classification model to use and train the classification model. **ML Model Versioning:** Create different ML versions and admin can select a model version to use for prediction and evaluation. **Model Evaluation:** Analyse and optimise the model performance. **ML Model deployment:** Deploy the models on the web-server and use them for prediction. • ***Admin (Frontend and Backend)*** **Prediction Page:** Admins can upload a CSV file and get loan predictions **Report Page:** Admins can see statistical information about the loan applicants such as the approval rate **Information Page:** A page on the admin UI where admins can read about information on what the system is about. **Performance page:** Implementing the representation of the model evaluation • ***User (Frontend and Backend)*** **Loan Application Page:** Created a user interface where the users can fill in and submit a loan application and receive instant prediction results. The application is also saved to the database. **About Us Page:** A page on the user UI where users can read about	CT

	information on what the system offers. **Welcome Page (only frontend):** Beautifying the welcome page UI	
Kanokwan Haesatith	**Registration/Login :** create a role-based signup/login system where user will be taken directly to the correct dashboard based on the role they registered with **Training data :** Maintain and manage the training data of the system in the database including new data. **System UI:** create the dashboard for both user and admin side according to the design,its respective menu bar and the overall look . **Chat assistance :** create the system AI chat assistant to help users with questions during the application process. **Notification :** create notification system on the admin side where it notifies the admin when there are new loan applications from users to be reviewed. **Application page:** create an application page where admin can update the status of each application from pending to reviewed Documentation: created the system sitemap diagram, use case diagram, ER diagram together with documentation in the project.	HK.
Nazli Moghaddam	**Login page:** Implemented the initial login functionality without considering the roles of the users.Using the Django's built-in functionality for authentication considering error-handling **Reset password:** Apply the reset password providing tokens to the users to create a new password **Performance page:** Implementing the representation of the model evaluation **Explainable AI:** Using shap library and negative and positive effect of each element to represent the evaluation of the submitted loan application by a specific user. **Unit tests:** writing unit tests for important functionalities in the system	NM

Section 1. The concept of the AI system

Author: Akuien

1.1. System Description

The loan prediction system tackles the difficulties encountered by many financial institutions in efficiently evaluating and assessing a large amount of loan applications. Conventional approaches often involve time-consuming manual assessments which lead to a large backlog of pending applications causing delays and occasional bias, resulting in inconsistent decisions.

Our solution to this problem is to improve the process by using artificial intelligence. This involves using AI to predict the likelihood of loan approval based on previous customer behaviour and historical data. Using individual details such as the income, age, loan amount, credit score and debt to income ratio, the AI model is able to predict the loan legibility for each application and provide a much quicker and reliable decision.

1.2. Goals and objectives

- ★ Automate the loan approval process to improve efficiency and reduce manual workload.
- ★ Ensure unbiased lending practices by using historical data to ensure objectivity.
- ★ Evaluate and minimise the risk of financial losses for financial institutions by accurately predicting the likelihood of a customer not being able to repay the loan.
- ★ Provide clear and interpretable explanations for loan approval decisions to both customers and bank employees (Explainable AI)
- ★ Use machine learning/deep learning algorithms to improve the precision of loan approval predictions.
- ★ Minimise false positives and false negatives.
- ★ Enhance customer experience by providing quick and unbiased loan decisions.

1.3. System requirements and Usage Scenarios

1.1. The system shall allow the creation of new users and admins.

1.2. The system shall provide an option for registered users and admins to log into the system.

1.3. The system shall allow users and admin to retrieve a forgotten password.

1.4. The system shall allow users to permanently delete their account.

1.3.1. User Requirements

Role 1 : Bank's Customer

1.5. The system shall allow users to view and edit their account details.

- ❖ As a user, I want the ability to view and edit my account details to ensure that my personal information is up-to-date and accurate.

1.6. The system shall contain a chat box providing instant help for common queries related to loan applications.

- ❖ As a user, I want access to a chat box providing instant help for common queries related to loan applications, ensuring a seamless and user-friendly experience.

1.7. The system shall allow users to input their details and receive a loan approval predictions

- ❖ As a loan applicant, I want to be able to submit personal and financial details through the app for the AI system to assess eligibility.

1.8. The system shall allow the user to input their desired loan amount.

1.9. The system shall provide users with an instant loan prediction result.

- ❖ As a loan applicant, I want to receive quick predictions on my loan application

1.10. The system shall provide the user with an explanation why their application was rejected.

- ❖ As a user whose loan application has been rejected, I want to receive recommendations on how I can enhance my loan eligibility for future applications.

1.11. The system shall provide users with rejected applications recommendations on how they can enhance loan eligibility.

1.3.2. Admin Requirements

Role 2 : Admin

1.12. The system shall allow an admin to upload a csv file of customer data and get loan predictions.

- ❖ As an admin, I want the ability to upload a CSV file containing customer data, so that I can obtain loan predictions for the provided dataset efficiently.

1.13. The system shall visualise the prediction results in table form and as a pie chart.

- ❖ As an admin, I want the system to present the predictions both in a table format for detailed information and as a pie chart for a quick and intuitive overview.

1.14. The system shall provide the admin with loan approval statistical reports.

- ❖ As an admin, I want the system to generate reports regarding approval rates, rejection rates, so that I can assess how efficiently the loan applications are processed.

1.15. The system shall allow for the admin to retrain a model

- ❖ As an admin, I want the ability to retrain the model to incorporate new data and improve its predictive capabilities.

1.16. The system shall allow the admin user to check the model evaluation metrics such as the confusion matrix and accuracy.

- ❖ As an admin, I want the ability to check and analyse model evaluation metrics, such as the confusion matrix and accuracy, to assess the performance of the current model.

1.17. The system shall notify the admin on new applications made by users.

- ❖ As an admin, I want to get the notifications, when new loan applicants are submitted by the customer, allowing me to stay informed about the system's activity.

1.18. The system shall show the admin a list of new applications made by users.

1.3.3. Non Functional Requirements

1. ***Performance:*** The system should use efficient algorithms and app design for optimal processing and response time
2. ***Accessibility:*** The system should be accessible through responsive web interface
3. ***Availability:*** The system should be deployed on a cloud based service to enable easy availability and access to the service.
4. ***Usability:*** The system should have friendly UI patterns and design to enable easy use and navigation for all users.

Section 2. Data sources

Author: Cynthia

2.1 Link to dataset

https://www.kaggle.com/datasets/nikhil1e9/loan-default/data

2.2. Dataset Description

The Loan Approval dataset we will be using is a comprehensive resource for analysing and predicting loan approval outcomes based on historical data. This dataset has a diverse range of loan application scenarios and customer profiles which is instrumental in leveraging past customer behaviours to anticipate potential loan defaulters.

This structured dataset is labelled, making it well-suited for supervised machine learning tasks. . It has 13 features with 252,000 data points, with each data point representing a unique loan application. This substantial size ensures robust model training and evaluation. The dataset includes both numerical and categorical variables. These features collectively capture crucial aspects of a loan application, ranging from the applicant's demographics to financial metrics. The features are as follows:

```
      LoanID  Age  Income  LoanAmount  CreditScore  MonthsEmployed  \
0  I38PQUQS96   56   85994       50587          520              80
1  HPSK72WA7R   69   50432      124440          458              15
2  C1OZ6DPJ8Y   46   84208      129188          451              26
3  V2KKSFM3UN   32   31713       44799          743               0
4  EY08JDHTZP   60   20437        9139          633               8

   NumCreditLines  InterestRate  LoanTerm  DTIRatio    Education  \
0               4         15.23        36      0.44   Bachelor's
1               1          4.81        60      0.68     Master's
2               3         21.17        24      0.31     Master's
3               3          7.07        24      0.23  High School
4               4          6.51        48      0.73   Bachelor's

  EmploymentType MaritalStatus HasMortgage HasDependents LoanPurpose  \
0      Full-time      Divorced         Yes           Yes       Other
1      Full-time       Married          No            No       Other
2     Unemployed      Divorced         Yes           Yes        Auto
3      Full-time       Married          No            No    Business
4     Unemployed      Divorced          No           Yes        Auto

  HasCoSigner  Default
0         Yes        0
1         Yes        0
2          No        1
3          No        0
4          No        0
```

Figure 2.1: Figure showing a description of the datasets features

From the dataset, it is evident that “Default” is the target feature, where approved loans are denoted by 0, and rejected ones are denoted by 1. Key features within the dataset include but are not limited to, applicant's income, loanAmount, creditScore and DITRation. These features collectively contribute to the predictive power of the dataset.

Moreover, the dataset boasts data integrity, free from missing or duplicate values, which minimises the need for extensive data cleaning. While categorical features are present, the preferred approach involves converting them into numerical representations using hot encoding, optimising the dataset for seamless integration into machine learning models.

Section 3. Features and inference model

Author: Cynthia

3.1. Data Validation and Cleaning

3.1.1 Data Cleaning

The initial clean process started with conducting an in-depth exploratory data analysis to get a better understanding about the dataset's structure and potential issues. We checked for missing values and duplicated rows in the dataset but they were not there. Furthermore, we performed exploratory Data Analysis by using a box plot to detect outliers in the numerical features. In addition, we used histograms as a visual inspection of data distributions which revealed that all the numerical features had uniform distribution. As the dataset originates from the USA and our intended system usage is in Sweden, we manually converted the currency for annual income and loan amount from USD to SEK by multiplying them by 10. Remarkably, the dataset was deemed inherently "clean," requiring minimal additional cleaning efforts due to its well-maintained state.

