**Research Document**

***Customer****:*

*Maurice Lamme*

|  |
| --- |
| **Date** : 15/06/2024 |
| **Version** : 1.0 |
| **State** : Finished |
| **Author** : Group 1 |

**Version Control**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Author(s)** | **Changes** | **State** |
| 1.0 | 15/06/2024 | Catalin Mihai Popoiu, Aleksandar Yordanov | Created research document | Finished |

**Table of contents**

1. **Research on File Storage Methods** 4
   1. **Introduction** 4
      1. Purpose 4
      2. Scope 4
   2. **File Storage Methods** 4
      1. Azure Blob Storage 5
      2. Local File Storage 5
      3. Database Storage 5
   3. **Comparison of File Storage Methods** 5
      1. Performance 5
      2. Scalability 6
      3. Cost 6
      4. Security 6
   4. **Conclusion and Recommendation** 6
      1. Conclusion 6
      2. Recommendation 7
   5. **Implementation Strategy**  7
2. **Research on Automated Calendar Notifications for Meetings**
   1. **Introduction** 9
   2. **Current Implementation** 9
      1. Subscription Mechanism 9
      2. Notification Scheduling 9
      3. Notification Process 9
   3. **Alternative Methods** 10
      1. Microsoft Graph API 10
      2. Backend Calendar Creation with Teams Integration
   4. **Comparative Analysis** 11
   5. **Conclusion and Recommendations** 11

**Research on File Storage Methods**

1. **Introduction**
   1. **Purpose**

This research document aims to provide a thorough analysis of various file storage methods, including Azure Blob Storage, local file storage, and database storage. The goal is to identify the most suitable storage solution for our specific use case, which involves storing, uploading, downloading, and updating presentation files (PDFs), considering factors such as performance, scalability, cost, and security.

* 1. **Scope**

The research focuses on three primary file storage methods:

* Cloud storage using *Azure Blob Storage*
* Local file storage on a *server*
* Storing files directly in the *database*

The analysis will highlight the pros and cons of each method and offer recommendations based on our project’s requirements.

1. **File Storage Methods**
   1. **Azure Blob Storage**

Azure Blob Storage is a cloud-based service for storing large amounts of unstructured data, such as text or binary data.

**Pros:**

* **Scalability**: Virtually unlimited storage capability.
* **Availability**: High availability with redundancy options (e.g., LRS, GRS).
* **Cost-Effective**: Pay-as-you-go pricing model, reducing upfront costs.
* **Security**: Advanced security features, including encryption and access control.

**Cons:**

* **Dependency on Internet**: Requires a stable internet connection for access.
* **Latency**: Potential latency issues compared to local storage.
* **Complexit**y: Requires configuration and understanding of Azure services.
* **Cost**: Incurs ongoing costs, unsuitable for zero-budget projects.
  1. **Local File Storage**

Local file storage refers to storing files on physical servers or storage devices within an organization’s infrastructure.

**Pros:**

* **Performance:** High-speed access due to local network storage.
* **Control:** Full control over hardware and data management.
* **Simplicity:** Easier to set up and manage for small-scale operations.
* **Cost:** No ongoing cost, making it ideal for projects with zero budget.
* **Accessibility:** Easily accessible by all team members when set up on each member’s local machine.
* **Security:** When files are placed in a private folder accessed through secure backend endpoints, it adds an extra layer of security.

**Cons:**

* **Scalability:** Limited by physical hardware capacity.
* **Maintenance:** Requires hardware management.
  1. **Database Storage**

Storing files directly in a database involves saving binary data (BLOBs) in database tables.

**Pros:**

* **Integration:** Simplifies data management with transactional consistency.
* **Security:** Leverages database security features and access controls.
* **Backup:** Streamlined backup processes as part of database management.

**Cons:**

* **Performance:** Can degrade database performance due to large data volumes.
* **Complexity:** Requires careful schema design and management.
* **Bloat:** Potentially bloats the application, impacting performance and manageability.

