# SelfMemo3: Online Email Reminder Service

Celine Florian, Stephan Robinig, Piotr Siewiera, and Nina Tschikof

706.041 Information Architecture and Web Usability 3VU WS 2025/2026
Graz University of Technology

31 Jan 2026

## Abstract

SelfMemo3 is a self-hosted email reminder management system, designed to address the growing dearth of usable, free reminder solutions in the modern digital landscape. As commercial reminder applications have increasingly become laden with intrusive advertisements, monetization schemes, and feature bloat, users seeking simple and reliable reminder functionality find themselves confronted with degraded user experiences and privacy concerns. Many popular platforms now prioritize revenue generation over core functionality, embedding persistent advertisements, implementing restrictive paywalls for basic features, and collecting extensive user data for targeted marketing purposes.

This project responds to these challenges by providing a clean, ad-free alternative that places user autonomy and data ownership at its center. Built on modern web technologies including Next.js, React, and PostgreSQL, SelfMemo3 delivers a comprehensive reminder system with flexible scheduling capabilities, recurring reminder support, and customizable notification preferences. By adopting a self-hosted architecture, the application ensures that users maintain complete control over their data, while eliminating dependency on third-party services that may change terms, introduce fees, or discontinue operations.

# Contents

# List of Figures

# List of Listings

# Chapter 1

# Introduction

SelfMemo3 represents a significant evolution in self-hosted reminder management, addressing both the increasing commercialization of productivity software and the technical limitations of previous implementations. This version marks a departure from cloud-dependent architectures, offering users complete control over their reminder infrastructure without reliance on external service providers. The project is open source and can be found on GitHub [Florian et al. 2026].

This iteration of SelfMemo introduces substantial enhancements to both deployment flexibility and reminder functionality. Previous versions of SelfMemo relied on external services for critical components of the application infrastructure. SelfMemo3 achieves true independence by supporting local deployment of all system components. The application now runs on standard server infrastructure with PostgreSQL for data persistence and a system-level cron for task scheduling, eliminating external service dependencies that previously complicated deployment and introduced ongoing operational costs.

The reminder system itself has been expanded to accommodate more sophisticated use cases:

- Reminders can now be configured to repeat until a specified end date, providing automatic termination for time-bound recurring tasks without manual intervention.
- Individual reminders can be distributed to multiple recipients, enabling shared task management and collaborative reminder workflows within households or small teams. Also, notifications can be sent when the specific reminder has been updated.
- Reusable reminder templates streamline the creation of frequently used reminder configurations, reducing repetitive data entry for common reminder patterns.
- Full time zone support ensures that reminders trigger at the intended local time for users, regardless of server location, which is critical for globally distributed deployments or users who travel frequently.
- A dedicated calendar view provides a graphical visualization of upcoming reminders, offering improved temporal awareness and helping users identify scheduling conflicts or reminder density across time periods.

# Chapter 2

# Email Reminder Services

There are a number of existing free-to-use email reminder services. Three of them are presented here: MemoToMe, Simply Remind Me, and SelfMemo 2.0, the immediate predecessor to this project.

## 2.1 MemoToMe

MemoToMe was a web-based email reminder service that allowed users to schedule both one-time and recurring email notifications. The service is no longer supported, nor is it accessible through the Internet. The MemoToMe user interface is shown in Figure 2.1. The service provided comprehensive scheduling functionality, enabling users to configure reminders based on either exact timestamps or time intervals. It supported multiple recurrence patterns, including one-time, daily, weekly, every few weeks, monthly, and yearly intervals. Additionally, the platform offered warning reminders to alert the user in advance of scheduled events. The application featured a dashboard displaying an overview of all configured reminders, supplemented by basic analytical tools that allowed users to examine their reminder usage patterns.

## 2.2 Simply Remind Me

Simply Remind Me is a comprehensive web-based reminder scheduling service offering both free and premium subscription tiers [Heathershaw 2021]. The free version, similar to MemoToMe, allows users to schedule email reminders with precision to the nearest minute, and reminders can be repeated daily, weekly, monthly, or yearly. The Simply Remind Me user interface is shown in Figure 2.2. The premium tier extends the base functionality by offering white-labeled email and text reminders, reminder status tracking, and the ability to distribute reminders to multiple recipients. Text and phone call reminders are also supported; they are included in the premium subscription and require an additional payment in the free tier.

