Swami Keshvanand Instituteof Technology, Management & Gramothan, Jaipur

FACULTY KIT

Objective–

An agent-based push mechanism is a dynamic system that uses intelligent software agents to deliver relevant and timely information to users proactively. Unlike pull-based systems, where users must request information, the push mechanism anticipates user needs based on predefined criteria, user behavior, or real-time context. This approach enhances efficiency by reducing the user's effort in searching and retrieving information. In such a system, agents act as intermediaries that monitor data sources, filter content, and deliver it to users. These agents are equipped with adaptive learning capabilities, enabling them to refine their recommendations over time based on user feedback or evolving preferences.
Additionally, they leverage technologies like machine learning, natural language processing, and user profiling to understand and predict user needs effectively.
Applications of agent-based push mechanisms span various domains, including personalized content delivery in e-commerce, real-time updates in financial markets, and proactive alerts in healthcare. The system also supports distributed environments by enabling communication between multiple agents for coordinated decision-making. By automating information dissemination and tailoring content to individual needs, agent-based push mechanisms enhance user experience, optimize resource utilization, and support timely decision-making in a rapidly evolving digital landscape..

Requirements Specification–

The**Agent Based Push Mechanism** willinclude:

- **User and Admin Roles:**
  The system supports two roles: Admin (manages agents and rules) and Users (receive push notifications)**.**
- **Agent-Based Automation**
  Admin can create agents that automatically send push notifications based on specific user actions or system events (e.g., login, form submission).
- **Rule Engine for Conditions**
  Each agent includes rules (conditions and triggers) that define when and to whom the notifications should be sent.
- **Admin Dashboard**
  Admin has a dashboard to create, manage, activate/deactivate agents, and view notification logs and performance stats.
- **Audit and Logs**
  All admin activities and triggered push events are logged for monitoring and debugging purposes.

Technology Familiarization–

The Agent-Based Push Mechanism project utilizes modern technologies for real-time communication between users and admin. The frontend is built using HTML5, CSS3, and React.js (or similar), while the backend uses Node.js with Express or Django/Laravel for rule processing and API management. Data is stored in MySQL or PostgreSQL, with optional use of MongoDB for logs. JWT ensures secure authentication. Docker is used for deployment, along with cloud platforms like AWS or GCP. Admins define agents and rules, which are triggered by CRON jobs or event listeners to push messages.

Database Creation–

The **Agent Based Push Mechanism** willutilizebothrelationalandnon-relational databases to handle different data types:

- **Users** – Stores user details (ID, name, email, role, preferences).
- **Admins** – Stores admin credentials and activity logs.
- **Agents** – Contains agent configurations, triggers, and status (active/inactive).
- **Logs** – Records agent executions and delivery outcomes.

High Level and Detailed Design

System Overview

The agent-based push mechanism is designed to push updates or notifications to users in real-time. This system ensures that users receive timely updates without needing to refresh their web page manually. The architecture of the system is built using HTML, CSS, JavaScript, PHP, and MySQL to provide an efficient, real-time push notification mechanism. Below is a breakdown of how each component interacts in the system:

1. **Frontend (HTML, CSS, JavaScript):**
    - **HTML** is used to structure the page, creating the necessary layout to display content.
    - **CSS** is used for styling the page, ensuring the notifications are presented in a user-friendly manner.
    - **JavaScript** will manage the frontend logic. It will handle receiving push notifications and displaying them in real-time using technologies like WebSockets or long polling.
2. **Backend (PHP, MySQL):**
    - **PHP** handles the backend logic. It serves as the intermediary between the frontend and the database. PHP scripts will manage authentication, user data retrieval, and triggering of push notifications.
    - **MySQL** is used for storing user data, push notification records, and history, ensuring that only authorized users receive notifications and that the notifications are logged.
3. **Database (MySQL):**
    - **MySQL** stores user information such as login credentials, preferences. It ensures that users only receive relevant notifications.