1. **Comparison of File Storage Methods**
   1. **Performance**

* **Azure Blob Storage:** Suitable for high availability and large-scale data but may introduce latency.
* **Local File Storage:** Offers the best performance for local access but limited by hardware capacity.
* **Database Storage:** Can impact database performance, suitable for small to medium-sized files.
  1. **Scalability**
* **Azure Blob Storage:** Highly scalable with virtually unlimited capacity.
* **Local File Storage:** Limited by physical storage capacity, requires hardware upgrades.
* **Database Storage:** Scalable but may require complex database sharding and management.
  1. **Cost**
* **Azure Blob Storage:** Cost-effective as you pay-as-you-go model but variable based on usage.
* **Local File Storage:** Initial investment in hardware, but low ongoing costs.
* **Database Storage:** Potentially high costs due to database storage and managements.
  1. **Security**
* **Azure Blob Storage:** Advanced security features with encryption and access controls.
* **Local File Storage:** Security depends on in-house measures and physical security. When combined with backend access controls, security is enhanced.
* **Database Storage:** Leverages robust database security but requires careful management.

1. **Conclusion and Recommendation**
   1. **Conclusion**

After a thorough evaluation of the available file storage methods, it was determined that local file storage is the most suitable solution for our project. The reasons for this choice are as follows:

* **Zero Budget:** Azure Blob Storage, while highly scalable and secure, incurs ongoing costs, making it unsuitable for a project with no budget.
* **Application Bloat:** Storing files directly in a database would lead to application bloat and degrade performance, making it impractical for our needs.
* **Performance and Control:** Local file storage provides the best performance for local access and gives us full control over our data and hardware.
* **Accessibility:** Setting up local file storage on each team member’s local machine ensures that all team members can easily access the files.
* **Enhanced Security:** Placing files in a private folder and accessing them through secure backend endpoints adds an extra layer of security.

Local file storage will enable us to efficiently manage the storage of presentation files (PDFs), facilitating their upload, download, and update without incurring additional costs.

* 1. **Recommendation**

Given the constraints and requirements of our project, local file storage is recommended as the primary file storage method. This approach will allow us to maintain high performance and control without incurring additional costs. To ensure successful implementation, the following steps are proposed:

* **Hardware Assessment:** Assess current hardware to ensure sufficient capacity and performance for local file storage.
* **Security Measures:** Implement robust security measures and secure backend endpoints, to protect stored data.
* **Maintenance Plan:** Develop a maintenance plan to ensure regular updates and hardware checks.
* **Scalability Considerations:** Plan for future scalability by considering potential hardware upgrades and additional storage solutions.

1. **Implementation Strategy**

* **Upload Functionality:** Develop a user-friendly interface for uploading presentation files, ensuring organized and accessible storage.
* **Download Functionality:** Implement efficient retrieval mechanisms for quick and reliable file downloads.
* **Update Functionality:** Ensure the system supports file updates, maintaining version control and data integrity.
* **Team Accessibility:** Configure local file storage to be accessible by all team members, ensuring seamless collaboration.
* **Backend Integration:** Ensure secure access to the private folder through backend endpoints to enhance file security.

Implementation Plan:

* **Backend Development:**
  + Set up a private folder structure for file storage.
  + Develop APIs in Spring Boot to handle file uploads, downloads, and updates, ensuring secure access through backend endpoints.
* **Frontend Development:**
  + Implement the user interface for uploading, downloading, and updating presentation files using React.js.
  + Integrate the frontend with backend APIs to enable seamless file operations.

By leveraging local file storage, we can efficiently manage our storage needs within budget constraints, ensuring high performance and control over our data management processes, specifically for handling presentation files (PDFs), while ensuring accessibility and security for all team members.

