CraftVerify

Low Level Design Document

Team Wanderer

https://github.com/JPJ-5/Senior-Project

Team Members:
Jason Jitsiripol (Team Leader)
Steven Hall
Parth Thanki
An Nguyen
Khuong Nguyen
Joshua Reyes
Shane Kerr

11/03/2023

Low Level Design Version Table:

Version	**Description**	**Date**
1.0	Initial Low Level Design Requirements ● Feature (Registration, Logging) ○ Sequence Diagrams ○ Classes/Interfaces ○ User Stories	**11/03/2023**

Table of Contents:

IRegistration Interface....................4
User Class....................4
RegistrationManager Class....................4
ILogger Interface....................5
LogEntry Class....................6
LogManager Class....................6

Core Components

Registration

Sequence Diagrams are in a separate document.

Success Scenarios:

- User initiates registration with email and password.
- System validates credentials:
    - Email format is checked.
    - Password strength is checked.
- User account is created with a unique username.
- System checks for persistent storage availability.
- User details are saved to the RDBMS.
- User receives a success message within 5 seconds.

Failure Scenarios:

- Invalid email or password:
    - System provides correction suggestions.
- Username assignment failure:
    - System suggests retrying or contacting support.
- Persistence failure:
    - Error is logged and the admin is notified.
- Performance issues:
    - Registration time over 5 seconds is logged.

Classes/Interface

IRegistration Interface

- Methods:
    - validateCredentials(email, password): Validates the provided email and password.
        - checkEmailFormat(email): Checks if the email adheres to a standard email format.
        - checkEmailUniqueness(email): Ensures the email is not already registered.
        - checkPasswordStrength(password): Verifies password against strength requirements.
    - createAccount(userDetails): Creates a new user account.
        - assignUsername(userDetails): Generates a unique username.
        - hashPassword(password): Hashes the password for secure storage.
    - saveAccountDetails(account): Saves account details to the database.
        - insertUserRecord(account): Inserts a new user record into the database.
        - sendConfirmationEmail(email): Sends a registration confirmation email to the user.

User Class

- Attributes:
    - email: The email address of the user.
    - password: The password for the user account.
    - username: A system-wide unique username.
- Methods:
    - validateEmail(): Validates the email address against standard and custom rules.
    - validatePassword(): Validates the password strength and compliance with security policies.
    - generateUsername(): Generates a unique username based on the email and other factors.

RegistrationManager Class

- Attributes:
    - storage: Reference to the persistent storage interface.
- Methods:
    - handleRegistration(email, password): Orchestrates the user registration process.

        - initiateValidationProcess(email, password): Starts the credential validation process.
        - completeRegistration(): Finalizes the registration process after all checks pass.
    - notifyUser(message): Notifies the user of the success/failure of registration.
    - notifyAdmin(message): Notifies the admin of any system-level issues during registration.

User Stories

- As a Prospective User:
    - If I mistakenly enter my email or username incorrectly, I want the system to:
        - Suggest corrections based on common typos or domain misspellings.
        - Prevent me from accidentally creating multiple accounts due to typos.
    - In case I'm located in a region with different character sets, I expect:
        - Full support for Unicode in all my registration details.
        - Accurate handling of name suffixes and titles that are part of my cultural norms.
- As a System Administrator:
    - I need to be made aware of when:
        - Multiple registration attempts are detected from the same IP in quick succession.
        - Registration details match known patterns of fraudulent behavior.
    - It's crucial that I can:
        - Easily update blacklists or rules to prevent registrations from suspicious sources.
        - Detect and investigate any irregularities in the registration process, like bulk account creations.
- As a Developer:
    - I want the system to be able to:
        - Handle edge cases where input fields are left blank or filled with special characters that could be used in an injection attack.
        - Reject email addresses from disposable email services.
    - The system should be resilient enough to:
        - Provide informative feedback when unexpected server errors occur during registration.
        - Gracefully recover from partial registration states, such as a crash happening between account creation and email confirmation.

Logging

Sequence Diagrams are in a separate document.

Success Scenarios:

- Event occurs and is captured.
- Event data is validated (type, format, timestamp).
- Log entry is created with a validated timestamp, log level, and category.
- System checks persistent storage availability.
- Log entry is saved to the RDBMS within 5 seconds.
- System confirms successful logging without user interaction delay.

Failure Scenarios:

- Delayed logging over 5 seconds is logged as an error.
- User interaction delay due to logging is recorded as an error.
- Persistence issues are logged and the admin is notified.
- Invalid log data results in an error and admin notification.
- Log modification attempts are recorded and reported.
- Archiving issues are logged and the admin is notified.

Classes/Interface

ILogger Interface

- Methods:
    - logEvent(event): Abstract method to log an event.
        - formatMessage(event): Formats the event message for logging.
        - determineLogLevel(event): Determines the log level based on the event type.
    - archiveLogs(): Abstract method to archive old logs.

        - selectLogsForArchival(): Selects logs that are eligible for archiving.
        - moveLogsToArchiveStorage(): Moves selected logs to archival storage.

LogEntry Class

- Attributes:
    - timestamp: UTC timestamp of the log entry.
    - logLevel: The severity level of the log (Info, Debug, Warning, Error).
    - category: The category of the log (View, Business, Server, Data, Data Store).
    - userActionOrSystemEvent: Description of the user action or system event.
    - message: A message describing the situation.
- Methods:
    - serialize(): Converts the log entry into a JSON or text format for storage.
        - toJSON(): Serializes the entry into JSON format.
        - toString(): Serializes the entry into a plain text format.
    - validateEntry(): Validates the log entry data before saving.
        - checkTimestamp(): Validates the timestamp format and accuracy.
        - checkDataIntegrity(): Ensures the log entry has not been tampered with.

LogManager Class

- Attributes:
    - storage: Reference to the persistent storage interface.
- Methods:
    - handleEvent(event): Captures and processes logging of events.
        - createLogEntry(event): Creates a new log entry object from the event.
        - saveLogEntry(logEntry): Saves the log entry to the database.
    - checkStorage(): Ensures the availability and sufficiency of the storage.
    - notifyAdmin(message): Alerts the admin in case of logging failures.

User Stories

- As a System Administrator:
    - I expect the logging system to:
        - Capture attempts to modify or delete logs and alert me immediately.
        - Alert me when the volume of logs is significantly higher than normal, which might indicate an underlying issue.
    - The system should allow me to:
        - Quickly locate and review logs related to security incidents.
        - Have an automated response for logging system failures, like fallback to a secondary logging system.
- As a Developer:
    - I need the logging system to:
        - Correctly handle log entries that contain characters from various languages.
        - Maintain log integrity, even when the log entry contains unexpected or malformed data.
    - It's important for the system to:
        - Provide mechanisms to test logging reliability and performance under stress conditions.
        - Log detailed error information when a user encounters a system exception, without exposing sensitive information.
- As a User:
    - When I'm using the system, I assume that:
        - Any personal information in logs is anonymized or encrypted to protect my privacy.
        - If an error occurs during my interaction, detailed information will be logged to facilitate a quick resolution, without compromising my data.
    - If I decide to delete my account or remove certain data, I want:
        - Confirmation that actions related to data deletion are securely logged for accountability.
        - Assurance that my deleted data isn't recoverable from the logs.