## 2.3 SelfMemo 2.0

SelfMemo 2.0 is the immediate predecessor to this work, and a successor of a SelfMemo 1.0 project. Both applications were developed by students of the Information Architecture and Web Usability course in winter semester 2023/2024 and winter semester 2024/2025. SelfMemo 2.0 is an open-source, self-hosted email reminder service [Lorber et al. 2025], built with Next.js [Vercel 2024]. It was designed to replace discontinued services, like MemoToMe. Figure 2.3 shows the reminder creation process in SelfMemo 2.0.

The application's functionality included:

- Scheduling reminders, including various types of reminders, such as: one-time, daily, weekly, n-weekly, monthly, yearly, and n-yearly.

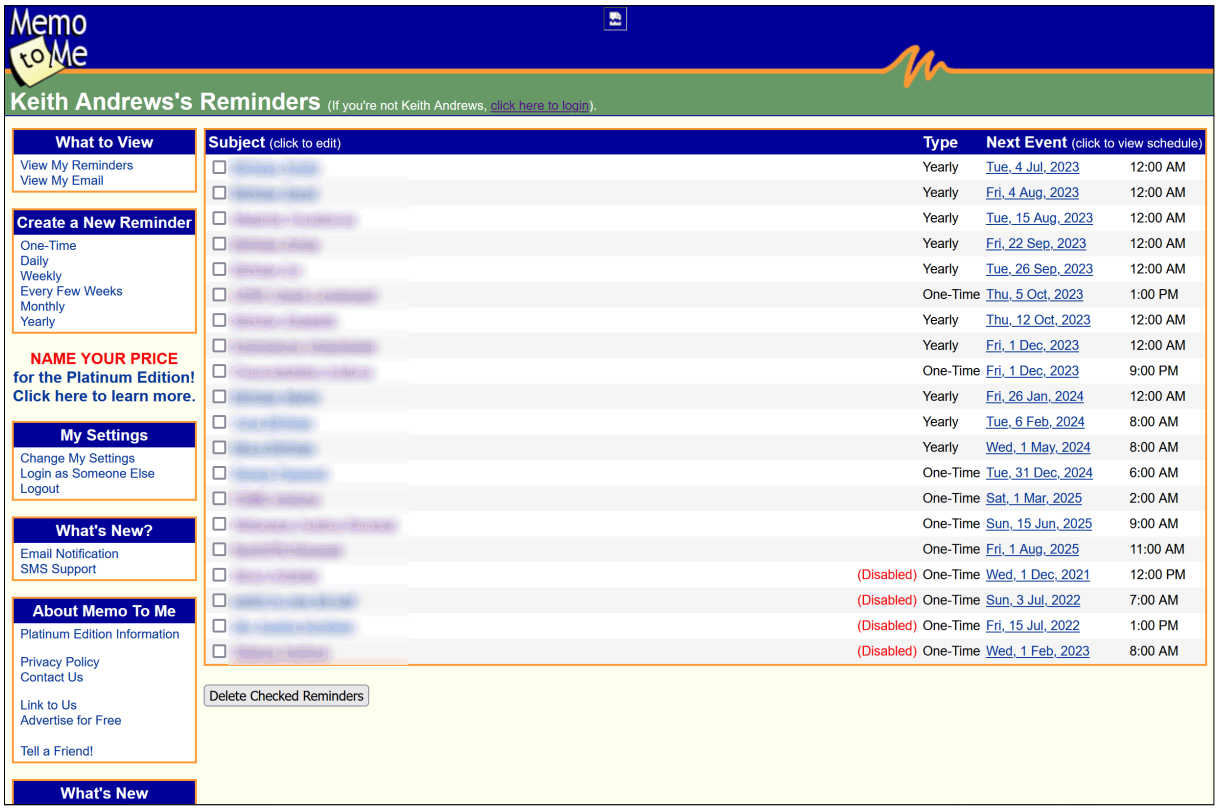


**Figure 2.1:** MemoToMe: User interface. [Screenshot taken by Keith Andrews. Used with kind permission.]

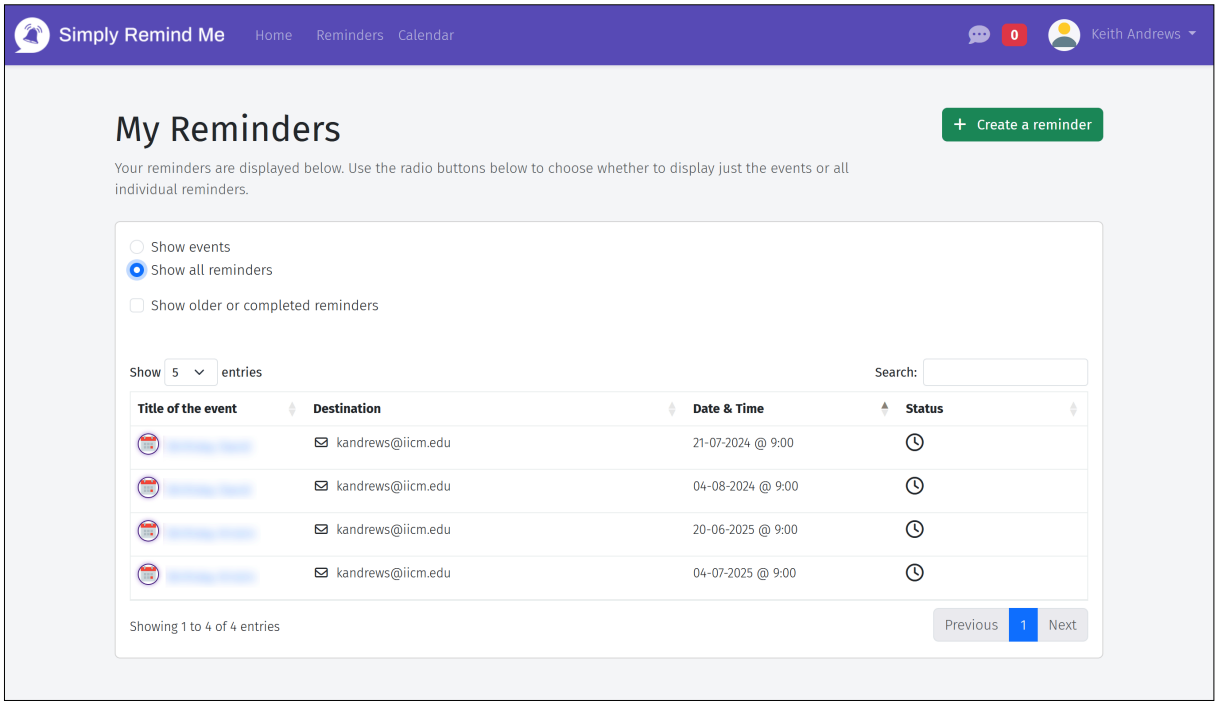


**Figure 2.2:** Simply Remind Me: User interface. [Screenshot taken by Keith Andrews. Used with kind permission.]

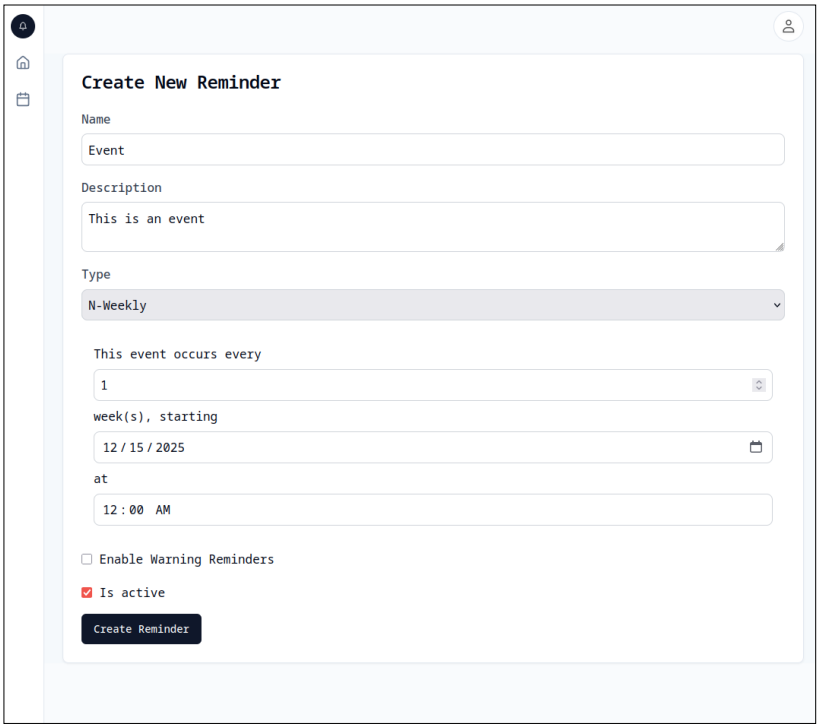


**Figure 2.3:** SelfMemo 2.0: Reminder creation. [Screenshot taken by the author(s) of this paper.]

- Turning on warning reminders.
- Managing user profiles, creating new users.
- Setting up the reminder intervals with minute-based precision.
- Sending email notification for reminders.

