Package Update Notifier

Final Report

Course: CS 4850-03

Semester: Spring 2023

Professor Perry

Team: INDY2

Prepared by:

Nicholas Ott, Jason Paek, Erik Smith, Haris Yosufi

Table of Contents

1.0 Introduction

1.1 Team Members

Nicholas Ott **Jason Paek** **Erik Smith** **Haris Yosufi**

1.2 Team Roles

Roles	**Name**	**Major responsibilities**	**Contact (Email, Phone, LinkedIn)**
Project owner	Nicholas Ott	Front+Back end developer project scope management critique project deliverables project detail provider	706-248-8301 nott@students.kennesaw.edu https://www.linkedin.com/in/nicholas-ott-288680233/
Team leader	Jason Paek	Front end developer documenter team management meeting planning	404-277-5277 jpaek4@students.kennesaw.edu https://www.linkedin.com/in/jasonpaek/
Team members	Erik Smith	Front end developer documenter	770-377-9456 esmit282@students.kennesaw.edu https://www.linkedin.com/in/erik-smith-2a9a01208/
	Haris Yosufi	Back end developer documenter	678-704-3103

			hyosufi@students.kennesaw.edu https://www.linkedin.com/in/haris-yosufi-900754270/
Advisor / Instructor	Sharon Perry	Facilitate project progress; advise on project planning and management.	Sperry46 in D2L !!

1.3 Abstract

Our application sends emails to users notifying them of outdated packages in their programming projects. The frontend is a website developed using Bootstrap and SvelteKit, while the backend is written in C# and interfaces with a SupaBase database. The application integrates with the user's GitHub account and compares package versions between the user's GitHub repository and the package's source repository. Additionally, it can be configured to exclude specific packages and send less frequent email reminders. Implementing this helpful tool will allow developers to fix any runtime error caused by outdated packages and manage their code's packages with ease.

1.4 Project Scope

These are the main goals of our application:

1. Should allow for end users to integrate their chosen repo website into our application, and then compare versions of those packages to versions of their own respective repos
2. The application should be configured to allow exclusion of certain versions and check how far packages have to go out of date before needing a reminder
3. It should save emails of users and send email notifications reminding them to update outdated packages and adjust frequency of these notifiers.

1.5 Definitions and Acronyms

Packages - A Package can be defined as a grouping of related types (classes, interfaces, enumerations and annotations) providing access protection and namespace management.

Repository (REPO) - A repository is computer storage for maintaining data or software packages. This location contains files, databases, or information organized for quick access over a network or directly. A repo allows consolidating data with a version control system to store metadata for every file and log changes.

FrontEnd - This is all software and hardware that is part of the user interface, including data input, buttons, programs, websites, and etc…

BackEnd - This refers to the parts of a program that goes behind the user interface and includes operations non accessible by the user.

GIT - It is a version control system/tool used for file tracking and tracking changes in the source code.

GitHub - A distributed version control platform open to user collaboration on.

API - aka Application Programming Interface, refers to a set of functions and procedures that allow for the creation of applications. Additionally, they access data and features of other applications, services, or operating systems.

Authentication - A verification of the identity of a user, process, or device to allow access to a certain resource in a system.

Third-party - This refers to any source that is independent of the supplier and customer of a major computer product.

1.6 Assumptions

These are the following assumptions of our project:

1. The Client has a GitHub account and is familiar with its application
2. The program should have access to the user's repos and have the ability to read said repos
3. The program should have the ability to pull update information from package repos

2.0 Design Constraints

2.1 Environment

The Package Update Notification system will be a standalone application hosted on a dedicated server. It will interface with a Supabase database to store user details. The application will use OAuth to authenticate the connection to GitHub. A modern web browser is required to access the Package Update Notification system.

2.2 User Characteristics

Developers (Priority):

Developers are responsible for being vigilant about checking email notifications for updates. They should be familiar with the system's method of distributing notifications and what to do upon receiving said notifications. They will generally have a background in computer science, with many having at least an undergraduate degree in the field. Their robust technical knowledge allows them to quickly learn new systems and utilize them accordingly.