3.1.2 Feature Engineering

To minimise prediction variance, we conducted feature selection by using a RandomForestRegressor to evaluate the significance of each independent feature to the dependent (target) feature. We used one-hot encoding to ensure the validity of categorical variables. The model assigned importance scores to individual features based on their contributions. These scores, derived from the trained regressor model, as shown in the diagram below (figure 3.1), shows the influence of each feature on the predictions made by the RandomForestRegressor model. We set a cut-off value of 0.03, where we considered features with scores above this threshold to be more influential in shaping the model's output.

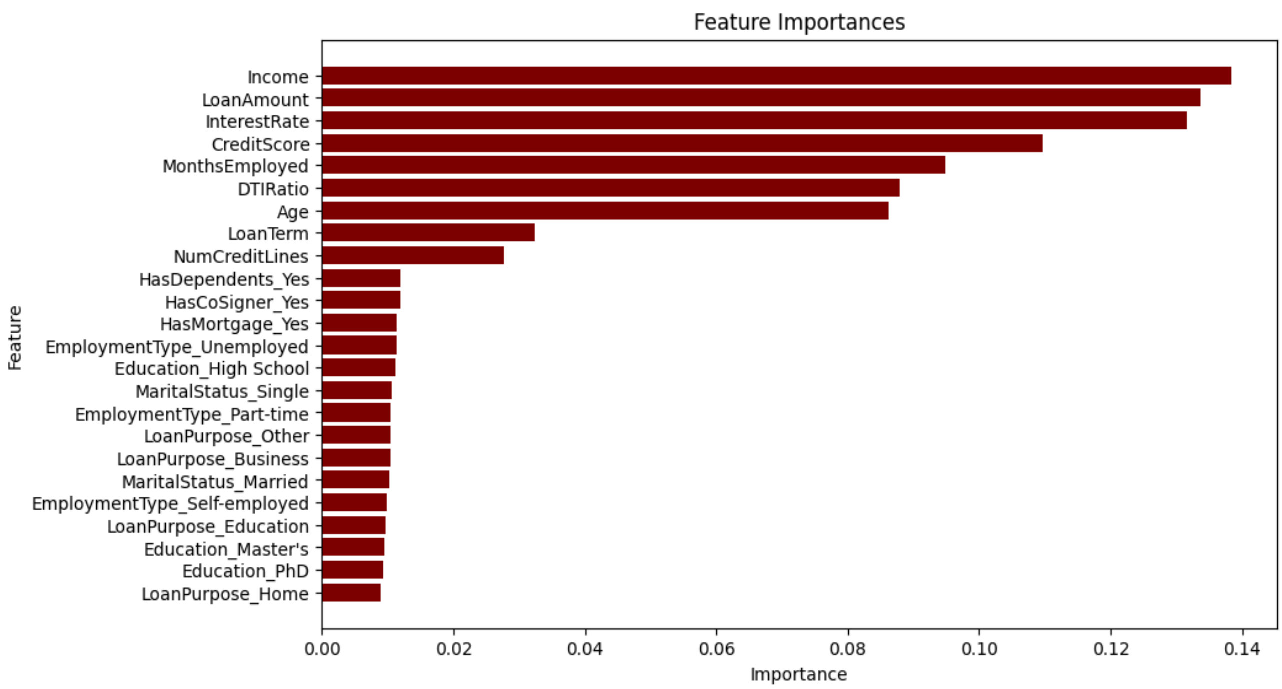


Figure 3.1. Graph showing the importance scores to individual features based on their contributions.

3.1.3. Target Feature Processing

While conducting exploratory data analysis (EDA) to get a better understanding of the features. We observed that the target variable exhibited a substantial class imbalance, with the rejected loan applications class being underrepresented. Recognizing the potential bias in model predictions that can be caused by this imbalance, we implemented an oversampling technique using Synthetic Minority Over-sampling Technique (SMOTE).

The application of SMOTE resulted in a balanced distribution of the target variable within the training set. This balanced dataset was then used to train the machine learning model, ensuring that both classes received equal consideration during the learning process. The figure below (figure 3.2 and 3.3)depicts the distribution of the target variable before and after applying oversampling.

Figure 3.2. Target variable distribution

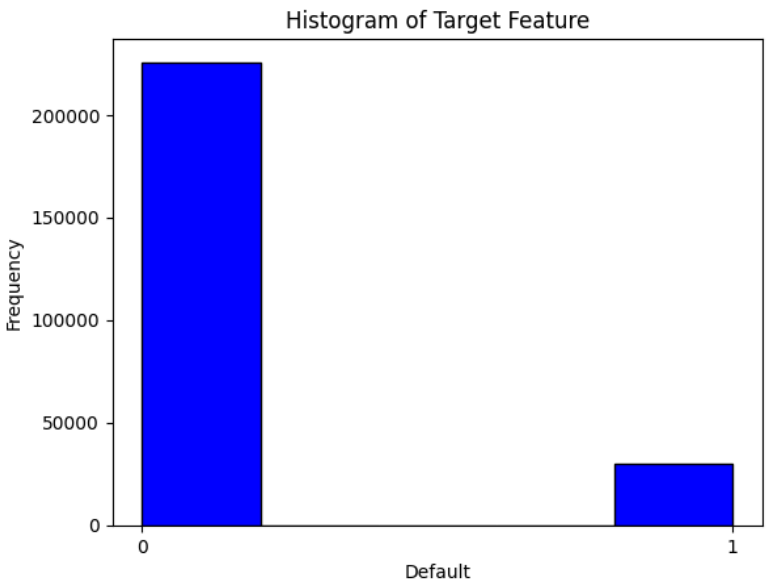
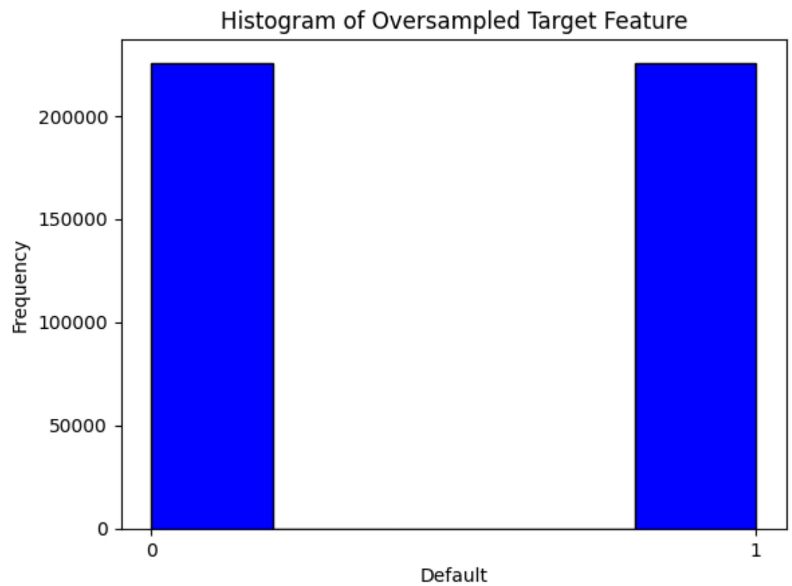


Figure 3.3 Target feature distribution after oversampling



3.2. Defining the Machine Learning Model

3.2.1 Choosing the Machine Learning Model

In the development of our loan prediction system, we decided to use a decision tree algorithm as our predictive model. Before settling on the decision tree model, we methodically trained several classification models to determine which performed best. During this evaluation, we considered metrics such as accuracy, precision, recall, and F1 score. The decision tree proved to be the model with the most predictive power, demonstrating its capacity to handle the nuances of the loan prediction task.

To validate the efficacy of oversampling, we trained the model using both the original dataset and the oversampled dataset. The model trained with the oversampled training set performed much better on both the training and unseen (testing) data. By generating synthetic samples, SMOTE facilitated more effective learning from the minority class, ultimately enhancing the model's ability to generalise to unseen data.

To further enhance the decision tree's performance, we conducted hyperparameter tuning. This strategic adjustment enabled us to systematically tune and explore various parameters, thereby improving the model's predictive accuracy. Additionally, we implemented feature scaling to standardise the numerical features in the dataset, ensuring uniform contribution of all variables to the decision tree's learning process. By integrating hyperparameter tuning and feature scaling into our model, we managed to enhance the decision tree's effectiveness.

3.2.2 Model Evaluation

In evaluating our machine learning model for loan prediction, we adopted a comprehensive approach, considering a diverse set of metrics to gain a thorough understanding of its performance. Given the sensitivity and consequential nature of predicting loan approvals, we analysed multiple evaluation metrics to address the potential risks associated with false positives and false negatives.

We used the accuracy score to provide an overall measure of the model's performance. Simultaneously, we considered the F1 score, which strikes a balance between precision and recall, offering a more nuanced assessment on the model's ability to handle both false positives and false negatives. Precision was crucial in determining the accuracy of positive predictions, particularly relevant in the context of avoiding approval of risky loans. However, acknowledging the importance of capturing all positive instances, we examined recall, emphasising the model's effectiveness in identifying applicants likely to repay their loans.

Given the significant repercussions of approving a loan to a risky applicant (related to precision), and the cost of missing out on creditworthy applicants (related to recall) , we sought a balanced evaluation. But ultimately our emphasis was on recall over precision due to the severe consequences associated with failing to identify applicants likely to repay their loans. In addition, recall supports the long-term profitability offer system by ensuring a larger pool of creditworthy customers.