Among the downsides of SelfMemo 2.0 was the sheer number of tools and dependencies required to host and run it:

- GitHub [GitHub 2026] for code hosting and distribution.
- Vercel [Vercel 2021] for app deployment.
- Neon [Neon 2022] for hosting the PostgreSQL database.
- cron-job.org [cron-job.org 2026] for triggering the reminder procedure via a cron job.
- An SMTP email provider for sending reminder notifications.

This setup complexity was one of the reasons for revisiting the project with SelfMemo3.

# Chapter 3

# SelfMemo3

This chapter presents SelfMemo3 and describes the implemented changes and newly added features compared to previous versions.

## 3.1 Setup Changes

In addition to functional extensions, several setup-related changes were implemented to improve maintainability, compatibility and ease of deployment. First, the majority of deprecated dependencies were updated to their current versions. This improves security and ensures better long-term compatibility with the underlying frameworks and libraries. Some remaining warnings originate from external dependencies and could not be fully resolved without replacing those packages. To simplify local development and testing a setup script for local hosting was introduced. This script provides an automated way to configure a local PostgreSQL database, a local alternative to the cron-based scheduling mechanism, and service wrappers for both the application and the cron process. Furthermore, UI components were maintained and aligned to ensure visual consistency and accessibility across the application. This contributes to a more coherent user experience. Finally, a local JSON-based database alternative was added as a option for development and testing environments.

## 3.2 Implemented Features

As part of the SelfMemo3 project several new features were implemented to improve usability, flexibility and the overall user experience. These features extend the core email reminder functionality and address limitations observed in previous versions:

- *Remind Until*: This functionality allows users to define an end date for the recurring reminders. By specifying an end date, users can control how long a reminder remains active. Once the specified date is reached, the reminder is automatically stopped and no further notifications are sent.

- *Timezone Awareness*: SelfMemo3 introduces timezone-aware reminders to ensure that notifications are sent at the correct local time. Users can explicitly select the timezone that should be used for a reminder via a dropdown menu. This is particularly useful for users who travel frequently or work across different time zones.

- *Calendar Date Picker*: In order to provide a more intuitive approach and reduce input errors, SelfMemo3 introduces a calendar-based date picker for selecting reminder dates. Instead of manually entering dates, users can choose a specific date directly from an interactive calendar interface. This prevents formatting mistakes which are common with manual date input. The date picker is integrated into the reminder creation and editing workflow. It allows users to define when a reminder or a reminder series should start. By providing intuitive visual navigation through months and days, the date picker simplifies scheduling and enhances the overall user experience.

- *Customizable Email Templates*: SelfMemo3 also introduces customizable email templates for reminder notifications. This feature allows users to adjust the content and structure of reminder emails, enabling more personalized messages tailored to the users preferences. For example, reminder emails can be adapted for personal tasks or more formal ones. Customizable templates improve the usability of the reminder application by providing more control over how information is shared.
- *Dark and Light Modes*: Support for dark and light modes was added to enhance accessibility and user comfort. Users can choose their preferred theme depending on personal preference or lighting conditions.
- *Multi-User Reminders*: This functionality can be enabled by selecting the dedicated checkbox during reminder creation or editing. Once the checkbox is enabled, additional participants may be added using an input field. All listed participants receive the reminder email when it is triggered. Furthermore, if notifications for reminder changes are activated, the updates are also communicated to all participants.
- *Import and Export Functionality*: This functionality allows users to easily back up their reminders, transfer them, or migrate the data. The export feature generates a structured JSON file containing all relevant reminder information such as reminder types and associated settings. It also allows the user to restore previously exported reminders or import reminders created in another environment.
- *Visual Calendar View*: The calendar view provides the user with an intuitive overview of all scheduled reminders. Instead of relying solely on the list-based representation, reminders are displayed in a monthly calendar layout, making their temporal distribution immediately visible. Each reminder is shown on its corresponding date, along with relevant information such as the scheduled time, name, and current status (enabled or disabled). Furthermore, the calendar view supports navigation between months, allowing the user to inspect past and future reminders efficiently.

# Chapter 4

# Technology Stack

The SelfMemo3 application is built upon a set of modern web technologies that provide a robust, scalable, and maintainable foundation:

- *Next.js*: Next.js serves as the primary application framework, providing both the frontend and backend infrastructure through its full-stack capabilities [Vercel 2024]. The framework's hybrid rendering approach enables server-side rendering, static site generation, and client-side rendering within a single application architecture. Next.js abstracts away much of the configuration complexity associated with modern React applications, while offering built-in routing, API route handling, and optimized build processes. The framework's file-system based routing convention simplifies the organization of pages and API endpoints, making the codebase more intuitive and maintainable.