Project Managers:

Project Managers are responsible for ensuring steady progress on various projects. They should be aware of the system's general functionality as it will be an integral part of whatever project they are managing. They will have a background in project management with many having some kind of college degree. Their technical knowledge is likely less broad than that of the developers.

3.0 Requirements

3.1 Website Requirements

3.1.1 Authentication with Git repository websites such as GitHub

3.1.1.1 Third party websites account info will have already been validated by the third party site

3.1.1.2 Should have access to read user's repositories

3.1.2 Data Access

3.1.2.1 Should be able to save user and repo settings to the connected Supabase database

3.1.3 Accessibility

3.1.3.1 Should be accessible from any major browser (Chrome, Firefox, Edge, etc.)

3.2 Back End Requirements

3.2.1 Access to Supabase database

3.2.1.1 Should be able to pull data from the Supabase database

3.2.2 Repo scanning

3.2.2.1 Should be able to parse package information for any accessible repos

3.2.2.2 Should be able to handle package info from multiple project types

3.2.3 Notification

3.2.3.1 Should be able to send users information about package out of date

3.2.3.2 Notification should be an email

3.2.3.3 The notification should be according to their specifications

3.2.4 Constraints

3.2.4.1 In order to sync in with the website they have to have Oauth integration which will allow read access to the users repositories. While this is the

case with websites such as GitHub and Bitbucket already, this may not be the case for all websites, which may mean those websites would go unsupported.

3.3 Security

Security of User accounts, user and repo preferences, repos the application has access to are accessed through GitHub and not stored by our application. This access is opt-in by the user, and this access can be opted out of at any time by the user through the website they signed into the application with. Some of the many security features include: Security Hardening, Encrypted Secrets, and Automatic token authentication which can all be further explained in their official website: https://docs.github.com/en/actions/security-guides/security-hardening-for-github-actions#reusing-third-party-workflows.

3.4 Capacity

At a bare minimum:

- The system should be able to handle 50 concurrent users with further scalability
- Other system requirements and data capacities are handled and managed by GitHub

3.5 Usability

User satisfaction surveys will be offered to testers during development and to end users post development. Quantifiable data should be gathered in order to evaluate system efficiency and quality of life to the program created. More data should be gathered upon the closer completion of the software, surveys can be created using google survey.

4.0 External Interface Requirements

4.1 User Interface Requirements

Users should be able to access the website on any major web-browser (Chrome, Firefox, Edge, etc…)

4.2 Hardware Interface Requirements

An external server is required to host both the frontend and backend for the application. Software developed for any operating system capable of opening a web browser. The server should run on Linux.

4.3 Software Interface Requirements

Building website with svelte kit, bootstrap for user interface, and supabase for authenticating users with their account in the repo website and to store and save user settings. C# libraries are being utilized for supabase which can be cross referenced to this website: https://supabase.com/docs/reference/csharp/v1/select . All SupaBase-Csharp objects and methods will be inherited into our project.

4.4 Communication Interface Requirements

An active internet connection is required for users to access the website application. The server used to host the application must be online to allow users to connect and to guarantee successful communication between the frontend and backend of the application.

5.0 Development Process

5.1 Tool Utilization

Front End incorporates a website that was built using SvelteKit and Bootstrap

SvelteKit - Framework providing project structure and JS-powered HTML

Bootstrap - CSS framework for JavaScript based web design

Back End was built using .NET 7 in C#

.Net 7 - platform for robust back-end services with extensive libraries

Supabase was utilized to help authenticate users in GitHub

5.2 Functionality Development

5.2.1 Log-in

For the log-in functionality, a button was created that users must click on before being able to access the rest of the application. This button prompts the user to sign in with their GitHub account. Signing in here gives the application access to the user's GitHub account. The sign in also utilizes OAuth to give Supabase access to the user's GitHub account. Supabase uses this information for storing user settings and allows the user to remain signed in upon reopening the website.