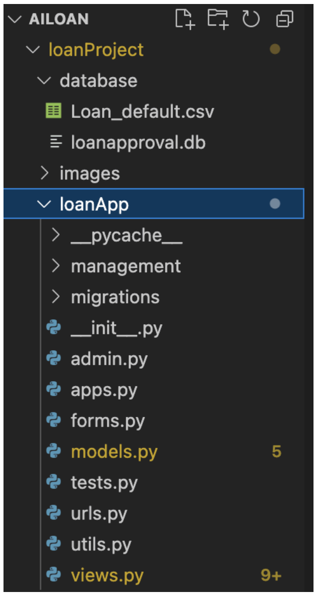
In addition, we incorporated the confusion matrix, which presented a detailed breakdown of true positives, true negatives, false positives, and false negatives, providing deeper insights into the model's strengths and areas for improvement. This multifaceted evaluation strategy allowed us to consider the trade-offs and make informed decisions based on the specific requirements and risks associated with loan prediction.

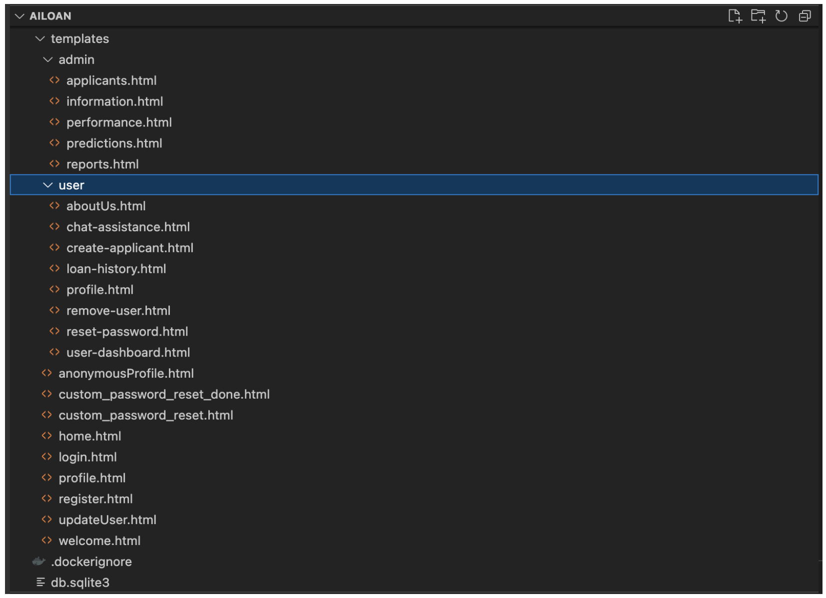
Section 4. Production system

Author: Nazli

4.1 Software components

In the loan application, we organised the application into distinct components to provide a clear structure. Each component plays an important role in the process of applying for a loan and getting clear predictions and analysis of the data. The most important parts of the loan application are the database, views, models, and templates which we will investigate thoroughly through the report.





Most of the functionalities of the app and the logic behind the web page are in ‘Views’. They process user requests, interact with existing models and render templates to display information. ‘Views’ handles tasks such as user authentication, loan application process, and administrative functionalities.

4.1.1 Examples of handling loan application in views

Most of the functionalities of the app and the logic behind the web page are in ‘Views’. They process user requests, interact with existing models and render templates to display information. ‘Views’ handles tasks such as user authentication, loan application process, and administrative functionalities.

The example below demonstrates the handling of loan application form, involving form validation, prediction using a machine learning model, and database interaction. We applied a context dictionary to pass the data to the template for rendering.

```python
@login_required
def create_applicant(request):
    context = {'messages': []}
    if request.method == 'POST':
        form = LoanForm(request.POST)

        if form.is_valid():
            Age = form.cleaned_data['Age']
            Income_SEK = form.cleaned_data['Income']
            LoanAmount_SEK = form.cleaned_data['LoanAmount']
            CreditScore = form.cleaned_data['CreditScore']
            MonthsEmployed = form.cleaned_data['MonthsEmployed']
            LoanTerm = form.cleaned_data['LoanTerm']
            DTIRatio = form.cleaned_data['DTIRatio']

            manual_exchange_rate = 10
            Income_USD = Income_SEK / manual_exchange_rate
            LoanAmount_USD = LoanAmount_SEK / manual_exchange_rate

            model_path = 'MLmodels/model_V3.joblib'
            model = load_model(model_path)
            prediction = model.predict([[Age, Income_USD, LoanAmount_USD, CreditScore, MonthsEmployed, LoanTerm, DTIRatio]])
            print("Prediction Result:", prediction[0])
```

4.1.2 Models

Models represent the data structure of the application.I have models defined for user data, loan applicants, and new loan applications. These models define the fields, relationships, and behaviours of the database tables.

Example of the models we have:

```python
class UserDetail(models.Model):
    userID = models.AutoField(primary_key=True)
    firstName = models.CharField(max_length=30)
    lastName = models.CharField(max_length=30)
    job = models.CharField(max_length=30, null=True)
    salary = models.FloatField(null=True)
    image = models.FileField(upload_to='profile', null=True, blank=True)
    date_of_birth = models.DateField(null=True, blank=True)

    def __str__(self):
        return self.firstName + " " + self.lastName


class LoanApplicant(models.Model):
    LoanID = models.CharField(max_length=12, primary_key=True, default=shortuuid.uuid)
    Age = models.IntegerField()
    Income = models.FloatField()
    LoanAmount = models.FloatField()
    CreditScore = models.IntegerField()
    MonthsEmployed = models.IntegerField()
    LoanTerm = models.IntegerField()
    DTIRatio = models.FloatField()
    Default = models.IntegerField(null=True, blank=True)
```

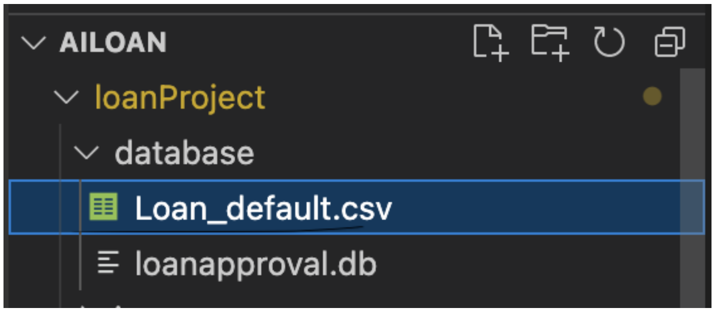
4.1.3 Templates

Templates are responsible for generating HTML in our project. The data that is being fetched from views and models should be represented in the user interface for various pages ensuring the consistent and appealing design.

4.1.4 Database structure

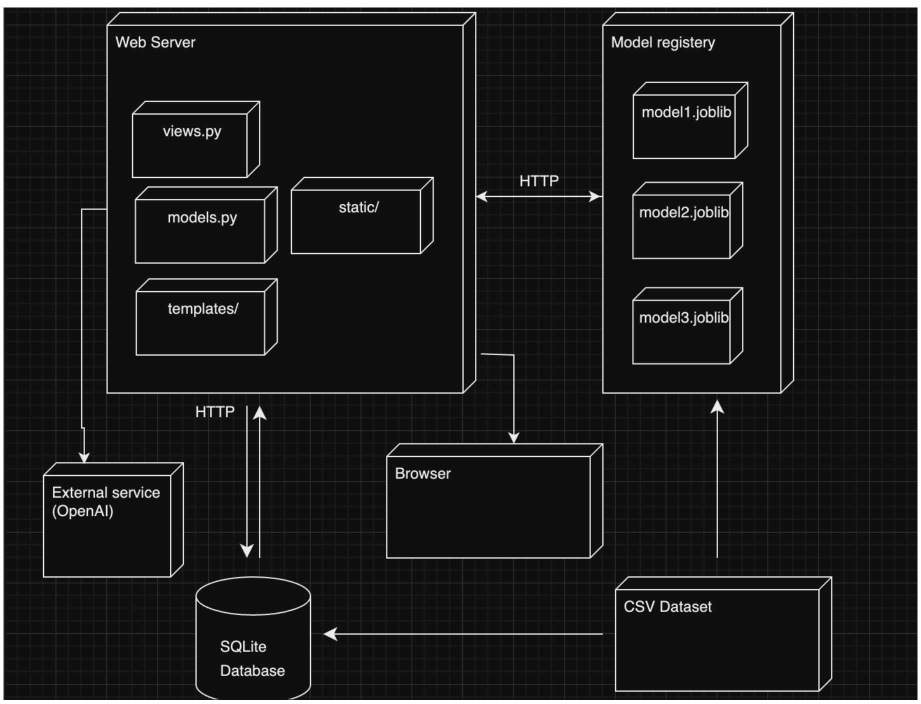
The database component in our system, stores and retrieves data. We've incorporated a CSV file named Loan_default.csv to initialise and populate the database with initial data. This file is a

data source for many aspects of the application.The database includes the tables corresponding to each model (CustomUser, LoanApplicant, NewLoanApplicant).
The relationships between the tables are established through foreign keys and there are primary keys(AutoField, CharField) that are uniquely identified for each record.



4.2 System architecture and design

This deployment diagram reflects a web-based application architecture with a Django backend that incorporates machine learning and external services to provide various functionalities.