Detailed Design

2.1 Frontend Design

The frontend is responsible for displaying data that is pushed from the backend and handling real-time communication. It's implemented using technologies like **HTML**, **CSS**, and **JavaScript**.

- **HTML**: The HTML defines the static structure of the page, including the layout and the areas where dynamic content will be displayed (such as a live updates section).
- **CSS**: CSS is used for styling the page, including the appearance of dynamically pushed content. It ensures that the content is displayed neatly and responsively across different screen sizes.
- **JavaScript**: JavaScript manages the client-side logic, including receiving the pushed data and updating the UI without requiring a page reload
- The frontend will dynamically update the webpage to show new data, such as a real-time update in a data feed or live status updates, without the user needing to refresh the page.

2.2 Backend Design

The backend is responsible for managing system events, processing business logic, and facilitating communication between the frontend and the database.

- **Event Management**: The backend listens for certain triggers or changes within the system, such as new data entries, status updates, or other system events that require pushing data to the user. The backend processes these triggers and prepares the data to be sent to the frontend.
- **Push Agent**: The agent is a continuously running process on the server that monitors for specific conditions or events that require real-time updates. For instance, it could monitor for new data entries in the database or listen for changes in the system state. When an event occurs, the agent prepares the relevant data and pushes it to the connected clients..

2.3 Push Agent

The **Push Agent** is the core component of the system that makes the push mechanism "agent-based." The agent runs as a process on the server, constantly checking for changes or events that should trigger a data push to the clients. These events could be changes in the database, updates from other systems, or manually triggered updates.

- **Event Detection**: The agent continually monitors the system for changes. For example, if the system is tracking user activity, the agent could detect when a user completes an action (like submitting a form or making a request), which then triggers the push.
- **Data Preparation and Distribution**: When the agent detects an event, it prepares the necessary data (such as a new message or status update) and sends this information to the frontend.

2.4 Database Design

The database is where persistent information is stored. In the context of the push mechanism, the database stores data related to users, events, or any system state that could trigger updates.

- **Event Logs**: The database contains a log of events, actions, or system statuses that can be monitored by the push agent. For instance, the database might have a table that tracks user activity or a queue that holds data waiting to be pushed to clients.
- **Data Retrieval**: When the agent detects an event, it retrieves relevant data from the database. The backend uses this data to generate the message or information that will be sent to the client.
- **Database Structure**: The structure of the database depends on the specific application. It could include tables like:

- Users table, which stores user information.
- Events table, which stores information about triggered events.
- Logs table, which keeps a history of the data pushed to clients.

Test Plan and Final Review

The test plan for the agent-based push mechanism involves validating push notifications between Admin and User roles. Key areas include testing notification delivery, user acknowledgment, and role-based permissions. The Admin should be able to send notifications to users, view logs, and handle multiple formats (text, images, links). User responses and security, such as encryption and access control, will be checked. Performance testing will assess system load under heavy traffic, while error handling ensures reliability. Security tests will focus on authentication and data protection. The plan includes manual and automated testing, with regression and performance validation for stability.

References and Documents used

- **HTML -** https://www.w3schools.com/Html/ \
- **CSS -** https://www.w3schools.com/css/
- **Javascript -** https://javascript.info/
- **PHP -** https://www.w3schools.com/php/
- **MySQL -** https://www.w3schools.com/MySQL/default.asp

Conclusion

The agent-based push mechanism project successfully facilitates seamless communication between Admin and User roles, ensuring efficient notification delivery and acknowledgment. By implementing role-based permissions, the system guarantees secure and personalized notifications, with users receiving only relevant content. Extensive testing, including performance, security, and error handling, validated the system's reliability under varying loads and potential failures. The project ensures real-time interaction, robust encryption, and an audit trail for accountability. With a focus on user experience and system stability, the solution is poised for production, offering a scalable and secure push notification platform for Admin and User engagement.