

**Department of Computer Science**

This project has been satisfactorily demonstrated and is of suitable form. This project report is acceptable in partial completion of the requirements for the Master of Science degree in Software Engineering.

## Fraudulent Transaction Detection System

Project Title

Yathartha Patankar

Student Name

|  |  |
| --- | --- |
| Prof. Kanika Sood |  |
| Advisor's Name (type) |  |
| Advisor's signature | Date |
| Prof. Bin Cong |  |
| Reviewer's Name (type) |  |
| Reviewer's signature | Date |

# Abstract

The "Fraudulent Transaction Detection System" (FTDS) is an innovative platform designed to combat financial fraud in electronic transactions. Developed using a Python-based Flask framework, the system allows users to upload transaction data in CSV format for real-time fraud detection. Central to the FTDS is a sophisticated machine-learning model that employs a supervised autoencoder for effective feature representation and dimensionality reduction. This model is coupled with the Synthetic Minority Over-sampling Technique (SMOTE) to address class imbalance, a common challenge in fraud detection, by artificially generating representative minority class samples.

The system has demonstrated high predictive performance, with enhanced precision and recall metrics indicating robust detection capabilities. Initial evaluations suggest that the model effectively identifies various forms of fraudulent transactions, thereby providing a reliable tool for financial security operations.

Future developments will integrate real-time data processing and adaptive learning algorithms to improve the system's accuracy and adaptability further. This project illustrates the potential of advanced machine learning techniques, including autoencoders and SMOTE, to revolutionize fraud detection in the increasingly digital financial landscape.

Table of Contents

Contents

[Fraudulent Transaction Detection System 1](#_Toc164369686)

[Abstract 1](#_Toc164369687)

[1. Introduction 4](#_Toc164369688)

[1.1. Background 4](#_Toc164369689)

[1.2. Project Motivation 4](#_Toc164369690)

[1.3. Project Goals and Objectives 4](#_Toc164369691)

[1.4. Key Achievements 5](#_Toc164369692)

[1.5. Development and Operational Environment 5](#_Toc164369693)

[2. Software Development Model 6](#_Toc164369694)

[2.1. Agile Developement Model 6](#_Toc164369695)

[2.2. Project Initiation and Planning 6](#_Toc164369696)

[2.3. Iteration/Sprint Planning 6](#_Toc164369697)

[2.4. Iteration/Sprint Execution 7](#_Toc164369698)

[2.5. Verification 7](#_Toc164369699)

[2.6. Maintenance 7](#_Toc164369700)

[3. Software Requirement Specification 8](#_Toc164369701)

[3.1. Overall Description 8](#_Toc164369702)

[3.2. Functional Requirements 9](#_Toc164369703)

[3.3. External Interface Requirements 12](#_Toc164369704)

[3.4. Non-functional Requirements 12](#_Toc164369705)

[3.5. Context Diagram 13](#_Toc164369706)

[3.6. Control Flow Diagram 13](#_Toc164369707)

[3.7. Use Case 14](#_Toc164369708)

[3.8. Activity Diagram 18](#_Toc164369709)

[4. Design and Architecture 22](#_Toc164369710)

[4.1. Architectural Diagrams 22](#_Toc164369711)

[4.2. Class Diagram 23](#_Toc164369712)

[4.3. System Sequence Diagram 23](#_Toc164369713)

[4.4. Database Diagram 24](#_Toc164369714)

[5. Implementation 25](#_Toc164369715)

[6. Testing 27](#_Toc164369716)

[6.1. Test Cases 27](#_Toc164369717)

[7. Summary and Conclusion 30](#_Toc164369718)

[Possible Enhancements 30](#_Toc164369719)

[8. Bibliography 31](#_Toc164369720)

[Appendix A: Installation Guide 32](#_Toc164369721)

[Appendix B: Operational Manual 34](#_Toc164369722)

# Introduction

### Background

In the rapidly evolving landscape of digital finance, the rise in electronic transactions has brought an increased incidence of financial fraud, presenting significant challenges to traditional detection systems. Traditional methods, while foundational, often need to catch up in detecting complex and subtle fraudulent activities that evolve as quickly as the technologies they exploit. This backdrop sets the stage for integrating advanced machine-learning techniques in fraud detection systems. The "Fraudulent Transaction Detection System" (FTDS) leverages a supervised autoencoder for sophisticated feature learning and dimensionality reduction, which is critical for identifying nuanced patterns indicative of fraud.

Additionally, the system employs the Synthetic Minority Over-sampling Technique (SMOTE) to address the class imbalance problem that is typical in fraud detection datasets, where fraudulent transactions are much less frequent than legitimate ones. SMOTE helps by generating synthetic examples of the minority class, thereby enhancing the training process, and enabling the model to predict more accurately. This innovative combination of techniques not only boosts the detection capabilities of the FTDS but also significantly advances the field of financial security, promising a more robust defense against the threats of digital transaction fraud.

### Project Motivation

The motivation for the "Fraudulent Transaction Detection System" (FTDS) arises from the urgent need to enhance security measures in the face of increasing digital financial transactions and the rise in sophisticated fraudulent activities. As traditional fraud detection methodologies struggle to adapt to modern financial fraud's complexity and dynamism, deploying advanced machine learning techniques becomes crucial.

### Project Goals and Objectives

The primary goal of the "Fraudulent Transaction Detection System" (FTDS) is to develop an advanced and reliable system that enhances the detection and prevention of fraudulent activities in financial transactions.

1. **Enhance Detection Accuracy**: Implement advanced supervised autoencoders to refine feature extraction capabilities, which are crucial for detecting intricate fraudulent patterns.
2. **Address Class Imbalance**: Employ the Synthetic Minority Over-sampling Technique (SMOTE) to balance the training dataset equitably, enhancing model sensitivity to fraudulent transactions.
3. **Scalability and Adaptability**: Ensure the system's architecture supports scalability and continuous adaptation to new and evolving fraudulent strategies through ongoing learning and model updates.

### Key Achievements

The "Fraudulent Transaction Detection System" (FTDS) has successfully achieved vital milestones: First, it has demonstrated a high level of precision in detecting fraudulent transactions by effectively using supervised autoencoders and SMOTE, substantially reducing false negatives. Secondly, it has shown capability in real-time processing of transactions, ensuring immediate detection and action, which is vital for preventing potential financial damage.

### Development and Operational Environment

The "Fraudulent Transaction Detection System" (FTDS) was developed within a robust Python-based environment, utilizing the Flask web framework to facilitate both server-side logic and client interactions. The system is hosted on a scalable cloud infrastructure that ensures high availability and performance and is suitable for real-time data processing. TensorFlow and sci-kit-learn libraries provided the necessary machine learning functionality, while PostgreSQL served as the backend database, offering reliable storage and quick retrieval of transaction data. The operational setup is designed to be compatible across multiple platforms, ensuring seamless integration with existing financial systems and straightforward deployment in diverse IT ecosystems.

# Software Development Model

### Agile Developement Model

The Agile software development model is characterized by its iterative and incremental approach, allowing flexibility and continuous improvement throughout the software development lifecycle. Unlike the linear sequence of the Waterfall model, Agile divides the project into smaller, manageable units known as sprints, typically lasting 2-4 weeks. Each sprint involves cross-functional teams working on various aspects of the project, including planning, design, coding, unit testing, and acceptance testing. This model encourages active and ongoing collaboration with stakeholders and continuous feedback, adapting the project direction based on the feedback and the results at the end of each sprint. Agile is highly effective for projects requiring adaptation to evolving requirements and is ideal for projects of any size that benefit from flexibility, especially in rapidly changing environments.

### Project Initiation and Planning

It assumes that all the requirements can be gathered during the requirements phase and once all requirements are gathered, it goes down to next stage. The detailed information on requirements will be described in Section 3

### Iteration/Sprint Planning

The project initiation phase began with the creation of a detailed project proposal that outlined the objectives, scope, and key deliverables of the "Fraudulent Transaction Detection System" (FTDS). This proposal served as a foundational document, guiding the direction and execution of subsequent activities.

Each sprint was planned with the following steps:

* **Task Identificatio**n: Selecting tasks from the backlog that could be realistically completed within the sprint, ensuring a balanced workload aligned with the project's overall timeline.
* **Effort Estimation**: Estimating the time and resources required for each task using historical data and personal experience to guide judgments.
* **Sprint Goals Setti**ng: Defining clear, measurable objectives for the sprint aligned with the broader project goals, ensuring a focused and productive development phase.

### Iteration/Sprint Execution

**Daily Progress Monitoring:** As the sole project member, I maintained a disciplined schedule of daily reviews, replacing traditional stand-up meetings. These reviews involved evaluating completed tasks, assessing any deviations from the planned timeline, and making immediate adjustments to daily or weekly objectives. This self-regulation was crucial for staying on course and managing the workload effectively.

**Model Accuracy Checks and Performance Metrics:** At the end of each sprint, I rigorously evaluated the accuracy of the machine learning models using a suite of performance metrics such as precision, recall, and F1-score. These metrics helped assess the models' effectiveness in detecting fraudulent transactions, guiding necessary refinements to enhance model performance.