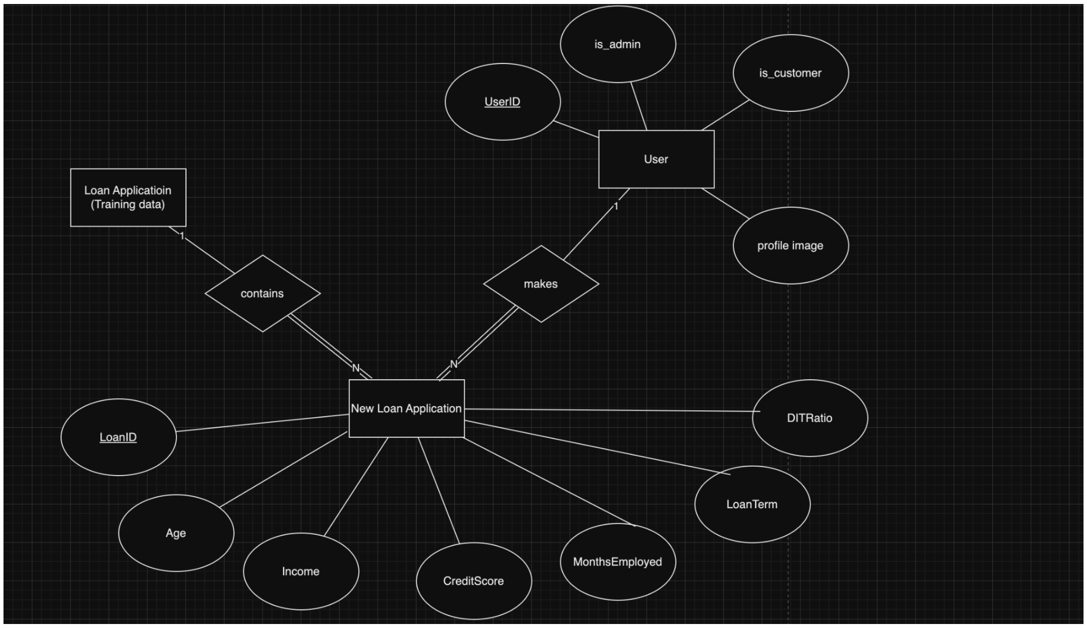
Model-View-Controller (MVC)

We follow the data and database interactions, Views handle user interface,user input and controllers to manage the flow of the data between models and views.

Microservice architecture

Services like OpenAI for chat assistance are external to the main Django application. Each of the features of this project, including chat assistance, user authentication, loan application processing, is independent of the others which provides a micro service architecture that can communicate with each other with well-defined APIs.

ER Diagram:



The above ER diagram shows the data model of the system. By looking at this, we can understand the structure of the data and how different entities are related.

4.3 The overall structure of the code

4.3.1 User authentication

The code that we provided for the login function handles the user authentication using Django's built-in authentication system. It includes views for user registration, login, logout, and profile page.User authentication is checked using the @login_required for certain views that require an authenticated user.

User roles, such as admin and customer determine access permissions and indicate the next redirection to either user dashboard or admin dashboard.

4.3.2 Loan application process for customer

Customers can submit loan applications through the creat_applicant views, which uses the LoanForm. These applications are processed using a machine learning model that the team integrated into the system. Its name is `MLmodels/model_V3.joblib.` SHAP values are computed for each application using the shap library to explain the model predictions.

4.3.3 Chat assistance
Customers can get some suggestions and chat-based assistance through the chat_assistance view functionality which uses OpenAI GPT-3.5 Turbo model.

4.3.4 Admin dashboard
Admins can view and manage loan applicants through the ‘applicants’ view.
In the ‘prediction’ view, loan applications can be processed and a CSV file can be uploaded for bulk predictions.
For evaluating the applications and model’s performance, the system has a ‘performance’ view and the admin can also see the reports on loan approval/rejection rates and feature statistics which are available through the ‘reports’ view.

4.3.5 Profile management
Authenticated users can access a personalised dashboard which they can view and update their profile information.

4.3.6 File handling
Some parts of the code require efficient file handling. This file handling requires users uploading files such as profile images and CSV files to apply predictions based on the models that are being integrated in the system.

4.3.7 Data visualisation
Plotly is used in the code for creating pie charts and other visualisations to display predictions and application statistics.

4.4 Testing Infrastructure (Unit testing)
The unit tests provide comprehensive coverage of critical features in the application view in the Django application, varying from basic views like the home page to more complex functionalities such as model evaluation, user registration and authentication, and external API integration for chat assistance.

```python
class RegisterViewTest(TestCase):
    def setUp(self):
        self.client = Client()

    def test_register_view_success(self):
        # Define test data
        user_data = {
            'username': 'testuser',
            'password1': 'testpassword123',
            'password2': 'testpassword123',
        }


        response = self.client.post(reverse('register'), data=user_data, follow=True)


        user_exists = CustomUser.objects.filter(username='testuser').exists()
        print(f"User exists in the database: {user_exists}")


        print(f"Response status code: {response.status_code}")


        print(f"User authenticated: {response.context['user'].is_authenticated}")

        if response.status_code == 302:

            print(f"Redirect URL: {response.url}")


            self.assertRedirects(response, reverse('login'), status_code=302)
        else:

            print("No redirect URL available for non-302 status code")
```

We applied mocking techniques which enhance the test isolation and examination of specific functionalities without reliance on external services. By performing these tests, we can ensure the reliability of the application's core features.

Example of how we applied the testing in our system is shown in the code above. The test case simulates a user registration attempt, checks the database for the user, checks the response and verifies if the registration was successful. Most of the test cases that we applied provide assertion methods like 'asseertRedirects' for convenient testing via the system.
The tests demonstrate the use of Django's testing tools to validate the behaviour of the registration view.

4.5 External libraries

- Plotly: Plotly is a data visualisation library that provides dynamic plots. In this project, Plotly is used to create histograms for visualising loan approval rates.
- Scikit-Learn: This is a machine learning library for python that provides simple and efficient tools for data analysis and modelling. Scikit-Learn is used for preprocessing data and applying machine learning models, including label encoding and calculating accuracy scores and confusion matrices.
- OpenAI GPT-3.5 Turbo: This is a language model developed by OpenAI and we employed this for chat assistance. The 'openai' library is used to interact with the OpenAI GPT-3.5 Turbo API
- Allauth: Allauth is a Django package that provides a set of views, forms, and templates for handling user authentication, including registration, login, password reset functionality.
- Shap: To explain the output of the machine learning models, we use the SHAP library. This library helps us to understand the overall behaviour of a model across the dataset of the project. Based on the predictions it gives us the positive and negative shap values.

These values show us how the person is eligible for the loan and helps the system to suggest to the customers how they can be more eligible for getting a loan.

- ❖ Seaborn: To provide a high-level interface for statistics in the project the team used seaborn. Seaborn simplifies the process of creating complex visualisations, and it provides a variety of colour pallets.
- ❖ Keras: The team used Keras library to apply neural networks APIs written in python.

4.6 Risk Mitigation

Model evaluation:

We applied model evaluation in the context of risk mitigation which is crucial for assessing the performance and reliability of a predictive model, especially when dealing with applications such as loan approvals. For example in this evaluation, we investigate specific cases of misclassification (e.g., false positives/negatives) that allows for a deeper understanding of the model's weaknesses and areas for improvement. We present the calculation of accuracy score which provides an overall measure of how well the model is performing.

Unit Testing:

The unit tests we implemented in our project ensures that individual components of the application function correctly and we can make sure that we can rely on the code.

There were some problems regarding the urls and rendering that the team noticed in the process of writing unit tests for the functions in the view of the project.

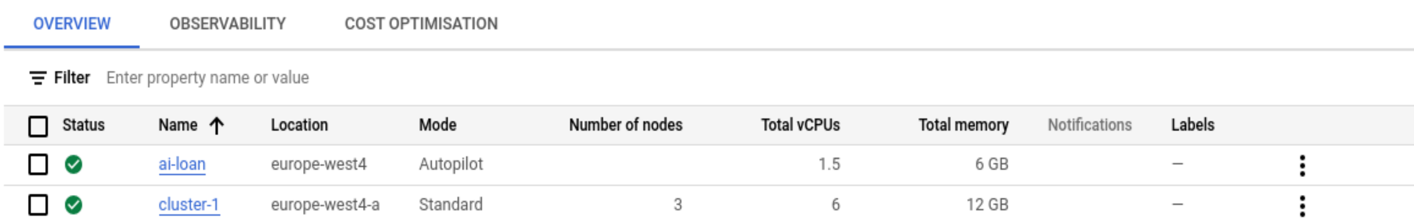
While making changes to the system, the team had to make sure that the functionalities work as the previous steps and before making the changes. These tests helped us to change the code with confidence, knowing that if tests pass, the system is still functioning as expected.

Deployment and containerisation via docker: ***Author: Nazli & Akuien***

In the process of deployment, we used docker which can contribute to risk mitigation in so many ways. It can improve the reliability, scalability, and security of the application. The team decided to implement the deployment using dockers because of isolation and the consistency that it provides to the system by providing isolation for the application and the dependencies that it has.

According to the lectures, it was suggested to apply the cloud-based deployment, however we didn't apply that method for the deployment purposes. What we could do in the process of deployment was to use both docker and cloud-based deployment together to leverage the benefits of both technologies.

Using Google Cloud Engine(GKE) we were able to create a kubernetes cluster with a single node. The cluster is set in autopilot mode making the allocation of the cloud resources adjustable automatically based on the workload of our system.
The cloud based cluster(ai-loan) can be seen as shown in the image attached below.