5.2.2 Repo Display/Update

The repo display functionality involves all three parts of our project: the front end, back end, and Supabase. Initially, the front-end website requests a response from Supabase through the Log-in feature as described in 5.2.1, which then awaits the repo information that displays on the website. Afterwards, we needed the user to be able to update their repo notification settings in order to choose their desired repos. To accomplish this task, we designed the website to allow the user to interact with their repos by opting into the repos they want through a checkbox feature in bootstrap. When the user opts in or out of a repo, the changed settings get sent to Supabase and interacts with the backend where the user settings are stored and manipulated.

5.2.3 Email Notification

The email notification feature primarily involves the backend where there's an 'EmailBuilder' class containing a constructor that takes 'ResponseBuilder' object as a parameter. The response builder class is used to build the list of out-of-date packages for

a given repository. The ‘makeEmail’ method creates an HTML email template with a title, header, and unordered list. The ‘foreach’ loop iterates through the packages in the ResponseBuilder object and creates an HTML list item for each package, displaying its name, repository version, and latest version. It also creates an HTML list item for the repository name and includes the package list within it. The ‘makeEmail’ method returns the email template with the repository and package list information inserted into it using ‘string.Format’. This class is designed to generate an HTML email template with information about any out-of-date packages in the users GitHub repository and works in conjunction with the ‘ResponseBuilder’ class to create the necessary information for the email.

5.3 Frontend Challenges

5.3.1 HTML Design

The main extent of our challenges resonates from the communication between SupaBase and our front-end HTML along with trying to understand the supplemental usage of both Bootstrap and SvelteKit. In reference to design, since most of us were new to Bootstrap and SvelteKit, the introductory code was easy to learn but hard to master. Simple techniques like creating user interface buttons or tables became research tasks into bootstrap design, and SvelteKit made it harder to comprehend the framework. The hardest design feature we had to implement came from displaying all of the repo data with readability taken into consideration. In order to make it organized and readable, we had to change the print display to incorporate tables, the colors of the background, and the sizing of the characters. Limiting the amount of data and details of the repo also came into discussion as throwing all the repos’ package, version, and update information would prove to be overwhelming to the user. We came to the decision that the user’s repo name along with a simple opt-in/opt-out would be enough for the display page. In the email notification settings page, we configured it to allow the user to change the day of the week and if they would like monthly or weekly notifications. By making the website simple to navigate and easy to read, we made it easy to learn and use. A majority of the repo information is included in the actual email that is sent from the backend which is configured according to the preferences/notification settings set by the user.

5.3.2 Functionality

By far, the hardest overall functionality came from parsing SupaBase to obtain the user’s GitHub information. When making requests to SupaBase or GitHub, it is necessary to understand the format of the data being returned. When requesting the user’s repos, a list is returned that contains arrays of data pertaining to each individual repo. Finding methods for parsing this data to acquire the desired fields proved challenging, especially when it came time to display this information to the user. One roadblock we encountered was with regards to handling individual package exclusion. Originally, we intended to

allow the user to choose specific packages in each repo to be excluded from notifications. While displaying the individual packages and their associated versioning proved nontrivial, excluding specific packages as part of the user settings proved too difficult to implement given the time constraints of the project. Since this was a key feature originally planned to be completed, it was a difficult decision to cut it from the application. We decided to add package exclusion to the future implementations list that contains ideas on how to expand and extend the application going forward.

5.4 Backend Challenges

The major challenges from building the backend were based on the integration and connections between four critical components, Supabase, Github, SMTP, and terminal commands. Supabase was the simplest to set up, though it required the most amount of code, as several data models were needed to interface with the database. Terminal commands took a decent amount of work to figure out in C#, but after researching critical components and syntax, it was fairly straightforward. Terminal commands were utilized to run the update scripts of each of the repos' relevant files. SMTP was handled through Google Workspace, which required a lot of setup for server connection and authentication. SMTP was the protocol in which emails were sent in our program. Finally, GitHub has its own OAuth features which needed to be accounted for, which caused a major slowdown regarding the production of the backend.