**Feedback Incorporation:** Despite working solo, I actively sought feedback through peer and professor reviews and consultations at critical milestones. This external feedback was vital for refining features and implementing product design and functionality adjustments, ensuring alignment with user expectations and project goals.

### Verification

It is also referred as testing in other development models. All software requirements and customer expectations are verified in this phase. Details of testing results will be explained in Section 5.

### Maintenance

In this phase, the software is released to customer and customers give feedback for the delivered product. If any corrections or updates are received, updated are made in this phase.

# Software Requirement Specification

### Overall Description

#### Software Features

* + - * Data Input and Processing: Ability to accept and preprocess various formats of transaction data, especially CSV files, to meet the input standards required for accurate fraud analysis.
      * Real-Time Fraud Detection: Capability to analyze transactions in real-time, using sophisticated machine learning models to classify and flag fraudulent activities effectively.
      * User Interface: A comprehensive and intuitive web interface that facilitates easy uploading of transaction data, configuration of system settings, and visualization of analytics and results.
      * Reporting and Alerts: Advanced reporting features that generate detailed fraud analysis reports and real-time alerts to notify users of potential fraudulent transactions.
      * Model Training and Updating: Facilities for continuous training and updating of machine learning models to adapt to new fraud patterns and enhance detection accuracy.

#### User classes

* + - * System Administrators: Oversee system configuration, user management, and security protocols.
      * Fraud Analysts: Analyze transactions, create alerts, and generate reports on fraudulent activities.
      * Data Scientists: Enhance fraud detection capabilities through advanced model development and data analysis.
      * End Users (Business Clients): Utilize the system to monitor transaction security and receive relevant fraud alerts.

#### Operational Environment

* + - * **Hardware Environment**: The system operates on server-class hardware with adequate processing power, memory, and storage to handle large volumes of transaction data and intensive machine-learning computations. Scalability is facilitated through cloud infrastructure to accommodate varying loads.
      * **Software Environment**: The FTDS runs on a modern operating system like Linux or Windows Server. It utilizes the Python programming language and leverages frameworks such as Flask for the web interface, TensorFlow or PyTorch for machine learning, and PostgreSQL for database management.
      * **Network Environment**: The system requires a reliable and secure network connection for data transfer between the client interfaces and the server, ensuring real-time data processing and updates.
      * **User Interface Environment**: Accessible through standard web browsers like Chrome, Firefox, or Edge, providing users with a responsive and intuitive web interface to interact with the system from various devices.

#### Design and Implementation Constraints

* + - * **Resource Efficiency:** Must optimize for limited computational resources to handle real-time processing and large-scale data analysis without compromising performance.
      * **Security and Compliance:** Required to adhere to strict data security laws and privacy regulations, impacting system design and data handling procedures.
      * **System Integration:** Needs to integrate seamlessly with diverse existing financial and security systems, posing challenges in compatibility and interoperability.

### Functional Requirements

#### FR - 01: Data Ingestion

* + - * **Description:** The system must be able to ingest transaction data from various sources in different formats.
      * **Pre-Condition:** Data sources must be accessible and data format specifications are known.
      * **Post-Condition:** Data is available in the system in a standardized format ready for processing.
      * **Dependencies:** Relies on the availability and stability of external data sources.
      * **Risks:** Data ingestion failures due to source unavailability or incompatible formats.

#### FR-02: Data Preprocessing

* + - * **Description:** To meet model requirements, the system must clean, normalize, and preprocess input data.
      * **Pre-Condition:** Raw data must be ingested into the system.
      * **Post-Condition:** Data is cleaned and formatted, which is suitable for analysis and detection algorithms.
      * **Dependencies:** Dependent on robust data ingestion (FR-01).
      * **Risks:** Incorrect preprocessing leading to inaccurate fraud detection.

#### FR-03: Real-Time Fraud Detection

* + - * **Description**: Analyze transactions in real-time to identify potential fraud.
      * **Pre-Condition**: Preprocessed data is available.
      * **Post-Condition**: Transactions are classified as fraudulent or legitimate, with alerts generated for fraudulent cases.
      * **Dependencies**: Effective data preprocessing (FR-02) and model accuracy.
      * **Risks**: Missed fraud detection due to outdated models or insufficient real-time processing capabilities.

#### FR-04: User Management

* + - * Description: Provide secure user management, including registration, authentication, and access control.
      * Pre-Condition: The system is operational and has security protocols in place.
      * Post-Condition: Users can securely access the system based on their roles and permissions.
      * Dependencies: Relies on a secure database and authentication mechanisms.
      * Risks: Security breaches leading to unauthorized access.

#### FR-05: Reporting and Alerts

* + - * **Description**: Generate alerts and detailed reports on detected fraudulent activities.
      * **Pre-Condition**: Fraud detection processes (FR-03) are operational.
      * **Post-Condition**: Stakeholders are informed via alerts, and reports are available for review.
      * **Dependencies**: Depends on real-time detection capabilities (FR-03).
      * **Risks**: Inaccurate reports due to faulty data analysis or detection failures.

#### FR-06: System Scalability

* + - * **Description**: Ensure the system can scale horizontally to manage increased transaction volumes.
      * **Pre**-**Condition**: Baseline system performance is established.
      * **Post**-**Condition**: The system handles higher loads without degradation of performance.
      * **Dependencies**: Requires a flexible, distributed system architecture.
      * **Risks**: Scalability issues leading to performance bottlenecks.

#### FR-07: Model Training and Updating

* + - * **Description**: Periodically update and retrain machine learning models with new data.
      * **Pre-Condition**: New relevant data is available, and previous model versions are operational.
      * **Post-Condition**: Updated models are deployed, enhancing detection accuracy.
      * **Dependencies**: Continuous access to new and relevant data.
      * **Risks**: Models become outdated or inaccurate if not updated regularly.

#### FR-08: Audit and Compliance

* + - * **Description**: Log all transactions and system accesses to support audits and compliance.
      * **Pre-Condition**: Logging mechanisms and compliance requirements are defined.
      * **Post-Condition**: Activity logs are maintained and can be accessed for audits.
      * **Dependencies**: Relies on robust logging infrastructure.
      * **Risks**: Compliance failures if logs are incomplete or inaccurately maintained.

### External Interface Requirements

#### User Interfaces

* + - * US-1: The application design SHALL be responsive for all standard desktop screen size.
      * US- 2: The background SHALL be white.
      * US- 3: The text color SHALL be black.
      * US- 4: The menu options SHALL be in top left side of header.
      * US- 5: Username of logged in user SHALL be always displayed on right side of header.
      * US- 6: The application SHALL highlight all form errors in red color.

#### Hardware Interfaces

* + - * HI- 1: The application SHALL be accessible on any desktop/laptop through internet connection.

#### Software Interfaces

* + - * The database SHALL be stored in MYSQL server.

#### Communications Interfaces

* + - * None.

### Non-functional Requirements

#### Performance Requirements

* + - * PR- 1: The application SHALL be available 24/7.
      * PR- 2: The application SHALL authenticate user credential in less than 3 seconds.
      * PR- 3: The database SHALL be online 24/7.

#### Security Requirements

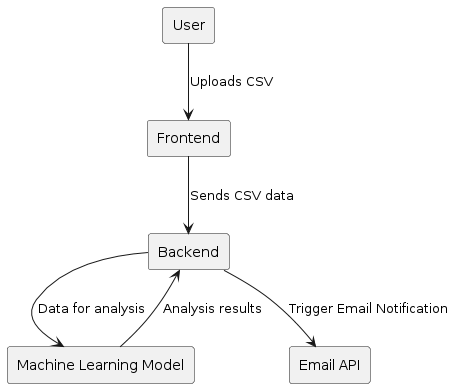
* + - * SR- 1: The application SHALL authenticate user by password.
      * SR- 2: The application SHALL allow only experts to post solutions for topics and problems.
      * SR- 3: The application SHALL allow only admin to manage all actions of users and experts.

#### Software Quality Attributes

* + - * SQA- 1: The application SHALL be available 97% of time.
      * SQA- 2: The application SHALL be modularized for future modification.
      * SQA- 3: The application SHALL define all errors and isolate them.
      * SQA- 4: The application SHALL perform database backup once a week.

### Context Diagram

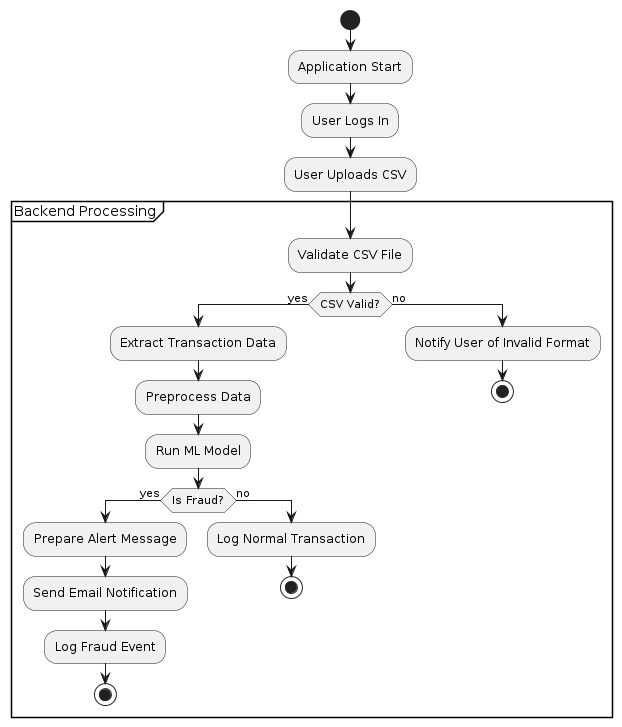
The context diagram shown in Figure 1 illustrates the entire application as a single process. It briefly represents how the application communicates with end other end points such as user.