- *Mantine*: Mantine provides a comprehensive collection of React components that form the foundation of the application's user interface [Rtishchev 2024]. The library offers a rich set of accessible, customizable components, including forms, modals, date pickers, and navigation elements, all built with TypeScript support and consistent design patterns. Mantine's theming system enables cohesive visual styling across the application, while its form management utilities simplify validation and state handling for complex input scenarios such as reminder configuration forms. The library's built-in hooks and utilities streamline common UI development tasks, reducing the need for custom implementations and ensuring consistent user experience patterns throughout the application.

- *TypeScript*: The application is written in TypeScript to take advantage of its static type checking [Microsoft 2024]. This ensures type safety across the entire codebase, reducing runtime errors and improving developer productivity through enhanced IDE support and code completion. TypeScript's type system provides compile-time verification of data structures, function signatures, and component props, which is particularly valuable when working with complex data models such as the various reminder configuration types.

- *PostgreSQL Database*: PostgreSQL functions as the primary data persistence layer, offering robust relational database capabilities [PostgreSQL 2024]. The database stores all user accounts, authentication sessions, and reminder configurations, ensuring data integrity through transactional support and foreign key constraints. PostgreSQL's rich feature set, including support for JSON data types, enables efficient storage of complex reminder configurations while maintaining the benefits of a structured relational schema for core entities.

- *Prisma*: Prisma serves as the object-relational mapping layer, providing a type-safe database client, that bridges the gap between the TypeScript application code and the PostgreSQL database [Prisma 2024]. The Prisma schema defines the database structure using a declarative syntax, from which Prisma generates a fully typed client that prevents SQL injection vulnerabilities and provides compile-time query validation. Prisma's migration system manages database schema evolution

across development, staging, and production environments, ensuring consistent database state and simplifying deployment workflows.

- *NPM Package Manager*: NPM is used to manage the application's dependencies and provides scripting capabilities for build, development, and deployment tasks [npm 2024]. As the standard package manager for Node.js ecosystems, NPM handles the installation, versioning, and resolution of the numerous third-party libraries upon which the application depends.

# Chapter 5

# Application Setup

The SelfMemo3 project provides a comprehensive setup infrastructure designed to facilitate both production deployment on Linux systems and local development environments. The setup system is implemented in Python and consists of modular components handling different aspects of installation and configuration.

## 5.1 Setup Components

The setup process is driven by the configuration file `setup/config.json`, which contains various deployment parameters. It is shown in Listing 5.1. This configuration defines the application server endpoint, trigger service parameters, and PostgreSQL database credentials. The modular design allows for easy adaptation to different deployment environments. When deploying the application, it is recommended to extract the build from the config file due to security concerns.

The Python setup script `setup/setup.py` handles the installation process through a command-line interface. The script implements the following workflow:

1. *Privilege Verification*: Ensures execution with the root privileges required for system-level operations.
2. *Service User Creation*: Establishes a dedicated `selfmemo` system user with restricted privileges.
3. *Component Selection*: Presents an interactive command-line interface allowing selective installation of:
   - Next.js application only
   - Trigger service only
   - PostgreSQL database setup only
   - Complete stack installation
4. *Modular Installation*: Delegates to specialized setup modules for each component.
5. *systemd Integration*: Registers services with `systemd` for automatic startup.

## 5.2 PostgreSQL Database Setup

SelfMemo3 can be run with PostgreSQL as a backend database for storing the users and reminders. To make the setup process more convenient, a setup script has been created. The application employs a relational data model (when using PostgreSQL) or an equivalent JSON structure comprising two principal entities: the User entity and the Reminder entity.

The User entity represents system users with the following attributes:

```
{
  "url": "http://localhost",
  "port": 3000,
  "endpoint": "/api/reminders/trigger",
  "interval_seconds": 60,
  "database": {
    "name": "selfmemo",
    "username": "selfmemo_user",
    "password": "your_secure_password_here",
    "port": 5432
  }
}
```

**Listing 5.1:** Configuration file for the setup process.

- `id`: Unique identifier.
- `username`: Optional unique username.
- `email`: Optional unique email address.
- `password`: Hashed password credential.
- `firstName`, `lastName`: Optional name components.
- `role`: Authorization role (admin/user).
- `defaultTimezone`: IANA[1] timezone identifier (default: "Etc/GMT").
- `dateFormat`: Preferred date formatting option.