6.0 Conclusion

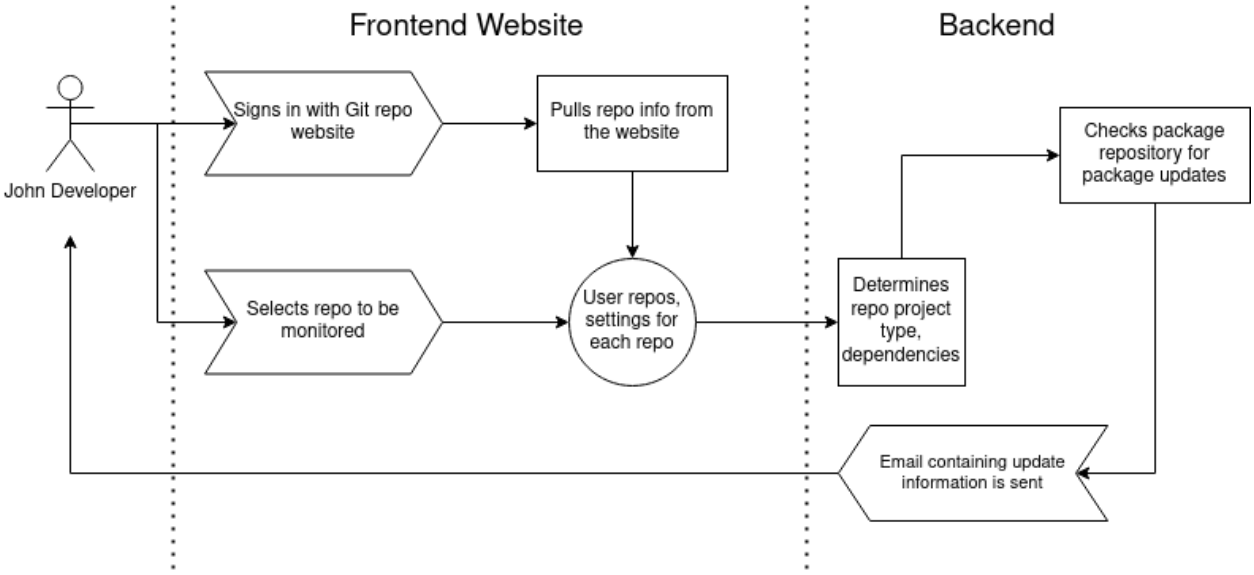
The Package Update Notifier has provided us with invaluable experience with Supabase by the challenges that it has presented in the implementation. In appendix B, a harmonious relationship between the front-end and the back-end is illustrated through its communication of calls and responses to Supabase. The Front End would incorporate the HTML user interface, send repo log information, and would send updates to Supabase for it to then send the changed user-desired repo notification settings to the backend. Supabase handles the communication between these two ends in addition to handling authentication through its OAuth functionality. The backend would then handle the user settings, obtain and send back all repo information, and would also have email notification handling. By understanding the flow of our functionality, we have created a helpful tool to aid developers in managing packages and prevent cascading errors resulting from outdated package dependencies.

APPENDIX A: Project Schedule

The following Gantt chart showcases the current schedule for the project along with the required work listed under “tasks”.