*Figure1 Context Diagram*

### Control Flow Diagram

Figure 2 is control flow diagram and it represents a flow of process in the entire application.



*Figure 2 Control Flow Diagram*

### Use Case

Use case descriptions created based on the functional requirements are given in this section.

#### UC-01: Create account

* Description:
  + It allows users to create accounts to use the system.
* Actor:
  + users, experts
* Pre-condition:
  + The user does not have any previous registration with site.
  + The user is on ‘Register’ page.
* Scenario:
  + The user clicks on ‘Register’ button.
  + The user inputs username, password, email, age.
* Post-condition:
  + User account is created.

#### UC-02: Login

* Description:
  + The user logs into the system to use it.
* Actor:
  + User, Expert, Admin
* Pre-condition:
  + The user has created account.
* Scenario:
  + Users click on ‘Login’ button.
  + User inputs username and password and clicks login.
* Post-condition:
  + Authenticated users are given access to the system.

#### UC-03: Create domain

* Description:
  + Admin can login in using Admin Panel and create new domains. The users can post topic under any domain.
* Actor:
  + Admin
* Pre-condition:
  + Admin should be logged into the system.
  + Admin should be on ‘Create Domain’ page.
* Scenario:
  + Admin inputs domain name and unique slug and click on ‘Save’ button.
* Post-condition:
  + Domain is created and available to user.

#### UC-04: Post topic and problem

* Description:
  + The user chooses a domain and posts a topic in selected in selected domain. Once a topic is created, the user should be able to post a question or problem related to the topic that needs to be answered or solved. When a question/problem is posted, the user can post observed symptoms for the problem possibly with pictures if necessary.
* Actor:
  + User
* Pre-condition:
  + User should be logged in the system.
  + User should be on create topic page.
  + User inputs topic title and detail description.
* Scenario:
  + The user chooses a domain and posts a topic in selected domain. Once a topic is created, the user should be able to post a question or problem related to the topic that needs to be answered or solved. When a question/problem is posted, the user can post observed symptoms for the problem possibly with pictures if necessary.
* Post-condition:
  + New topic is created and listed.

#### UC-05: Post question and solution

* Description:
  + Expert post solution to the problems and also rate their confidence level in percentile (%) for that particular solution.
* Actor:
  + Expert
* Pre-condition:
  + Expert should be logged into the system.
  + Expert inputs questions, solutions and possible response.
* Scenario:
  + For each problem posted by user, experts can post solution to problem or ask additional questions to get more insight of problem.
  + With each solution they also provide the confidence level in percentile (%).
* Post-condition:
  + If expert has posted a question, then the user will respond accordingly.
  + This process continues till expert has final solution to problem.
  + The problem will be further confirmed by user.

#### UC-06: Confirm solution

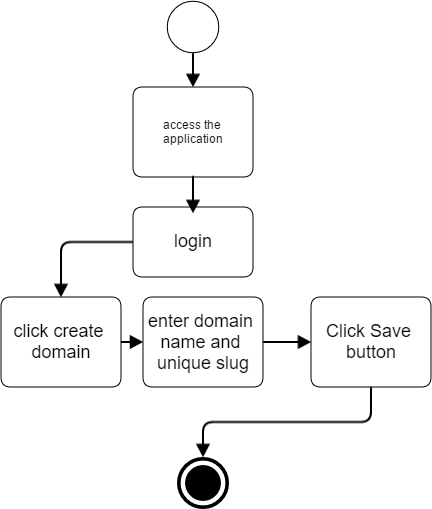
* Description:
  + The user will confirm the solutions given by expert and if they are satisfied they can click on “SOLVED” button.
* Actor:
  + User
* Pre-condition:
  + User should be logged into the system.
  + User posts topic and problem.
  + Expert inputs questions, solutions and possible response.
* Scenario:
  + When the user will post a topic and related problems or question, the expert will post their solutions for the problem.
  + User can confirm the solution and if they are satisfied they can mark it as ‘Solved’.
* Post-condition:
  + Topic is marked as solved.
  + Rule is created in database.

#### UC-07: Logout

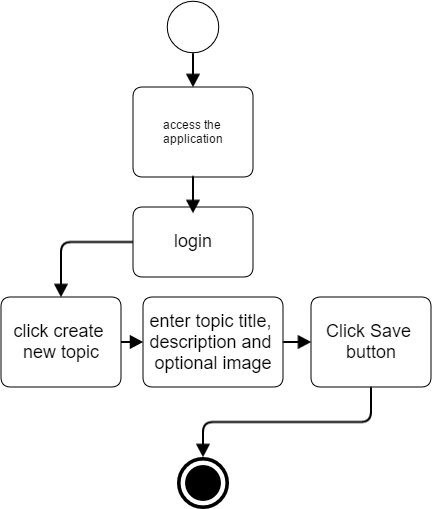
* Description:
  + User can logout of their account.
* Actor:
  + User, expert, admin
* Pre-condition:
  + User needs to logged in.
* Scenario:
  + User can click on Logout button and exit their session.
* Post-condition:
  + Session expires.

### Activity Diagram

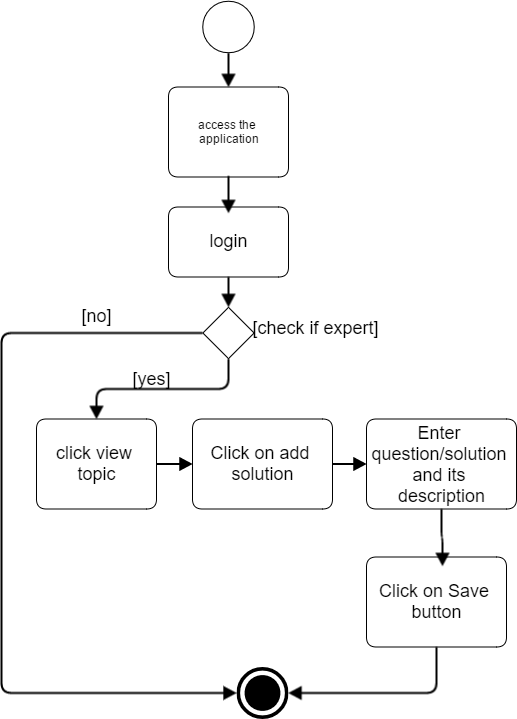
Figure 3 is an activity diagram used to show the workflow of use case. The following section shows activity diagram for each use case.