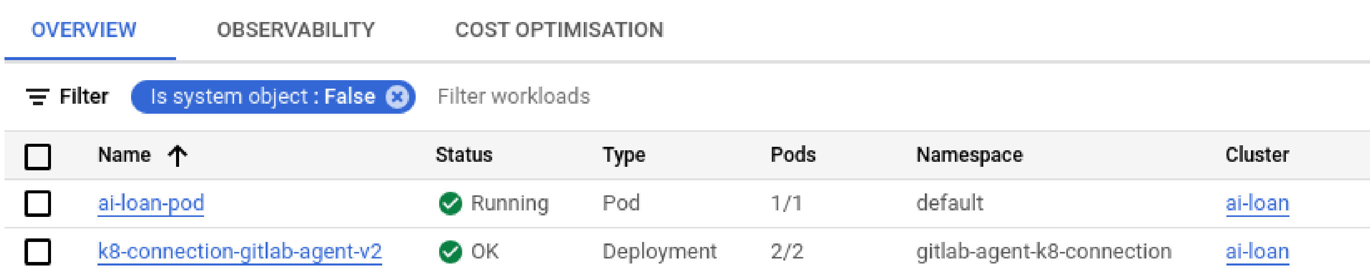

OVERVIEW OBSERVABILITY COST OPTIMISATION

Filter Enter property name or value

Status	Name ↑	Location	Mode	Number of nodes	Total vCPUs	Total memory	Notifications	Labels
✔	ai-loan	europe-west4	Autopilot		1.5	6 GB		–
✔	cluster-1	europe-west4-a	Standard	3	6	12 GB		–

Inside our cluster node we created a pod for our container image of the ailoan system from the gitlab container registry. We connected the cluster with gitlab to enable the pulling of the container image.

Below we can see the pod and the deployment connection between the cluster and gitlab using an agent and helm connection.


OVERVIEW OBSERVABILITY COST OPTIMISATION

Filter Is system object : False Filter workloads

Name ↑	Status	Type	Pods	Namespace	Cluster
ai-loan-pod	Running	Pod	1/1	default	ai-loan
k8-connection-gitlab-agent-v2	OK	Deployment	2/2	gitlab-agent-k8-connection	ai-loan

Below we were able to deploy the container image from our gitlab container registry to our pod in the ai-loan cluster. Then we exposed the pod via a loadbalancer service to enable access to the system. This enables anyone to access our system via the endpoint 34.90.250.219 at port 80.

Containers

Name ↑	Status	Image	Restart count	Logs
ai-loan-pod	Running	registry.git.chalmers.se/courses/dit826/2023/group6/ailoan:latest	0	View logs

Exposing services

Name ↑	Type	Endpoints
ai-loan-pod-5q65s	Load balancer	34.90.250.219:80
loan-app-svc	Node port	34.118.225.145:80 TCP

- To access the app service, follow the link http://34.90.250.219/
- To pull the container image, use the image path: ***registry.git.chalmers.se/courses/dit826/2023/group6/ailoan:latest***

4.7 AI Efficiency

Decouple feature engineering:

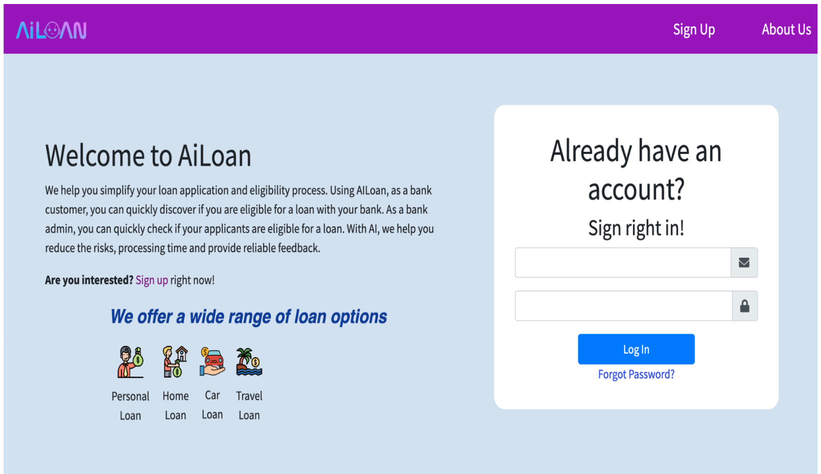
We used shap values to achieve explainable AI which can give an insight to the individual's loan application. It analyses if each of the inputs in the application form has a positive or negative impact in the overall decision about the declining or approving the application.
This analysis is separated from the core model training, therefore we managed to apply the decoupling feature engineering.

We didn't apply model sequencing (referring to the lecture 7), because it adds more complexity to the overall system and managing the flow of information between multiple models can be challenging. For simplicity and ease of maintenance, avoiding model sequencing reduces the complexity of the system.

4.8 User interaction with the system

Welcome Page

The welcome page serves as the entry point to the application, offering a user-friendly experience. Users are presented with clear options to either log in or sign up if they don't have an existing account, ensuring a seamless and personalised interaction with the app.

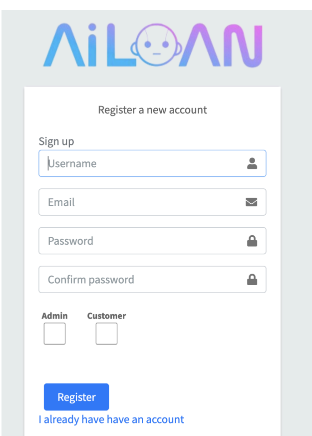


Login Form

Within the login section, users encounter a straightforward form for easy access to their accounts. To enhance user convenience, a password reset feature is also available. This ensures a smooth sign-in process, allowing users to reset their password effortlessly should they encounter any difficulties during login.

Sign up

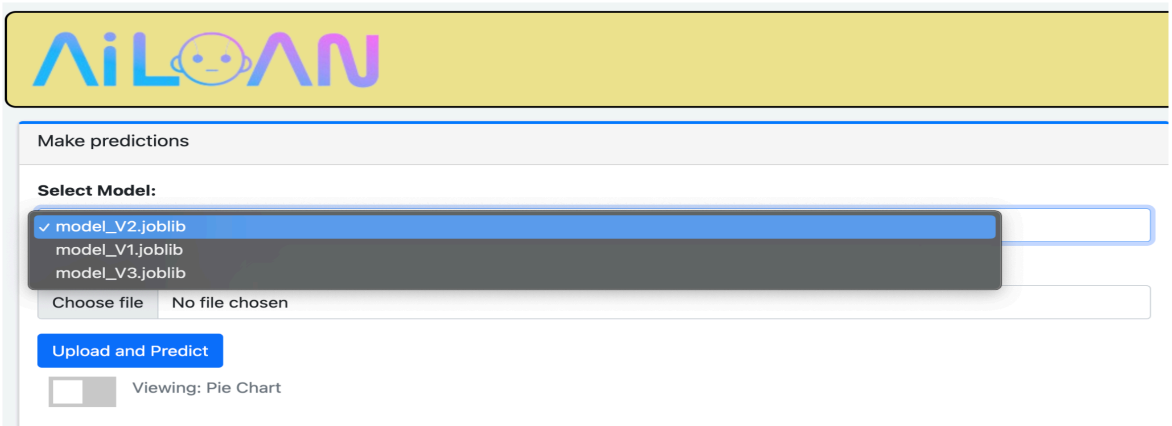
If the user doesn't have an account, an account can be created via the sign up link in the form and at the top of the welcome page. The user can also choose the role they want to sign up for, whether it's an admin role or the customer role.



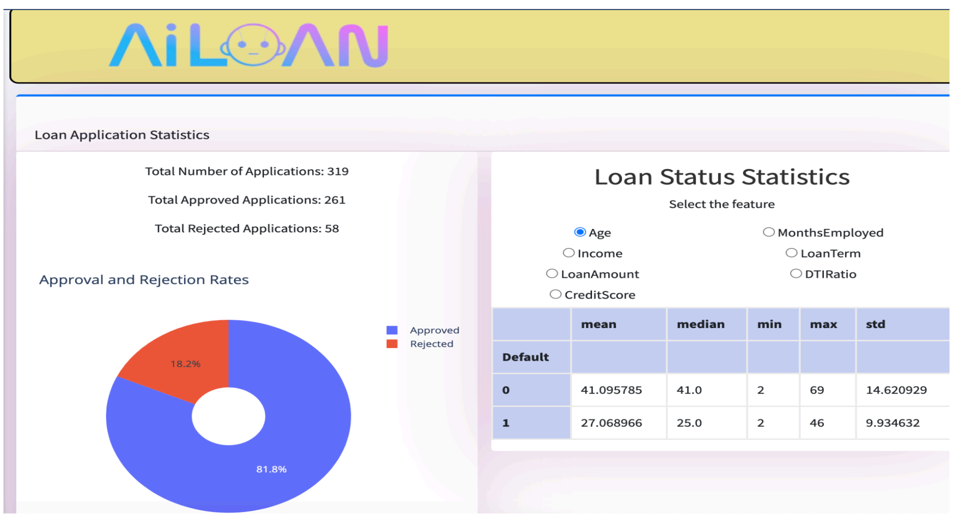
Starting with the Admin Dashboard, the admin can choose any of the options such as predictions, reports, performance, applicants, information and logout from the side bar for data analysis of applications.

The admin can upload the CSV. file to apply the predictions that can be shown via the pie chart and the table.

The admin can choose one of the three models and get the prediction result based on the model they choose.There is also an option to view the results as the pie chart or table.