The Reminder entity encapsulates reminder configurations:

- `id`: Unique identifier.
- `userId`: Foreign key to User entity.
- `name`: Reminder name/title.
- `description`: Detailed reminder description.
- `type`: Reminder recurrence pattern (one-time, daily, weekly, monthly, yearly, etc.).
- `config`: JSON-encoded configuration specific to reminder type.
- `isDisabled`: Boolean flag for temporary deactivation.
- `lastSent`: Unix timestamp of most recent notification.
- `hasWarnings`: Boolean indicating warning reminder. capability.
- `warningNumber`: Count of warning reminders to send.
- `warningInterval`: Interval type for warnings.
- `warningIntervalNumber`: Numeric value for warning interval.

[1] IANA 2024

- `timezone`: IANA timezone for reminder evaluation.
- `emailTemplate`: Template identifier for email composition.
- `additionalUserIds`: JSON array of additional recipient user IDs.

The PostgreSQL setup module `setup/postgres/setup.py` handles database installation and configuration through automated detection and installation. If PostgreSQL is not detected, the system offers automated installation via the system package manager (currently supporting apt-based systems). This approach ensures compatibility with Debian-based distributions while maintaining extensibility for other package management systems.

Following successful installation, the setup module executes the following database initialization procedures:

1. *User Creation*: Establishes a dedicated database user with appropriate privileges:
   ```
   CREATE USER selfmemo_user WITH PASSWORD 'password' CREATEDB;
   ```
2. *Database Creation*: Creates the application database with proper ownership:
   ```
   CREATE DATABASE selfmemo OWNER selfmemo_user;
   ```
3. *Privilege Grant*: Assigns necessary permissions to the database user:
   ```
   GRANT ALL PRIVILEGES ON DATABASE selfmemo TO selfmemo_user;
   ```

The module performs existence checks before creation operations to ensure idempotent behavior, allowing the setup script to be re-executed without errors.

After successful database setup, the script generates the appropriate connection string:

```
DATABASE_URL="postgresql://user:password@localhost:5432/selfmemo"
```

This connection string must be added to the application's `.env` file to enable database connectivity.

## 5.3 Application Setup

The application setup module `setup/app/setup.py` handles the deployment of the Next.js web application. It also deploys a `systemd` service, providing robust process management, automatic restart capabilities, and logging integration.

The trigger service implements a critical component of the SelfMemo3 architecture, the periodic reminder evaluation. In production deployments, this service replaces external cron services by implementing an internal scheduler that periodically invokes the reminder processing endpoint.

The installation process involves:

1. *Service Directory Creation*: Establishes the `/opt/trigger_service` directory.
2. *Script Deployment*: Copies Python trigger script to service directory.
3. *Configuration Deployment*: Installs configuration file from setup directory.
4. *systemd Unit Installation*: Deploys service unit file to `/etc/systemd/system`.
5. *Dependency Installation*: Installs Python requests library via `apt`.
6. *Permission Configuration*: Sets ownership to `selfmemo` user.
7. *Log File Creation*: Establishes log file with appropriate permissions.

The service reads configuration from `config.json`, constructs the trigger endpoint URL, and executes HTTP GET requests at the specified interval (default: 60 seconds).

## 5.4 Local Development Setup

For local development scenarios, SelfMemo3 provides the script `setup/run_local.py`, a development orchestration script that eliminates the need for system-level service installation.

## 5.5 Configuration and Environment Variables

The application relies on the following environment variables for runtime configuration, specified in a `.env` file:

- `DATABASE_URL`: PostgreSQL connection string (optional for file-based mode).
- `AUTH_SECRET`: Secret key for session token signing.
- `AUTH_TRUST_HOST`: Trust proxy headers (true/false).
- `SMTP_HOST`, `SMTP_PORT`, `SMTP_USER`, `SMTP_PASS`: Email server configuration.
- `SMTP_MAIL`: Sender email address.
- `ADMIN_EMAIL`, `ADMIN_PASSWORD`: Initial admin account credentials.

## 5.6 Schema Migration

When using PostgreSQL, Prisma migrations manage database schema evolution:

```
npx prisma migrate dev
```

For file-based storage, only Prisma client generation is required:

```
npx prisma generate
```

## 5.7 Repository Pattern Implementation

SelfMemo3 implements the Repository pattern to abstract data access logic and provide a unified interface for data operations regardless of the underlying storage mechanism. This architectural decision enables the application to operate with either PostgreSQL database or JSON file storage without modifying the business logic. The User Repository interface `IUserRepository` defines user data operations, and is shown in Listing 5.2. The Reminder Repository interface `IReminderRepository` specifies reminder data operations, and is shown in Listing 5.3.