**Research on Automated Calendar Notifications for Meetings**

1. **Introduction**

In today’s fast paced business environment, efficient communication and scheduling are paramount. This research document explores the methods of sending automated calendar notifications through email for meeting invitations from an application. The primary focus is on a scenario where users subscribe to a calendar feature within an app, and subsequently receive invitations for all upcoming meetings. The research covers the existing implementation using Quartz and Power Automate and explores potential alternatives such as Microsoft Graph API and backend calendar creation with MS Teams integration.

1. **Current Implementation**
   1. **Subscription Mechanism**

* Users subscribe to the calendar function within the app.
* Upon subscription, users are added to a list of recipients who will receive meeting invitations.
  1. **Notification Scheduling**
* Invitations are scheduled to be sent twice a year.
* Quartz scheduler is utilized to manage the timing of these notifications.
  1. **Notification Process**
* An email containing all recipient and meeting dates is sent to a designated email account associated with the app.
* A Power Automate flow is triggered by the receipt of this email.
* The flow processes the email and sends out individual meeting invitations to each recipient.

1. **Alternative Methods**
   1. **Microsoft Graph API**

* **Overview**: Microsoft Graph API provides a unified endpoint to access Microsoft 365 services, including Outlook, Calendar, and Teams.
* **Implementation**:
  + **User Subscription**: When a user subscribes, their details are stored, and necessary permissions are granted via OAuth2.
  + **Scheduling Notifications**: Use the Graph API to create calendar events directly in users’ calendars.
  + **Automated Invitations**: The API can send meeting invitations directly to users’ Outlook calendars, ensuring seamless integration and real-time updates.
* **Benefits:**
  + Real-time updates and notifications.
  + Direct integration with users’ calendars, reducing manual steps.
  + Enhanced security and compliance with Microsoft’s infrastructure.
* **Challenges:**
  + Requires thorough understanding of Microsoft Graph API.
  + Initial setup and OAuth2 permission management can be complex.
  1. **Backend Calendar Creation with Teams Integration**
* **Overview:** This method involves creating and managing calendars and Teams meetings directly from the backend of the application.
* **Implementation:**
  + **Calendar Creation:** Use Microsoft Graph API or direct backend logic to create a unified calendar for meetings.
  + **Teams Integration:** Attach Teams meeting links to calendar events for virtual meetings.
  + **Automated Notifications:** Use backend logic to trigger invitations and updates.
* **Benefits:**
  + Unified calendar management with Teams integration.
  + Streamlined process for scheduling and notifications.
  + Enhanced user experience with direct access to Teams meetings.
* **Challenges:**
  + Complexity in backend integration.
  + Requires robust error handling and user management.

1. **Comparative Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | Current Implementation (Quartz & Power Automate) | Microsoft Graph API | Backend Calendar Creation with Teams |
| Setup Complexity | Moderate | High | High |
| Real-time Updates | No | Yes | Yes |
| Direct Calendar Integration | Yes | Yes | Yes |
| Scalability | Moderate | High | High |
| Security | Moderate | High | High |
| User Experience | Moderate | High | High |

1. **Conclusion and Recommendations**
   1. **Conclusion**

The current implementation using Quartz and Power Automate is functional but has limitations in terms of real-time updates. Transitioning to using Microsoft Graph API or backend calendar creation with Teams integration offers significant advantages in terms of user experience, real-time updates, and security. However, these alternatives come with increased complexity in setup and management. A detailed cost-benefit analysis and consideration of the organizational capabilities should guide the decision-making process.

* 1. **Recommendations**
* **Long-term**: Plan for a phased transition to Microsoft Graph API or backend calendar creation with Teams integration, focusing on training and infrastructure readiness.

***References***

Microsoft Graph API documentation: [Microsoft Docs](https://docs.microsoft.com/en-us/graph/overview)

Quartz Scheduler documentation: Quartz Docs

Power Automate documentation: [Microsoft Docs](https://docs.microsoft.com/en-us/power-automate/)

This research document aims to provide a comprehensive overview and guide for improving the automation of calendar notifications for meetings, enhancing efficiency and user satisfaction.