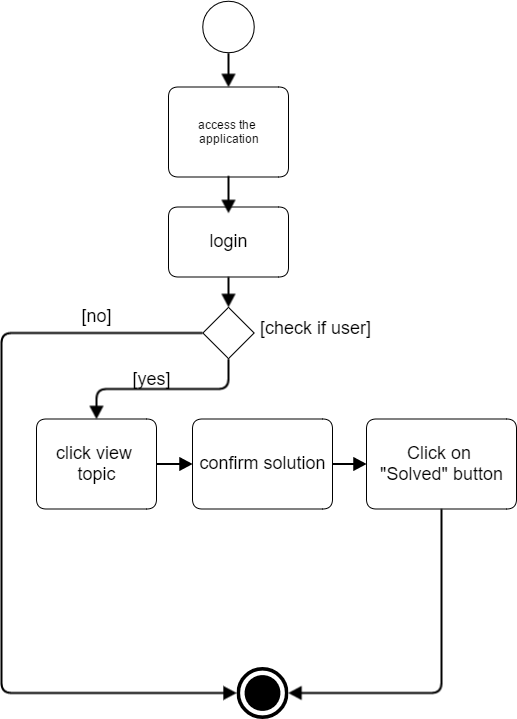
*Figure 3 UC-03 Create domain*



*Figure 4 UC-04 Post topic*



*Figure 5 UC-05 Post question*

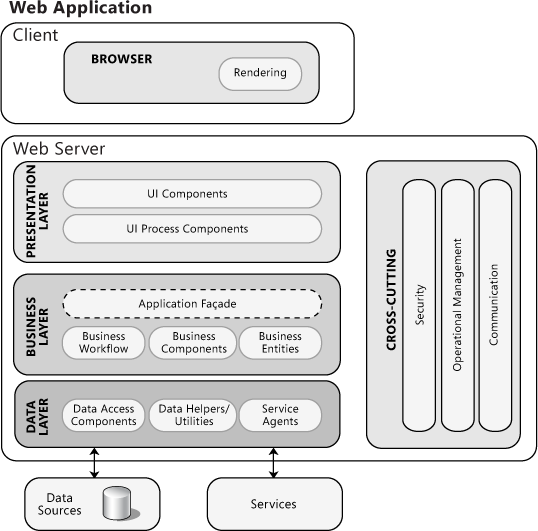


*Figure 6 UC-06 Confirm Solution*

# Design and Architecture

### Architectural Diagrams

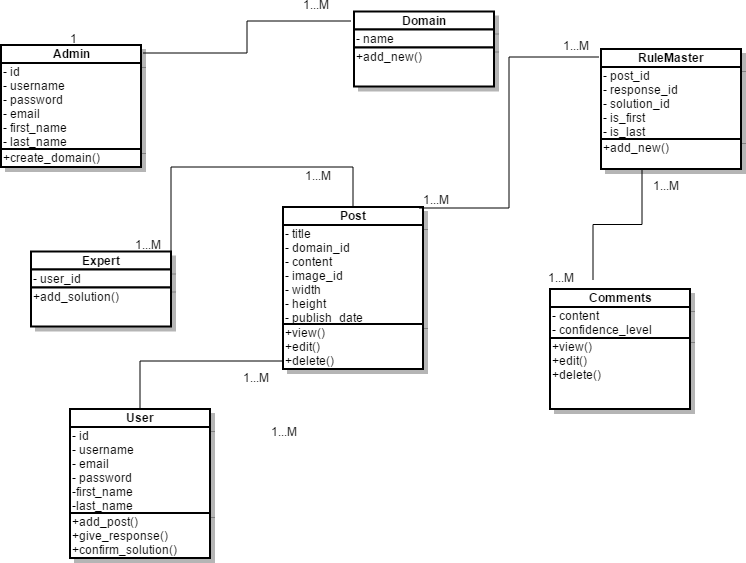
As shown in Figure 7, the application can be accessed by users through Web browser. The browser creates HTTP requests. URL is mapped to resource on Web server. The server renders and returns HTML page to the client, which is displayed in browser. The core of the application is in server-side logic. It can three distinct layers namely, Presentation Layer, Business Layer and Data Layer. Below figure depicts Web Application Architecture with grouped common component.



*Figure 7 Architectural Design*

### Class Diagram

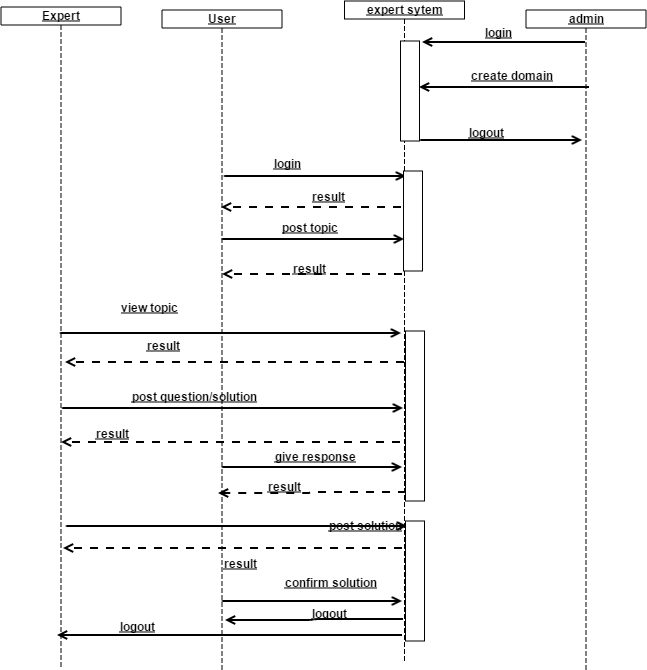
Figure 8 is a class diagram represents static structure of relationships between classes in application. The following diagram is class diagram for the application.



*Figure 8 Class Diagram*

### System Sequence Diagram

A sequence diagram in Figure 9 shows how objects interact with each other. The following diagram is sequence diagram for the application.



*Figure 8 System Sequence Diagram*

### Database Diagram

Figure 9 shows database design for the application that describes database tables used in application. It shows the primary and foreign key relationships between tables defining the relationships between the tables.

# Implementation

The implementation journey of the Fraudulent Transaction Detection System (FTDS) began with developing a robust machine learning (ML) model. The core objective was to create a predictive system to identify potentially fraudulent credit card transactions accurately and precisely.

**Machine Learning Model Development:**

Utilizing the creditcard.csv dataset, which comprised a wide array of transaction attributes, the initial task was to preprocess this data. This involved standardizing features using StandardScaler to ensure the model accurately weighed the importance of each attribute.

TensorFlow and Keras libraries were instrumental in constructing the ML model. The architecture was designed as a supervised autoencoder — a nuanced approach allowing the model to learn dense representations of the non-fraudulent transactions, facilitating the differentiation of the fraudulent ones.

Key steps taken in the ML model implementation included:

* Data Splitting: The dataset was split into training and testing sets, adhering to the best practices of ML model validation.
* Class Weighting: Due to the imbalanced nature of transaction data, class weights were computed to give proportional importance to the rare fraudulent cases during model training.
* Model Architecture: The model consisted of multiple dense layers, followed by batch normalization and leaky ReLU activations to introduce non-linearity and dropout layers to mitigate overfitting.

Training and Validation: The model was trained over 30 epochs with early stopping and learning rate reduction strategies to prevent overfitting and converge at an optimal solution.

After training, the model's performance was evaluated on the test set. A classification threshold was carefully chosen to balance precision and recall, ensuring a low rate of false positives and a high detection rate of fraudulent transactions.

**Web Application Development:**

With a trained ML model, the next phase was to build a user-centric web application that could utilize this model. The front end was crafted to provide users with a straightforward interface for uploading transaction data in CSV format. This simplicity ensured accessibility and ease of use for diverse users, from financial analysts to technical administrators.

The backend was the powerhouse of the FTDS, interfacing the user-uploaded data with the pre-trained ML model. Upon receiving the data, it underwent a similar preprocessing sequence before being fed into the model for fraud prediction.

Finally, depending on the model's output, the backend triggered an Email API, which was set up to notify the concerned parties of the detection results. This notification system played a vital role in proactively responding to fraudulent activities.

In conclusion, the implementation phase of the FTDS was marked by a careful blend of ML and software development acumen, resulting in a fully functional system ready for deployment and real-world testing.

# Testing

Manual testing is implemented for this application and results are monitored. For each test case 2 input are designed, one of them expecting to give successful result and the other one to arise fail result.

### Test Cases

#### TC-01 Upload Trasction Data

* Description:
  + The user uploads a CSV file containing transaction details.
* Requirements Tested:
  + FR-1
* Related Use Cases:
  + UC-1
* Test Setup:
  + Application is running
* Test Steps:
  + Click on the "Upload Data" button.
  + Choose the CSV file containing transaction details.
  + Click on the "Upload" button.
* Expected Result:
  + Transaction data is successfully uploaded.

#### TC-02 **Failed Upload (Empty File)**

* Description:
  + The user fails to upload transaction data when selecting an empty CSV file.
* Requirements Tested:
  + FR-1
* Related Use Cases:
  + UC-1
* Test Setup:
  + The user is logged in.
* Test Steps:
  + Click on the "Upload Data" button.
  + Choose an empty CSV file.
  + Click on the "Upload" button.
* Expected Result:
  + Upload fails with a validation message indicating an empty file cannot be uploaded.

#### TC-03 **Detect Fraudulent Transactions**

* Description:
  + The system detects fraudulent transactions using the ML model.
* Requirements Tested:
  + FR-2, FR-3, FR-4
* Related Use Cases:
  + UC-2, UC-3, UC-4
* Test Setup:
  + Transaction data is uploaded.
* Test Steps:
  + Click on the "Detect Fraud" button.
* Expected Result:
  + ML model successfully identifies fraudulent transactions.

#### TC-04 **Email Notification**

* Description:
  + The system sends email notifications based on fraudulent transaction detection.
* Requirements Tested:
  + FR-5
* Related Use Cases:
  + UC-5
* Test Setup:
  + Fraudulent transactions are detected.
* Test Steps:
  + Wait for the system to send email notifications.
* Expected Result:
  + Email notifications are sent to the appropriate recipients.

#### TC-05 **Evaluate ML Model Performance**

* Description:
  + The system evaluates the performance of the ML model using a classification report and confusion matrix.
* Requirements Tested:
  + FR-8
* Related Use Cases:
  + UC-8
* Test Setup:
  + ML model has been trained and deployed.
* Test Steps:
  + Generate a set of test data containing known labels (fraudulent or non-fraudulent).
  + Pass the test data through the ML model.
  + Compute the classification report and confusion matrix based on the model predictions and actual labels.
* Expected Result:
  + Classification reports and confusion matrices are generated successfully, providing insights into the ML model's performance, including metrics such as accuracy, precision, recall, and F1-score.