## 5.8 Repository Factory

The `RepositoryFactory` module implements runtime repository selection based on environment configuration. This factory pattern enables transparent storage backend selection. The presence or absence of the `DATABASE_URL` environment variable determines which implementation is instantiated. Service layer code remains agnostic to the underlying storage mechanism, depending only on the repository interfaces.

## 5.9 Email Templates

Reminders can have reoccurring patterns. Email templates are used to predefine standard text sections, without the need to retype them every time a new reminder is created. The template system is defined in the file `public/email-template.json`. Template configuration uses a two-level hierarchy, as shown in Listing 5.4. The system defines the following template types:

```
interface IUserRepository {
  create(user: CreateUserDto): Promise<User>;
  update(user: UpdateUserDto): Promise<User>;
  delete(id: string): Promise<User>;
  getById(id: string): Promise<User | null>;
  getAll(): Promise<User[]>;
  getWhere(where: any): Promise<User[]>;
  getByEmail(email: string): Promise<User | null>;
  getByUsername(username: string): Promise<User | null>;
  updatePassword(id: string, newPassword: string): Promise<User>;
  searchByUsername(query: string): Promise<User[]>;
}
```

**Listing 5.2:** The User Repository interface.

```
interface IReminderRepository {
  getAll(): Promise<Reminder[]>;
  getById(reminderId: string): Promise<Reminder | null>;
  create(reminder: CreateReminderDto): Promise<Reminder>;
  update(reminder: UpdateReminderDto): Promise<Reminder>;
  delete(id: string): Promise<Reminder>;
  updateLastSent(reminderId: string, timestamp: number): Promise<Reminder>;
  getAllByUserId(userId: string): Promise<Reminder[]>;
}
```

**Listing 5.3:** The Reminder Repository interface.

- `default`: Generic reminder template.
- `birthday`: Birthday-specific reminders.
- `meeting`: Meeting reminder template.
- `task`: Task deadline reminders.

Each template type contains three notification subtypes:

- `reminder`: Primary notification when reminder is due.
- `warning`: Advance warning notifications.
- `edit`: Notification sent when reminder is modified.

Adding custom templates is done by adding a new template type following the described structure. The three subtypes `reminder`, `warning`, and `edit` must be present, otherwise the default type is used.

Templates support placeholder variables using double-brace syntax:

- `{{firstName}}`: User's first name
- `{{lastName}}`: User's last name
- `{{reminderName}}`: Name of the reminder
- `{{description}}`: Reminder description
- `{{nextdate}}`: Next scheduled notification date

```
{
  "template_type": {
    "notification_type": {
      "subject": "Email subject with {{placeholders}}",
      "body": "Email body with {{placeholders}}"
    }
  }
}
```

**Listing 5.4:** The structure of email templates.

# Chapter 6

# Future Work

Although great progress has been made in further developing the functionality of the SelfMemo3 application, some features could not be implemented due to time constraints.

## 6.1 Static Build

The Next.js framework, by default, operates as a server-side application that requires a Node.js runtime environment to execute. When deploying with `next start`, the application runs as a dynamic server capable of handling server-side rendering, API routes, and other runtime features that depend on the Node.js execution environment. This approach necessitates maintaining a Node.js server infrastructure, which introduces complexity in terms of deployment, scaling, and resource management.

In contrast, a static build transforms the Next.js application into a collection of pre-rendered HTML, CSS, and JavaScript files that can be served directly by any conventional HTTP server such as Nginx, Apache, or even Python's built-in HTTP server. This static export eliminates the dependency on Node.js in the production environment, significantly simplifying the deployment architecture. The resulting static files can be hosted on commodity web servers, content delivery networks, or even basic file-serving infrastructure without requiring specialized Node.js hosting capabilities.

## 6.2 Dynamic Template Input Fields

The current email template system in SelfMemo employs a static field approach, where only predefined placeholders enclosed in double braces (such as `{{reminderName}}`, `{{firstName}}`, and `{{description}}`) are recognized and substituted during email generation. This rigid structure limits extensibility, as administrators must manually update both the template configuration file and the reminder form component whenever new data fields are needed. Such tight coupling between templates and form logic introduces maintenance overhead and reduces the system's ability to adapt to evolving user requirements.