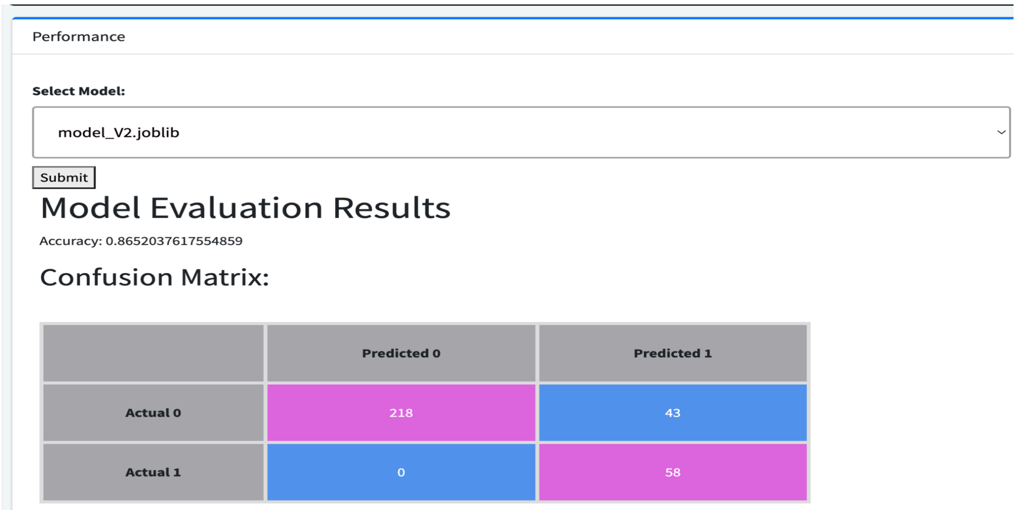


In the reports page, the Admin can investigate the statistics related to the loan applications to get the whole picture of the applications that have been registered. In this case, they can see the total number of applications, number of approved applications and number of rejected applications.



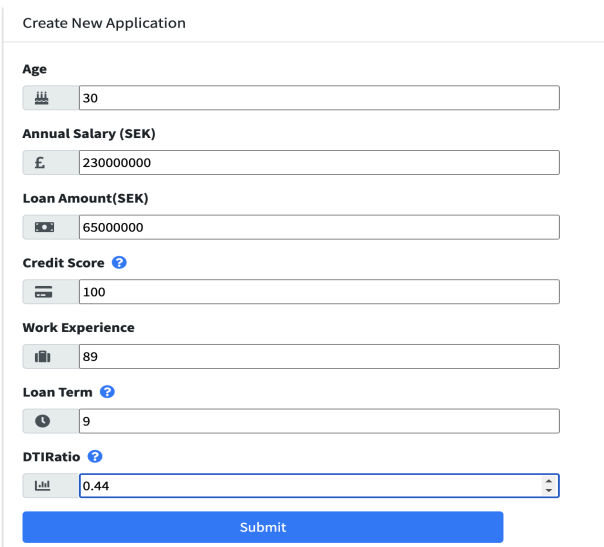
	mean	median	min	max	std
Default					
0	41.095785	41.0	2	69	14.620929
1	27.068966	25.0	2	46	9.934632

In the ‘performance’ page, Admin can investigate the accuracy and model evaluation based on the confusion matrix which represents true positive, true negative, false positive and false negative values based on the number of rejections and approvals.

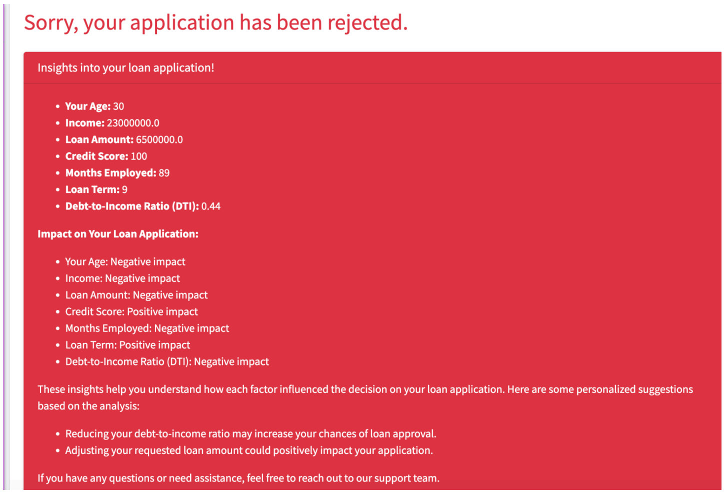


If the user signs in as a ***customer***, they will have access to functionalities such as account setting, create application, chat assistance and logout.

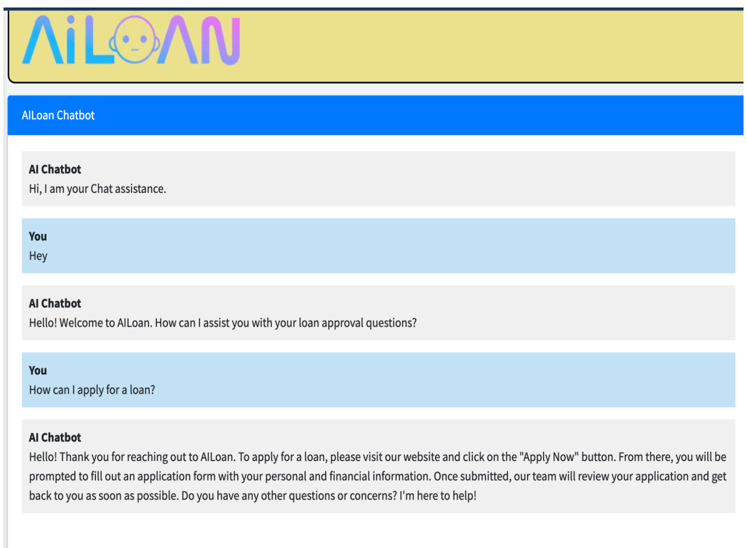
The user can apply for a loan via the 'Create Application' page. They also would get some suggestions about their application and how they can improve their application to be eligible for getting a loan in the bank.



For getting more information about the application after submission, the implementation of explainable AI can show an insight into the application and what can be improved in the application.



Applicants can get more information using the chatbot we integrated into the system. The chatbot will answer the user's loan approval related question and assist them to have a better experience with the application.



Section 5. Software quality

Author: Kanokwan

5.1 Software quality in code structuring

5.1.1 Modularity

In addressing the programming principles for code structuring, our project follows the Model-View-Controller (MVC) architecture provided by Django. This architectural pattern emphasises modularity where models, views, and controllers have distinct roles.

For instance, models in models.py encapsulate data-related logic (models.py) , views in views.py control and manage the function between the data and the interfaces, while the presentations are all stored in the templates that we have as a separate folder to contains all of our admin and user HTML files.

views.py

```python
@login_required
def applicants(request):
    context = {'messages': []}
    if request.user.is_authenticated:
        applicants = NewLoanApplicant.objects.all()
        pending = NewLoanApplicant.objects.filter(status='pending')
        context['pending_count'] = pending.count()

        return render(request, 'admin/applicants.html', {
            "applicants": applicants,
            "pending_count": context['pending_count'],
        })

    return redirect(login)

def update_status(request):
    if request.method == 'POST':
        loan_id = request.POST.get('loan_id')
        new_status = request.POST.get('new_status')

        try:
            applicant = NewLoanApplicant.objects.get(LoanID=loan_id)
            applicant.status = new_status
            applicant.save()
            messages.success(request, f'Status updated successfully for Loan ID {loan_id}.')
        except NewLoanApplicant.DoesNotExist:
            messages.error(request, f'Loan ID {loan_id} not found.')

    return redirect('applicants')
```

models.py

```python
class NewLoanApplicant(models.Model):
    STATUS_CHOICES = [
        ('pending', 'Pending'),
        ('reviewed', 'Reviewed'),
    ]

    LoanID = models.CharField(max_length=12, primary_key=True, default=shortuuid.uuid)
    Age = models.IntegerField()
    Income = models.FloatField()
    LoanAmount = models.FloatField()
    CreditScore = models.IntegerField()
    MonthsEmployed = models.IntegerField()
    LoanTerm = models.IntegerField()
    DTIRatio = models.FloatField()
    Default = models.IntegerField(null=True, blank=True)
    status = models.CharField(max_length=10, choices=STATUS_CHOICES, default='pending')
    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name='loan_applications')

    def __str__(self):
        return f"Loan Application {self.LoanID} by {self.user.username} - Status: {self.status}"
```

applicant.html

```html
applicants.html ×
loanProject > templates > admin > applicants.html > html > body.hold-transition.sidebar-mini.layout-fixed > div.wrapper > div.content-wrapper
                <th>Loan Amount</th>
                <th>Credit Score</th>
                <th>Months Employed</th>
                <th>Loan Term</th>
                <th>DTI Ratio</th>
                <th>Status</th>
                <th>User</th>
                <th>Update Status</th>
              </tr>
            </thead>
            <tbody>
              {% for applicant in applicants %}
                <tr>
                  <td>{{ applicant.LoanID }}</td>
                  <td>{{ applicant.Age }}</td>
                  <td>{{ applicant.Income }}</td>
                  <td>{{ applicant.LoanAmount }}</td>
                  <td>{{ applicant.CreditScore }}</td>
                  <td>{{ applicant.MonthsEmployed }}</td>
                  <td>{{ applicant.LoanTerm }}</td>
                  <td>{{ applicant.DTIRatio }}</td>
                  <td>{{ applicant.status }}</td>
                  <td>{{ applicant.user.username }}</td>
                  <td>
                    <form method="post" action="{% url 'update_status' %}">
                      {% csrf_token %}
                      <input type="hidden" name="loan_id" value="{{ applicant.LoanID }}">
                      <select name="new_status" class="form-control">
                        <option value="pending" {% if applicant.status == 'pending' %}selected{% endif %}>Pending</option>
                        <option value="reviewed" {% if applicant.status == 'reviewed' %}selected{% endif %}>Reviewed</option>
                      </select>
                      <button type="submit" class="btn btn-primary btn-sm mt-2">Update</button>
                    </form>
                  </td>
                </tr>
```