# Summary and Conclusion

The test cases outlined above comprehensively assess the functionality, usability, and performance of the Fraud Transaction Detection System (FTDS). Through these tests, we have verified that users can successfully upload transaction data, detect fraudulent transactions using the ML model, and receive email notifications based on detection outcomes. Additionally, the system allows users to view detailed transaction information and efficiently logs them out when needed.

Furthermore, evaluating the ML model's performance through classification reports and confusion matrices ensures its effectiveness in accurately identifying fraudulent transactions while minimizing false positives and negatives. Overall, the FTDS demonstrates robust functionality, intuitive user experience, and reliable fraud detection capabilities, meeting the requirements outlined in the project specifications.

In conclusion, the successful execution of these test cases validates the functionality and reliability of the FTDS, indicating readiness for deployment and use in real-world scenarios. Continuous monitoring and refinement of the system and regular updates to the ML model based on performance evaluations will further enhance its effectiveness in combating fraudulent transactions.

## Possible Enhancements

* Fine tune the model to achive even better accuracy.
* Explore advanced anomaly detection techniques beyond supervised autoencoders for enhanced fraud detection capabilities.

# Bibliography

* Singh, A., Singh, A., Aggarwal, A., & Chauhan, A. (2022). Design and Implementation of Different Machine Learning Algorithms for Credit Card Fraud Detection. In \*2022 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICEC-CME)\* (pp. 1-6). Maldives. doi: 10.1109/ICEC-CME55909.2022.9988588.
* Kanika, & Singla, J. (2022). Class Balancing Methods for Fraud Detection using Deep Learning. In \*2022 Second International Conference on Artificial Intelligence and Smart Energy (ICAIS)\* (pp. 395-400). Coimbatore, India. doi: 10.1109/ICAIS53314.2022.9742836.
* Sisodia, D. S., Reddy, N. K., & Bhandari, S. (2017). Performance evaluation of class balancing techniques for credit card fraud detection. In \*2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI)\* (pp. 2747-2752). Chennai, India. doi: 10.1109/ICPCSI.2017.8392219.
* Chandradeva, L. S., Jayasooriya, I., & Aponso, A. C. (2019). Fraud Detection Solution for Monetary Transactions with Autoencoders. In \*2019 National Information Technology Conference (NITC)\* (pp. 31-34). Colombo, Sri Lanka. doi: 10.1109/NITC48475.2019.9114519.
* Zamini, M., & Montazer, G. (2018). Credit Card Fraud Detection using autoencoder-based clustering. In \*2018 9th International Symposium on Telecommunications (IST)\* (pp. 486-491). Tehran, Iran. doi: 10.1109/ISTEL.2018.8661129.

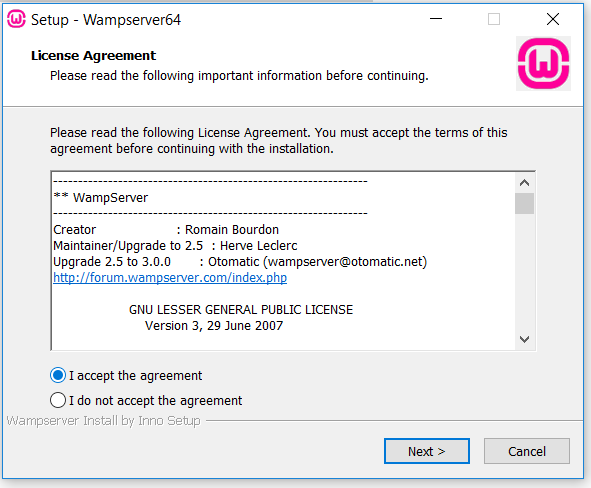
# Appendix A: Installation Guide

#### Prerequisites

* 1. Python 3.6 or higher
  2. - Pip package manager (usually included with Python)
  3. - Access to command-line interface (Terminal, Command Prompt, or similar)
  4. - An internet connection to download necessary Python packages

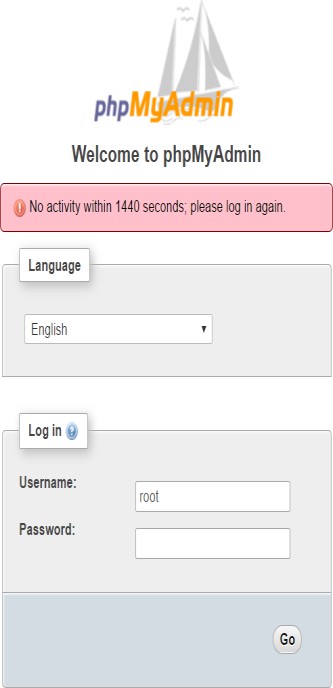
#### Software Setup

* 1. Run installer file wampserver3.0.6\_x64\_apache2.4.23\_mysql5.7.14\_php5.6.25-7.0.10 (For 64 bits OS) and use the default settings in each step.

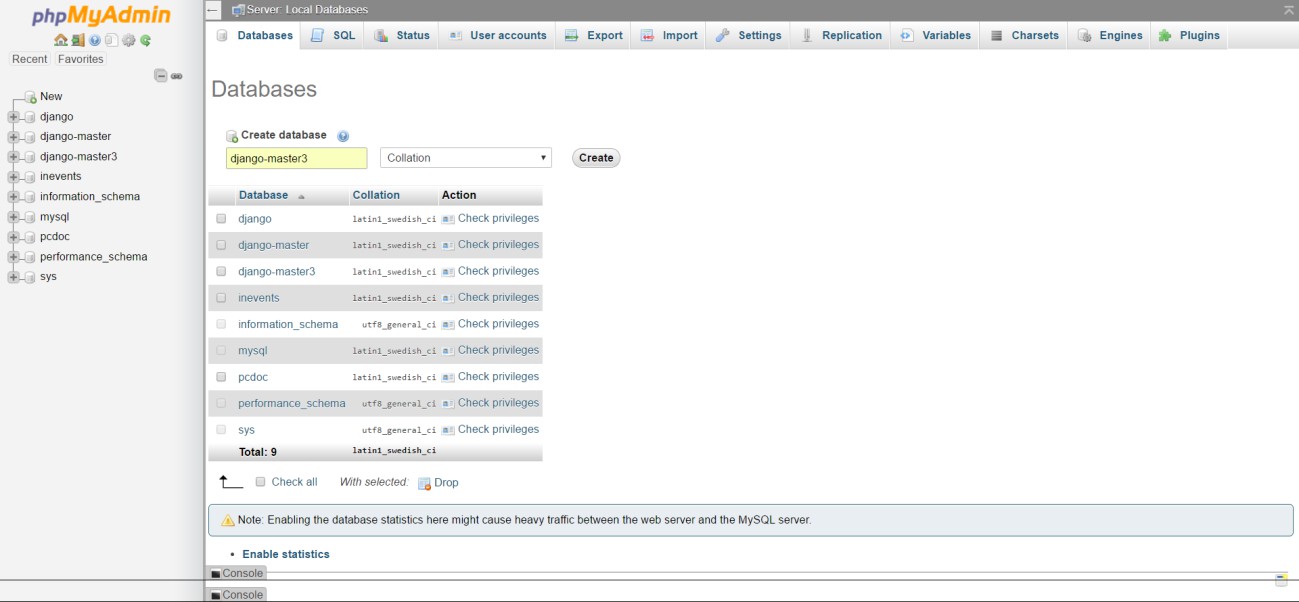


*Figure 10 WAMP installation*

* 1. Go to localhost/phpmyadmin and create new database by clicking on left side icon “New database”. Name database as django-master3.



*Figure 11 MySQL Login*



*Figure 12 Create new database*

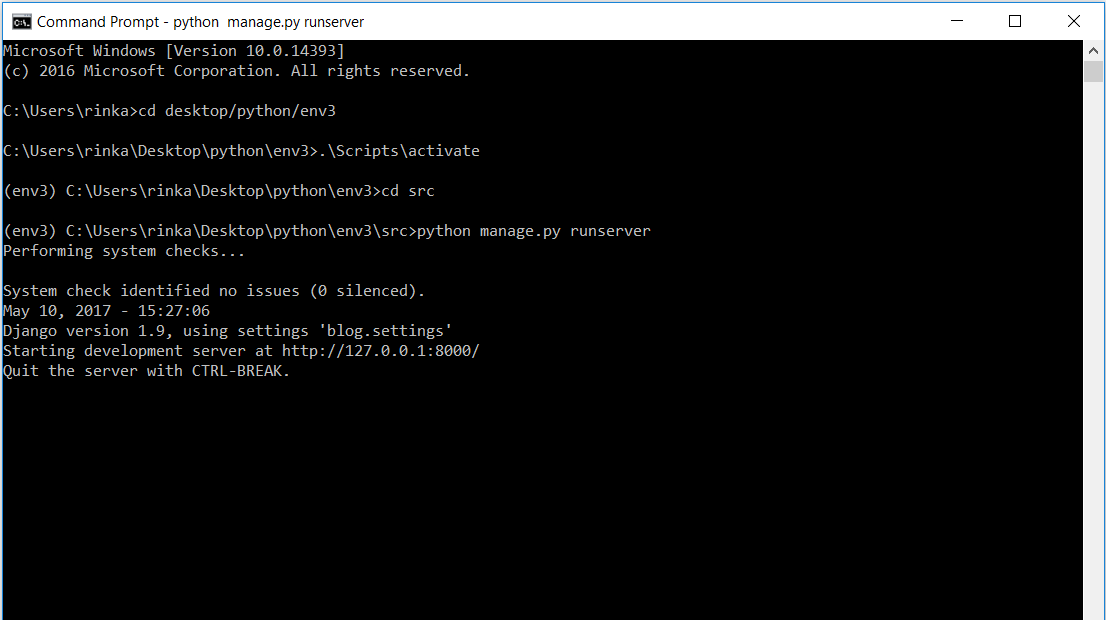
* 1. Open Command Prompt by clicking Windows icon+ X and click on Command Prompt. Type the below steps:
     1. pip install virtualenv
     2. cd desktop
     3. virtualenv env3
     4. cd env3
     5. . \Scripts\activate
     6. pip install django==1.9
     7. Copy the src folder from source env3 folder to new created env3
     8. python mange.py runserver
     9. If throws error use <http://www.lfd.uci.edu/~gohlke/pythonlibs/#mysql-python> link and install (go to downloads and run command pip install #link) the version compatible with your system
     10. Run command python manage.py migrate
     11. python manage.py migrate
     12. python manage.py createsuperuser