	Project Name:	INDY-2—Package Update Notification																			
	Report Date:	2/3/2023																			
						Milestone #1						*Milestone #2 & C Day*			*Milestone #3*	*Milestone #4*					
	Deliverable	**Tasks**	**Complete%**	**Current Status Memo**	**Assigned To**	**02/10**	**02/17**	**02/24**	**03/03**	**03/10**	**03/17**	**03/24**	**03/31**	**04/07**	**04/14**	**04/21**	**04/28**				
	Requirements	Meet with stakeholder(s) SH	60%		ALL	20	12														
		Define requirements	20%		ALL																
		Review requirements with SH	30%		ALL		10	15													
		Get sign off on requirements	0%		Nic			5	10												
	Project design	Define tech required *	0%		ALL				10	4											
		Database design & setup	0%		Nic, Haris				5	10	10			10							
		Front end design	0%		Jason, Erik						10	10		10							
		back end design	0%		Nic, Haris			6	6	10	5	5	5								
		Develop working prototype	0%		ALL						10	10									
		Test prototype	0%		ALL							5	10	5	6	5					
	Development	Review prototype design	0%		ALL										5	5	10				
		Rework requirements	0%		Jason, Erik										5	10	20				
		Document updated design	0%		Jason, Erik										10						
		Test product	0%		ALL												8				
	Final report	Presentation preparation	0%		Erik, Haris											5					
		Demo Preparation	0%		Jason											5	10				
		Final report submission to D2L and project owner	0%		ALL											5	5				
				Total work hours	**332**	**20**	**22**	**26**	**31**	**24**	**35**	**30**	**15**	**15**	**26**	**35**	**53**				
		* formally define how you will develop this project including source code management																			

APPENDIX B: Use Case Diagram



APPENDIX C: Project Plan Document

INDY-2—Package Update Notifier

CS 4850 - Section 03 – Spring 2023

Date: 01/24/2023

Overview

Thousands of lines of code and dozens of files are managed by you and your team as it is near complete; however, as the deadline sneaks up--BAM!--you are flooded with runtime errors from package update requirements within repositories. To remove the possibility of ever going through the painstaking effort of updating and checking every individual package to ensure that they are up to date, we will develop a website that should allow users to integrate with their desired repo website and compare versions of those packages to versions of their own respective repos. Additionally, it can send recipients emails reminding them to update their outdated packages, configure to allow exclusion of packages, check how far packages have to go out of date before needing reminder, and adjust frequency of these checks. Implementing this helpful tool will allow developers to fix any runtime error caused by outdated packages, and manage through their code and packages with ease.

Project Team

Roles	Name	Major responsibilities	Contact (Email and/or Phone)
Project owner	Nicholas Ott	Front+Back end developer project scope management critique project deliverables project detail provider	706-248-8301 nicsonfire4@gmail.com
Team leader	Jason Paek	Front end developer documenter team management meeting planning	404-277-5277 jasonpaek2000@gmail.com
Team members	Erik Smith	Front end developer documenter	770-377-9456 eriksmith98@gmail.com
	Haris Yosufi	Back end developer documenter	678-704-3103 Harisyosufi199@gmail.com
Advisor / Instructor	Sharon Perry	Facilitate project progress; advise on project planning and management.	Sperry46 in D2L !!

Project website

Final Project Application
Packupnt.com

Final Deliverables

1. A backend application built using .NET 7 and C# that will be able to handle the major processes of the application, including checking package validity, and sending notification emails out to end-users.
2. A website written using SvelteKit that should be accessible by the end-user, which would handle registering for the application, as well as user variables and settings.
3. Final report summarizing our project efforts and details.

Milestone Events

#1 - By 3/17/23
- First Prototype
- Supabase database set up and functional
- Presentation of Prototype

#2 - By 4/7/23
- Completion of Website
- User authentication through Supabase set up and functional
- C-day Application

#3 - By 4/14/23
- Draft of final report

#4 - By 4/27/23
- Complete Website and functional backend application
- Complete Final Report
- Presentation with Demonstration

Future Milestone Meeting Schedule Date/Time

- 3/13/23 Mon 5PM-7PM
- 4/3/23 Mon 5PM-7PM
- 4/10/23 Mon 5PM-7PM
- 4/24/23 Mon 5PM-7PM

Collaboration and Communication Plan

The main method of communication for this project will be Discord. Google Docs will be utilized for collaboration on project documentation. All source code for both front end and back end will be on a shared GitHub repo. The team will meet weekly

at predetermined times, based upon available schedule, in discord to discuss milestone objectives. They will also meet in person, as able, every one to two weeks.

Risk Assessment

Package update notifier being incapable of sending out adequate/timely emails to users or accessing the update information for the repos.

Version Control Plan

Utilize a GitHub repo for the project to submit all code and documentation. It has an automatic version control.