In addition to that, We make sure that different folders are used for specific reasons and are structured in a proper way. Templates are organized in a separate folder, and forms.py that encapsulates logic handled in views.py is also stored in a different file as well. This separation ensures that when modification is made in one area, it does not affect other components in the other area, maintaining clear separation of concerns in our code structure makes it easier for us to debug.

forms.py

```python
class LoanForm(forms.Form):
    Age = forms.IntegerField()
    Income = forms.FloatField()
    LoanAmount = forms.FloatField(label= "Loan Amount")
    CreditScore = forms.IntegerField(label= "Credit Score")
    MonthsEmployed = forms.IntegerField(label= "Months Employed")
    LoanTerm = forms.IntegerField(label= "Loan Term")
    DTIRatio = forms.FloatField(label= "Debt To Income Ratio")
```

views.py

```python
@login_required
def create_applicant(request):
    context = {'messages': []}
    if request.method == 'POST':
        form = LoanForm(request.POST)

        if form.is_valid():
            Age = form.cleaned_data['Age']
            Income_SEK = form.cleaned_data['Income']
            LoanAmount_SEK = form.cleaned_data['LoanAmount']
            CreditScore = form.cleaned_data['CreditScore']
            MonthsEmployed = form.cleaned_data['MonthsEmployed']
            LoanTerm = form.cleaned_data['LoanTerm']
            DTIRatio = form.cleaned_data['DTIRatio']

            manual_exchange_rate = 10
            Income_USD = Income_SEK / manual_exchange_rate
            LoanAmount_USD = LoanAmount_SEK / manual_exchange_rate

            model_path = 'MLmodels/model_V3.joblib'
            model = load_model(model_path)
            prediction = model.predict([[Age, Income_USD, LoanAmount_USD, CreditScore, MonthsEmployed, LoanTerm, DTIRatio]])
            print("Prediction Result:", prediction[0])
```

5.1.2 Reusability

Other than the mentioned modularity, our software promotes reusability by incorporating a separate `utils.py` file. This file contains utility functions that can be shared across different components of the project that we can reuse. This approach enhances maintainability and avoids duplicating code for common functionalities.

utils.py

```python
utils.py  2 ×
loanProject > loanApp > utils.py > ...
import pandas as pd
import io
import os
import joblib
from django.http import HttpResponse
from .models import *


def get_available_models():
    model_folder = 'MLmodels'
    models = [file for file in os.listdir(model_folder) if file.endswith('.joblib')]
    return models

def select_model(selected_model):
    model_path = os.path.join('MLmodels', selected_model)
    if not os.path.exists(model_path):
        raise FileNotFoundError("Selected model not found")

    return load_model(model_path)

def read_csv_file(csv_file):
    try:
        csv_content = csv_file.read().decode("utf-8")
        print("CSV Content:")
        print(csv_content)

        csv_data = pd.read_csv(io.StringIO(csv_content), sep='[;,]', engine='python')
        return csv_data

    except pd.errors.EmptyDataError:
        return HttpResponse("Empty CSV file")
    except Exception as e:
        return HttpResponse(f"Error reading CSV file: {e}")

def apply_label_encoders(csv_data):
```

5.1.3 Scalability

Our project contains scalability in how we structure our code to support the addition of new apps if needed, easily accommodating software requirements in the future. So that if there is a need for additional functionality or services, this design allows for us to add more apps that are responsible for the specific feature without affecting our already existing code structure.

5.2 Software quality in code documentation

5.2.1 Readability

Our project promotes readability by incorporating these practices in our code:

- *Meaningful Variable and Function Names:* The name of variable and function in our project are consistently chosen to enhance the code comprehension and its functionality.
- *Comments:* We added comments on the complex functionality in the code to ensure that everyone in the group and people who read it understand how it functions.
- *Consistent Indentation and Formatting:* We adhere to a uniform coding style and it is maintained throughout the project, promoting a visually cohesive look and comprehensible code.

5.2.2 Maintainability

Our project implements measures to ensure code maintainability, promoting a smooth development process and prevent pipeline problem:

- *Version Control Integration:* Our project is hosted on a version control system, such as Git, facilitating change tracking, collaboration, and easy reversion to previous states if necessary.
- *Dependency Management:* All Dependencies, along with their versions, are explicitly outlined in a dedicated file (requirements.txt), making it easier for our team members to stay synchronised with the project's dependencies.

Section 6. Reflections

Author: Akuien

6.1 Challenges

Wrong data set:

- The team had to change the training data set after the fair which led to a redo of most of the tasks such as training the new model.

Task distribution:

- One team member dropped the course in week 4 towards the fair which led to a task distribution problem which affected the team's delivery.
- Another team member was unreachable and stopped delivering on their allocated tasks from week 7 leaving the rest of the members to take on additional tasks.

GitLab project repository:

- The team's GitLab project repository crashed and we were unable to push or pull from the repository for several days. The team moved the project to GitHub before creating a new project repository on GitLab after discussing with our TA.

6.2 What went well

- Despite the challenges, the team was still able to adjust, adapt and deliver in time.
- All the planned system requirements and user stories were delivered.

6.3 Possible extension for the system

After a prediction of a loan application, we intended to add new ML models that would:

- predict how much money an applicant would be eligible for based on the details provided in the application if it was declined.
- predict the loan term an applicant would be eligible for based on the details provided in the application if it was declined.

Section 7. Advanced AI functionality

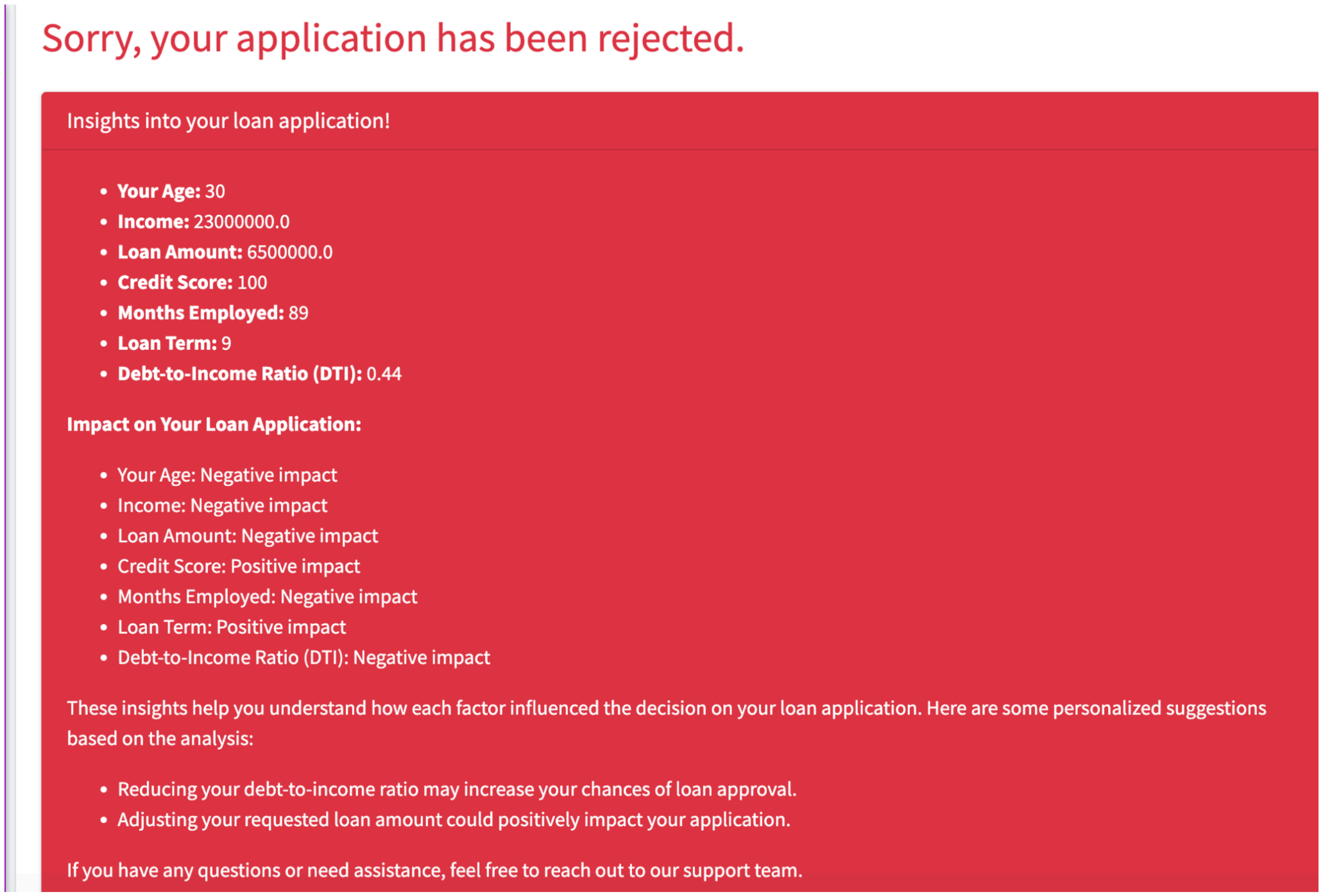
Author: Akuien

7.1 Explainable AI

As our main advance AI functionality and to fulfil the system requirement that (the system should be able to provide an explanation to the user as to why their loan application has been rejected in case of a denied loan application), we implemented an explainable AI feature using the SHAP(SHapley Additive exPlanations) library which is a powerful library in Python for explaining the output of a ML model. It's based on game theory and uses Shapley values to estimate the importance of each feature in a particular prediction. After a rejected prediction, using global interpretability, we calculate the feature importance to check how each of the customer details in their application affected the outcome of the prediction.