username: (Enter desired username) Email: (Enter Email ID)

password: (Enter desired password)

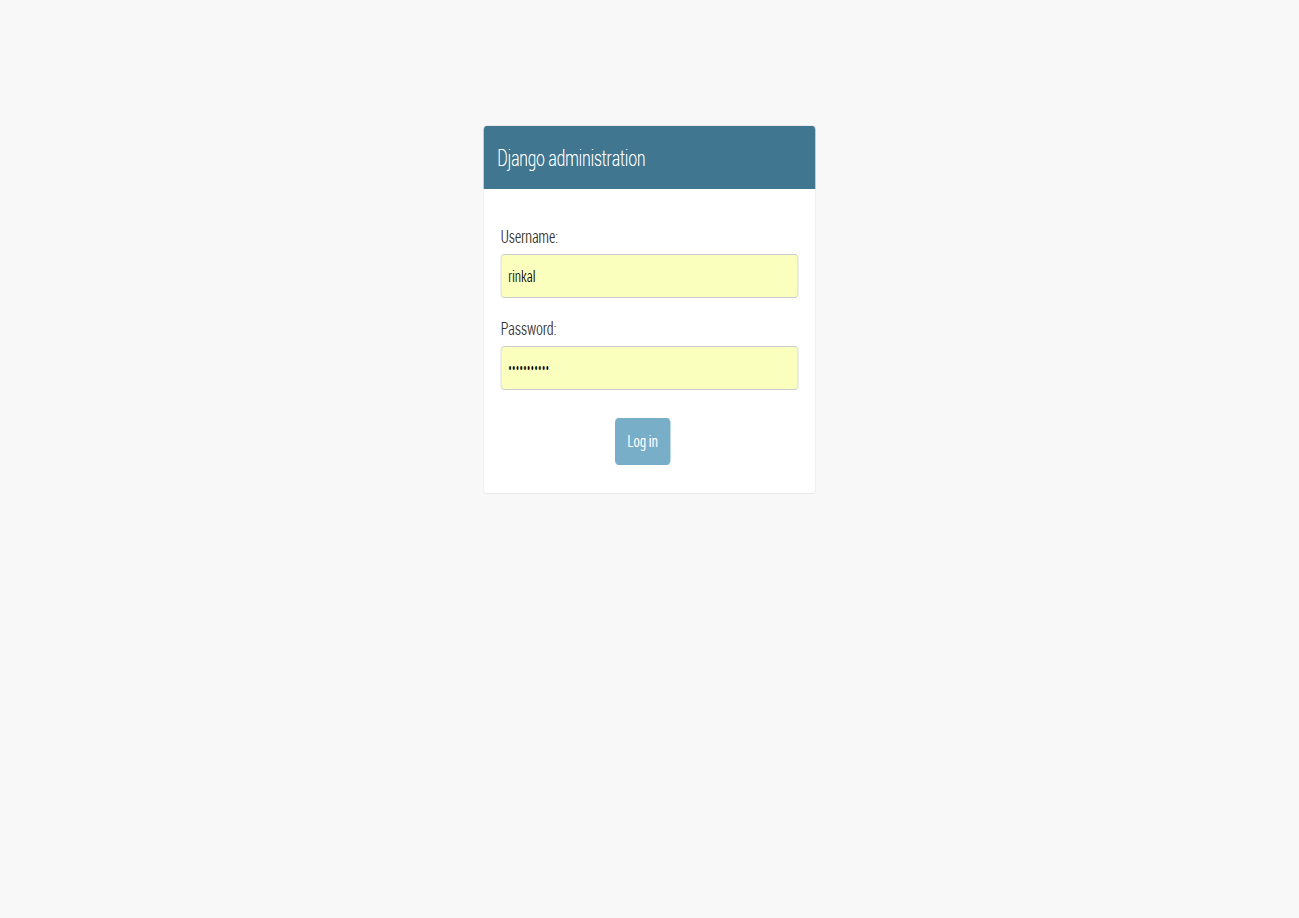
## Appendix B: Operational Manual

#### A. Run the application

1. Click Windows icon + X and click on Command Prompt.
2. Enter command cd Desktop/env3
3. .\Scripts\activate
4. Cd src
5. Python manage.py runserver
6. Open http://127.0.0.1:8000/ in browser

*Figure 13 Run application*

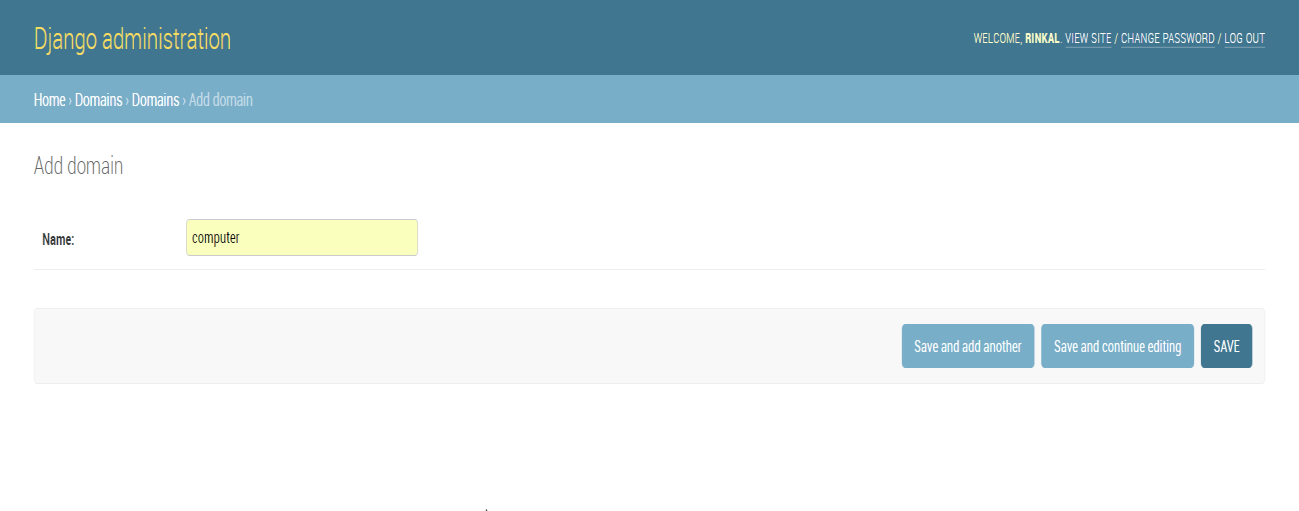
#### Login as Admin

* 1. Go to http://127.0.0.1:8000/admin
  2. Enter admin login information and click on “Login” button.