A more sophisticated approach would involve implementing a dynamic field detection mechanism that parses email templates at runtime to identify placeholder variables present in the template body and subject lines. The system would then automatically generate corresponding input fields in the reminder creation form based on the detected variables. For instance, if a custom template contains placeholders like `{{location}}` or `{{deadline}}`, the form would dynamically render the appropriate input components without requiring code modifications. This would be achieved by scanning the selected template's JSON structure, extracting unique variable names through regular expression matching, and programmatically constructing form elements with appropriate labels and validation rules.

## 6.3 International Date Format Types

It would be great to expand the number of date formats beyond the current "full" default, to cover international date formats like:

- ISO 8601: `YYYY-MM-DD` (e.g., 2026-02-01).
- US format: `MM/DD/YYYY` (e.g., 02/01/2026).
- European format: `DD/MM/YYYY` (e.g., 01/02/2026).
- Long format: `February 1, 2026`.
- Short format: `Feb 1, 2026`.
- Custom format strings using libraries like `date-fns` or `luxon`.

and to support Schema modifications like:

- Update `dateFormat` field to enum or validated string.
- Adding `timeFormat` field (12h/24h).
- Integrate with existing `defaultTimezone` for complete locale support.

## 6.4 Multi-System Support

The current setup scripts are designed exclusively for Debian-based systems using the APT package manager. This limitation restricts deployment to distributions such as Ubuntu, Debian, and their derivatives. The setup infrastructure exhibits several distribution-specific dependencies. Package management operations assume APT availability and use `apt-get` for package installation. Service configuration assumes `systemdt` without any fallback options. File system paths and service locations follow Debian conventions, and database installation procedures are specific to Debian package repositories. To support broader deployment scenarios, the setup infrastructure should be extended to accommodate multiple system types.

# Chapter 7

# Concluding Remarks

SelfMemo3 builds upon previous work to depart from cloud-dependent architectures, offering users complete control over their reminder infrastructure without reliance on external service providers. It also provides substantial enhancements to both deployment flexibility and reminder functionality. Although great progress has been made in further developing the functionality of the SelfMemo3 application, due to time constraints some features have not yet been implemented.

# Bibliography


cron-job.org [2026]. *Free cronjobs - from minutely to once a year.* 2026. https://cron-job.org/ (cited on page 5).

Florian, Celine, Stephan Robinig, Piotr Siewiera, and Nina Tschikof [2026]. *SelfMemo 3.0*. 02 Feb 2026. https://github.com/StofflR/SelfMemo3 (cited on page 1).

GitHub [2026]. *GitHub*. 2026. https://github.com/ (cited on page 5).

Heathershaw, Andy [2021]. *Simply Remind Me*. Waggybytes, 2021. https://simplyremind.me/ (cited on page 3).

IANA [2024]. *IANA Time Zone Database: Collaborative Compilation of Time Zone Information*. Internet Assigned Numbers Authority, 2024. https://www.iana.org/time-zones (cited on page 12).

Lorber, Niklas, Maria Seiser, and Kilian Weisl [2025]. *SelfMemo 2.0: Open-Source, Self-Hosted Email Reminder Service*. Technical report. Graz University of Technology, 04 Feb 2025. https://courses.isds.tugraz.at/iaweb/projects/ws2024/iaweb-ws2024-g4-project-selfmemo2.pdf (cited on page 3).

Microsoft [2024]. *TypeScript: JavaScript with Syntax for Types*. 2024. https://typescriptlang.org/ (cited on page 9).

Neon [2022]. *Neon Serverless Postgres - Ship faster*. Neon, LLC, 2022. https://neon.com/ (cited on page 5).

npm [2024]. *npm: Node Package Manager*. 2024. https://npmjs.com/ (cited on page 10).

PostgreSQL [2024]. *PostgreSQL: The World's Most Advanced Open Source Relational Database*. PostgreSQL Global Development Group, 2024. https://postgresql.org/ (cited on page 9).

Prisma [2024]. *Prisma: Next-generation Node.js and TypeScript ORM*. 2024. https://prisma.io/ (cited on page 9).

Rtishchev, Vitaly [2024]. *Mantine: A Fully Featured React Components Library*. 2024. https://mantine.dev/ (cited on page 9).

Vercel [2021]. *Vercel: Build and deploy the best web experiences with the AI Cloud*. 2021. https://vercel.com/ (cited on page 5).

Vercel [2024]. *Next.js: The React Framework for the Web*. 2024. https://nextjs.org/ (cited on pages 3, 9).