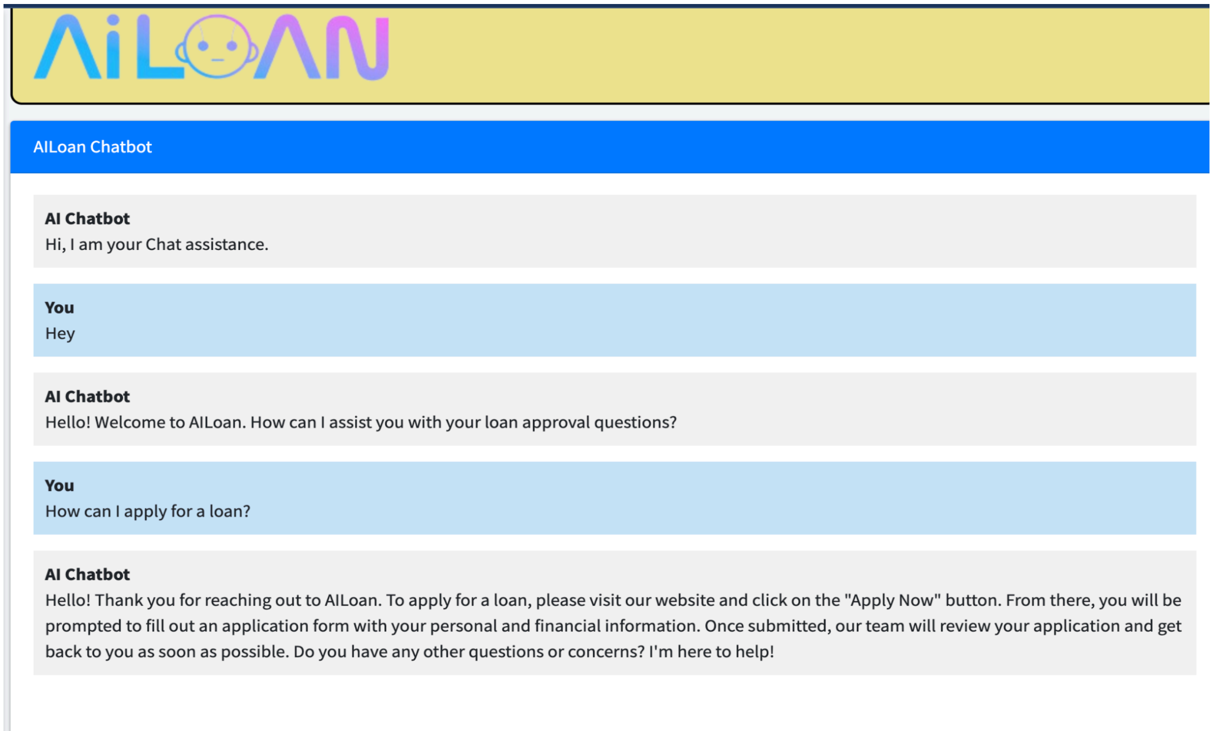
The impact can be either negative or positive accompanied by a calculated number showing how huge or small the impact was in either of the two categories. In case a feature has a negative impact on the prediction, we provide recommendations on how the user can change that feature in order to reduce or remedy its negative impact and possibly change their prediction positively.

The diagram below, shows a case scenario of how the explainable ai works



7.2 ChatBot

In case the explanation provided from the explainable ai was not enough to some users, we encourage the users to use our chatbot which acts as a complementary feature to the explainable ai. With our chatbot, the user can inquire further assistance or explanation on anything they might not have understood. This ensures that the users are able to get all the help and information to improve their chances of getting a positive prediction from their loan applications.
The chatbot is as shown below.



Appendices

Appendix I: Team activity

Attended Meetings	Assignment 1 (out of 6)	Assignment 2 (out of 13)	Notes
Akuen Akoi Deng	6	12	
Cynthia Tarwireyi	6	13	
Daniel Dovhun	5 out of 5	5	In Assignment 1, he was added to the group late. He was present in most meetings but he did not participate. And he did not work on the report
Kanokwan Haesatith	6	13	

Nazli Moghaddam	6	12	
Andreea Lavinia Fulger	4 out of 5	3	Dropped out of the course in week 5

Appendix II: Responsibilities

	Who was responsible	**Delivered**	**Integrated**	**Notes**
	Week 1 (Sprint 1)			
Project ideas brainstorming: Each member was expected to come up with a project idea, find dataset sources, create a simple UI design with Figma and put it in writing for presentation in the next meeting	Everyone	Yes	Yes	
Choosing three project ideas for Tuesday presentation: The team went through all the available project ideas and chose only 3 that could be taken forward.	Everyone	Yes	Yes	
Find the dataset for each of the three ideas	Everyone	Yes	Yes	
	WEEK 2			
Rank the project ideas in terms of preference: Loan, News, Plants	Everyone	Yes	Yes	
Design the presentation slides	Everyone	Yes	Yes	
Choose main project idea: **Loan system**	Everyone	Yes	Yes	
Brainstorm user stories and functional requirements	Everyone	Yes	Yes	
Design the system UI prototype	Andreea	Yes	Yes	

Transfering the dataset to SQLite	Kanokwan Nazli	Yes		
Do risk assessment and management	Nazli	Yes	Yes	
Modifying the readme file	Cynthia	Yes	Yes	
setting up wiki page	Akuien	yes	yes	
Writing sprint 1 report	Akuien	Yes	yes	
	Week 3 (Sprint 2)			
Data Cleaning	Cynthia	Yes	Yes	
Decide which ML model to use	Cynthia/Akuien	Yes	Yes	
Creating the toy prediction model	Cynthia/Akuien	Yes	Yes	
Model evaluation	Cynthia	Yes	Yes	
Setting up CI/CD	Akuien	yes	yes	
Research about Django	Everyone	Yes		
	Week 4			
Setting up initial Django webserver	Kanokwan	Yes	Yes	
Connecting Django, sql via python	Nazli/Kanokwan	Yes	Yes	
Login page (customer)	Akuien/Nazli	Yes	Yes	
Sign up page (customer)	Kanokwan	Yes	Yes	
Loan application (User UI)	Andreea	No	No	Did not manage to implement it during week 4
Profile page (User UI)	Daniel	No	No	Did not manage to implement it

				during week 4
Docker(initial)	Akuein	Yes	yes	
Forgot password	Nazli	Yes	Yes	
deployment(k8)	Akuien	yes		
Feature Engineering	Cynthia	Yes	Yes	
Optimise ML model	Cynthia	Yes	Yes	
Information Page on admin UI	Cynthia	Yes	Yes	
	Week 5 (Sprint 3)			
Loan application (User)	Cynthia	Yes	Yes	
Profile page (User)	Daniel	No	No	It is in progress
Upsampling of target variable	Akuien	Yes	Yes	
Upload csv file and make predictions (Admin)	Cynthia	Yes	Yes	
Create a dashboard with side menu bar (User & Admin UI)	Kanokwan	Yes	Yes	
ER diagram	Kanokwan	(50%)		
deployment(k8)	Akuien	yes	yes	
Create chatbot	Kanokwan	Yes	Yes	
Model Performance page (Admin)	Nazli	Yes	Yes	
Re-train model	Nazli			
	Week 6			
Add logout feature (Admin &User)	Kanokwan	Yes	Yes	
Report page (Admin)	Cynthia	Yes	Yes	
Update welcome page	Akuien	yes	yes	

About Us Page (User UI)	Cynthia	Yes	Yes	
Add profile pic on profile page (User)	Daniel	Yes		
Fine-tuning the chatbot prompt	Kanokwan	Yes	Yes	
Explainable AI	Nazli/Akuien	Yes	Yes	
Unit testing	Nazli/Akuien	Yes	Yes	
Update UI to be more cohesive	Kanokwan	Yes	Yes	
View list of new applicants in a table (Admin)	Daniel	No	No	Did not manage to implement it during week 6
	Post Fair			
	Week 7 (Sprint 4)			
Find new Database	Cynthia	Yes	Yes	
Data Cleaning	Cynthia	Yes	Yes	
Feature Engineering	Cynthia	Yes	Yes	
Model Training	Cynthia	Yes	Yes	
Model versioning	Cynthia	Yes	Yes	
Delete everything related to old model	Cynthia	Yes	Yes	
Update ER diagram	Kanokwan	Yes	Yes	
Update existing code to complement the new ML model and DB	Cynthia	Yes	Yes	
Add more statistical summaries on report page (Admin)	Cynthia	Yes	Yes	
Train new models for predicting amount and loan term of rejected applications	Akuien, Cynthia	Yes	No	
	Week 8			

Beautify welcome page	Cynthia	Yes	Yes	
Update aboutUs page (User)	Cynthia	Yes	Yes	
Notification of new loan applicants (Admin)	Kanokwan	Yes	Yes	
View list of new applicants in a table (Admin)	Kanokwan	Yes	Yes	
Update Explainable AI (User)	Nazli, Akuien	Yes	Yes	
Deploy final latest version of app	Akuien	Yes	Yes	
Manual UI testing	Cynthia, Nazli, Kanokwan, Akuien	Yes	Yes	