*Figure 14 Django admin login*

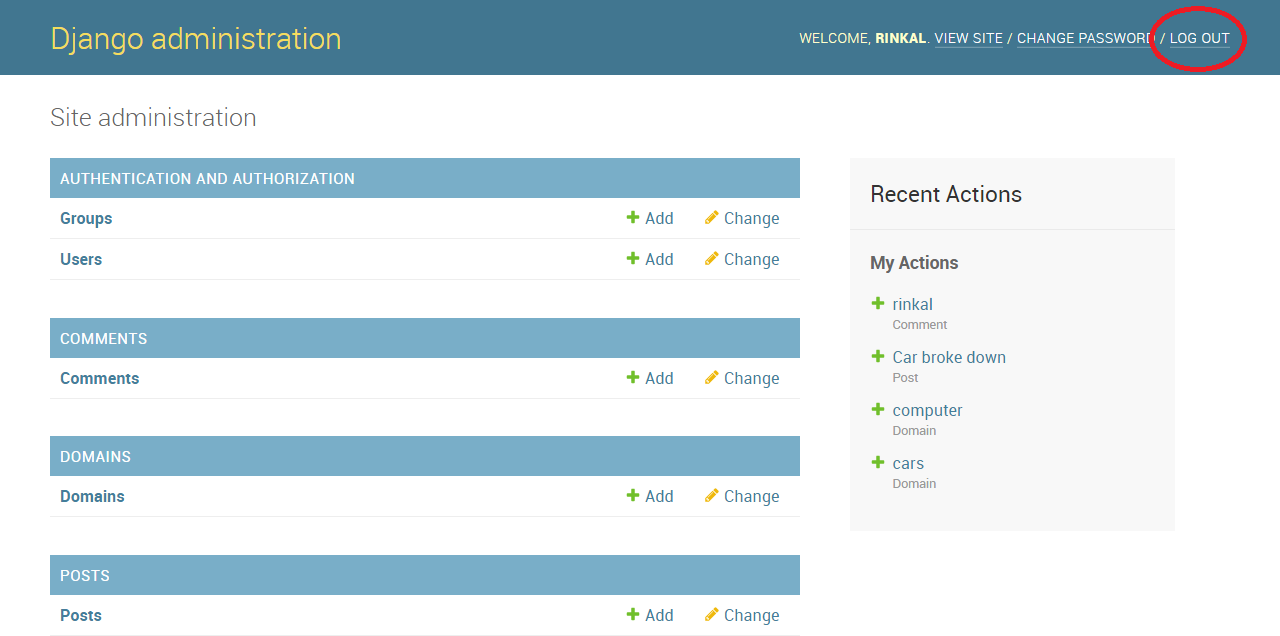
#### Create new Domain

* 1. Click on Domains->Add Domain
  2. Enter Domain name and click on Save button.



*Figure 15 Add new Domain*

#### Logout as Admin

* 1. Click on LOGOUT button on header.

*Figure 16 Logout Admin*

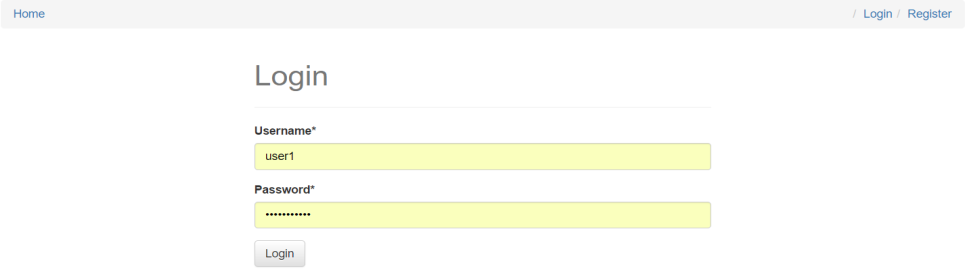
#### Go to User Panel

* 1. Click on “VIEW SITE” link in header.



*Figure 17 Visit User Panel*

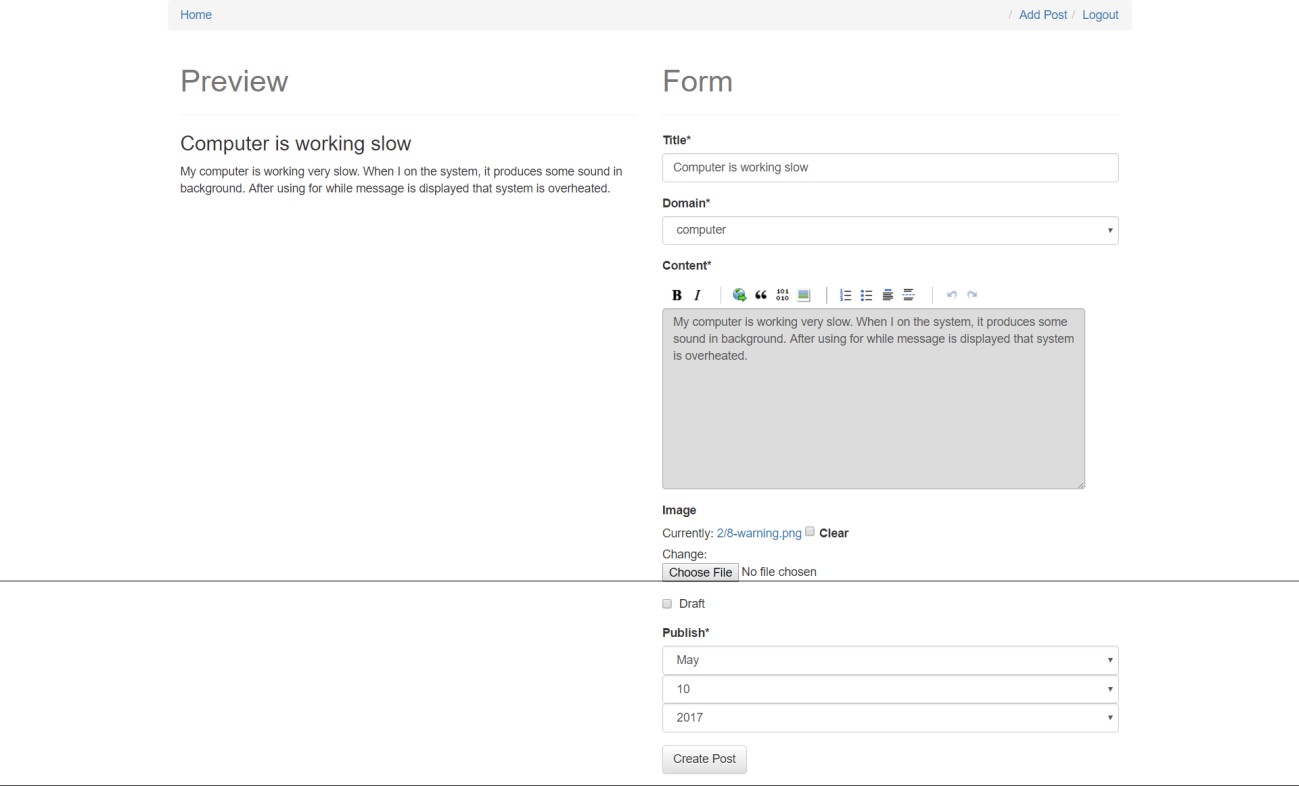
#### Login as User

* 1. Click on Login link in header.
  2. Enter username and password. Click on “Login” button.

*Figure 18 User Login*

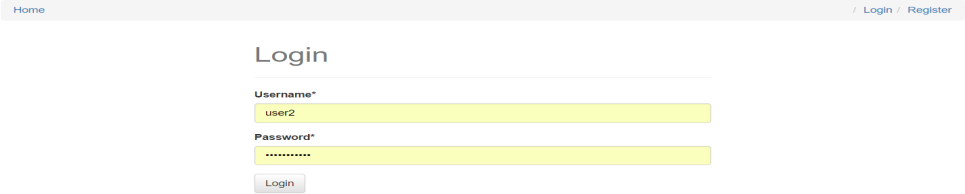
#### Post a Problem

* 1. Click on “Add Post” link.
  2. Input all the fields.
  3. Click on “Create Post”.



*Figure 19 Post Problem*

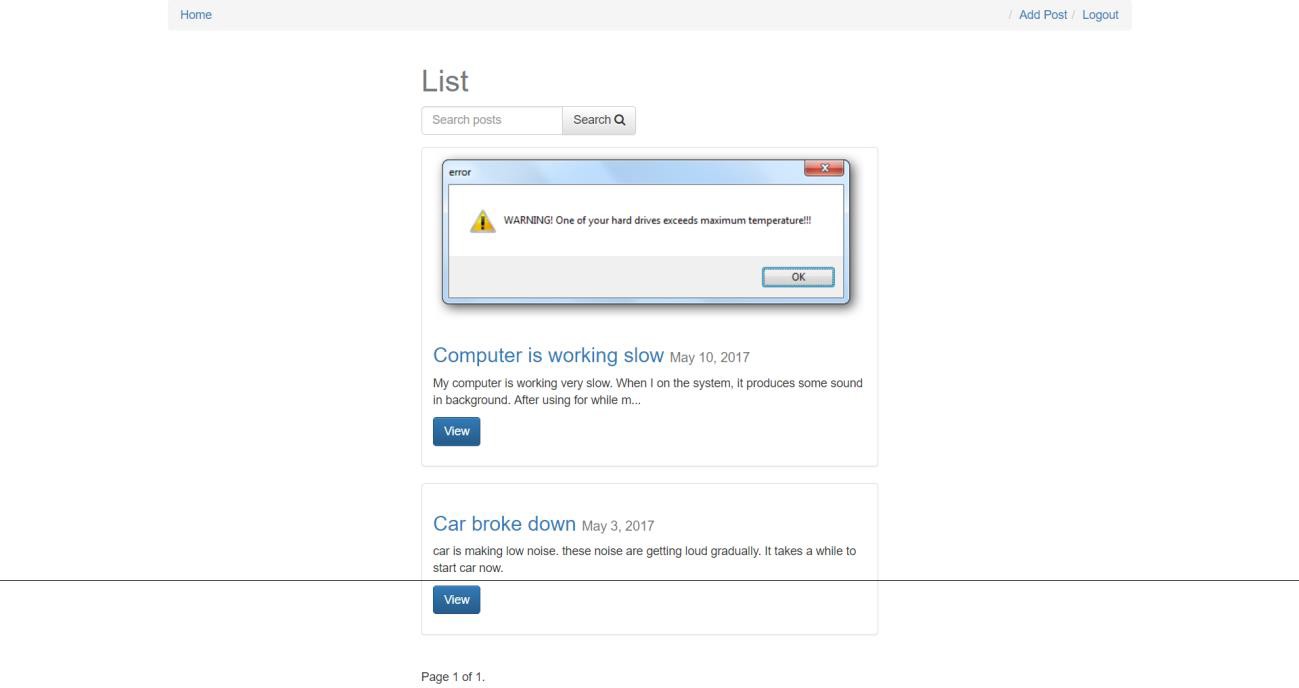
#### Login as Expert

1. Logout out of the system.
2. Click on Login link and input credentials for expert.
3. Click on “Login” button.

*Figure 20 Expert Login*

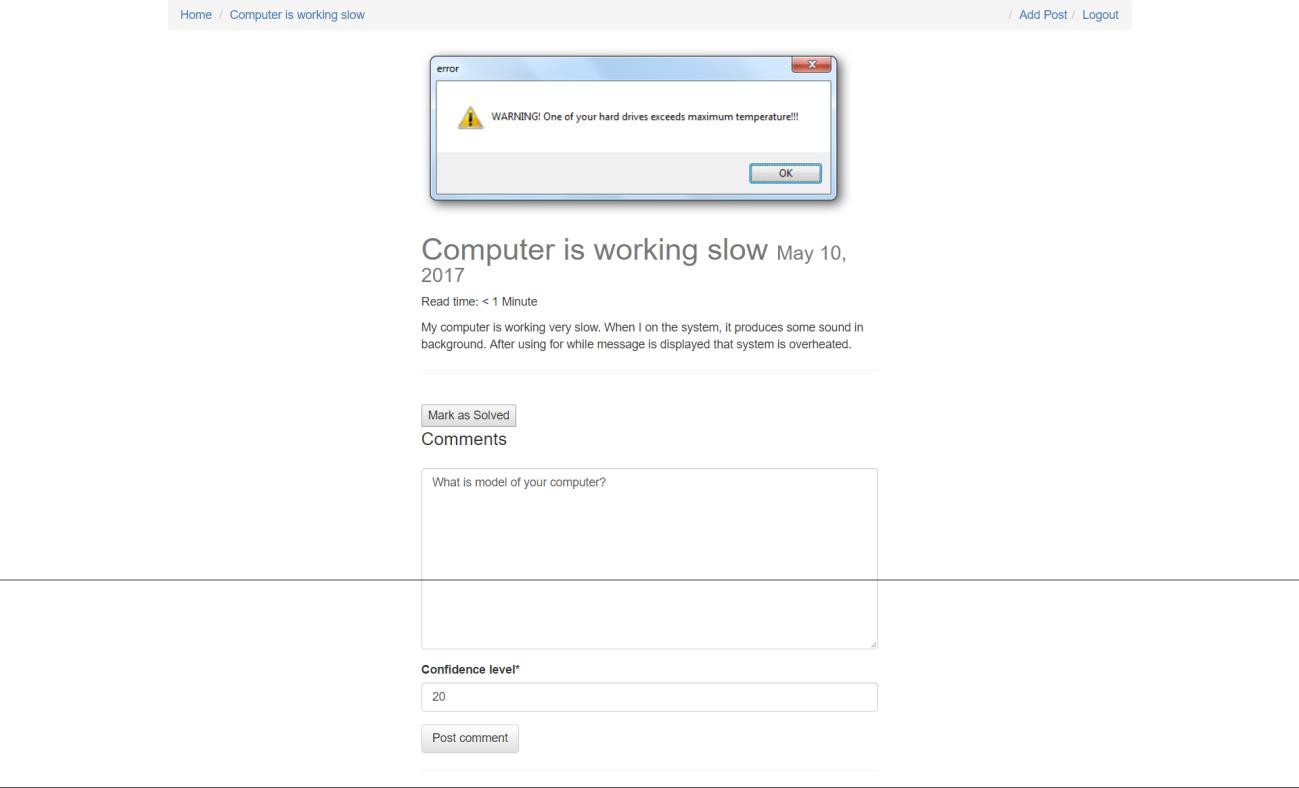
#### View the Problem

* 1. Login and Click on “View” from any post post in displayed list.



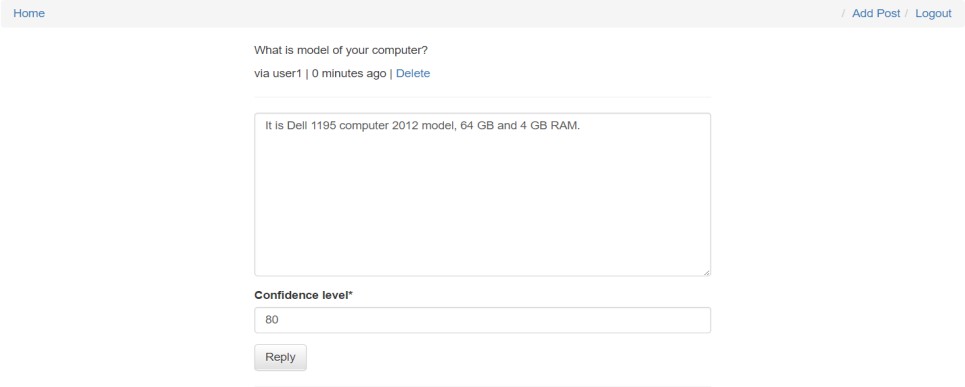
*Figure 21 View Problem*

#### Post question for Problem

* 1. Login as expert and Click on “View” button for any post.
  2. Input question and confidence level and click on “Post Comment”.

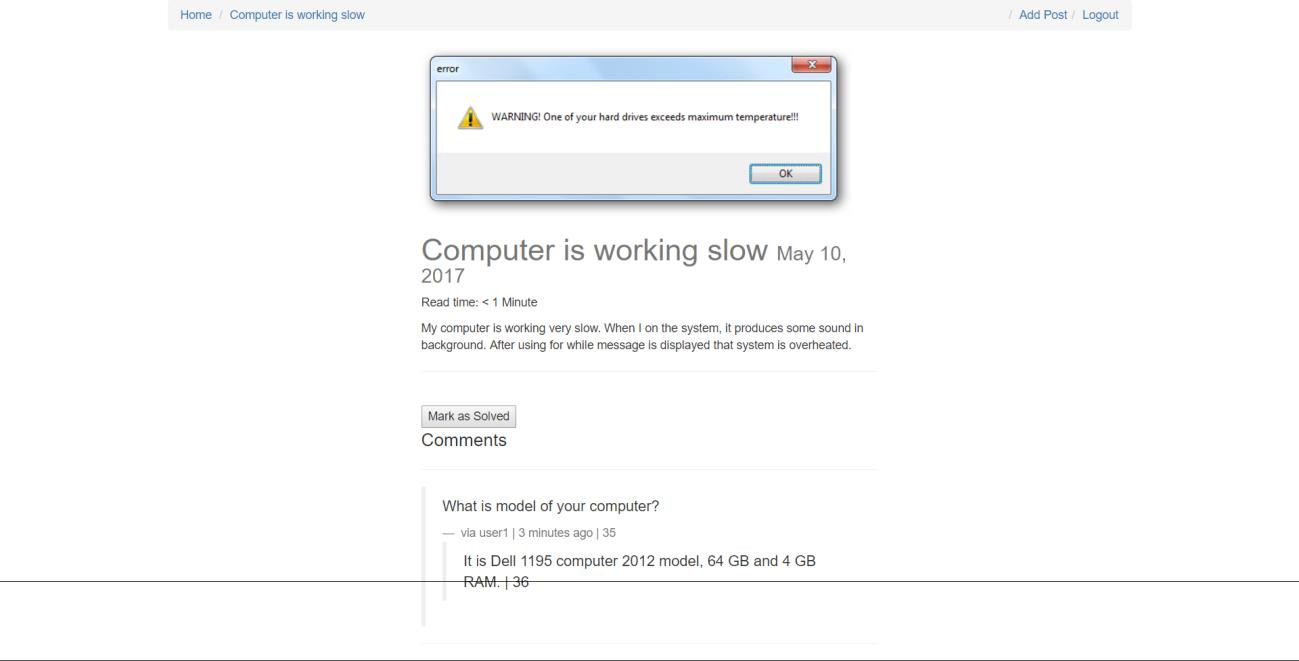
*Figure 22 Post Solution*

#### Give Response to Expert question and verify the solution.

* 1. Login as User.
  2. Click on “View” post button.
  3. Click on “Give Response” button for question.
  4. Enter response and confidence level.
  5. Click on “Reply” button.

*Figure 23 Give Response*

#### Mark Problem as “Solved”

* 1. Login as user and verify the solution given by expert.
  2. Click on “Mark as Solved”.

*Figure 24 Mark Problem as “Solved”*