**Technical Design Document (TDD)**

**Version 1.1**

DevSecOps Team

Spring 2022

University of Maryland Global Campus

SWEN 670

Prepared by Robert Wren and Andrew Nicolette

For Approval by Dr. Mir Assadullah

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author(s)** |
| 2/5/2022 | 1.0 | Initial document | Robert Wren  Andrew Nicolette |
| 03/26/2022 | 1.1 | Updates based on feedback from initial submission | Robert Wren  Andrew Nicolette |

**Table of Contents**

1. Introduction 4

1.1 Purpose 4

1.2 Scope 4

1.3 Definitions, Acronyms, and Abbreviations 5

1.4 References 6

2. Software Architecture Design 6

2.1 Automation 6

2.1.1 Code Repository 6

2.1.2 GitHub Actions 7

2.1.3 Continuous Integration 7

2.1.4 Continuous Delivery 8

2.1.5 GitHub Secrets 9

2.2 Package Management 10

2.2.1 Package Registry 10

2.2.2 Package Manager 10

2.2.3 Container Registry 10

2.2.4 Identity and Permissions 10

2.3 Software Supply Chain 11

2.3.1 Dependencies 11

2.3.2 Vulnerabilities 11

3. Deployment Platforms 11

3.1 Deploy to Google Play 12

3.1.1 Build and Release the Android App 12

3.2 Deploy to App Store 14

3.2.1 Build and Release the iOS App 14

# 1. Introduction

## 1.1 Purpose

The Spring ’22 Software Engineering Capstone project requires the enhancement and development of a mobile application, MemorEZ, to assist people with short-term memory loss. This semester the goal will be to continue the work of prior teams and publish the app in the Apple App Store and Google Play. This Technical Design Document (TDD) describes the design and architecture of the MemorEZ DevSecOps Continuous Integration and Continuous Delivery (CI/CD) pipeline.

A Continuous Integration (CI) pipeline with a centralized code repository will be developed for the two application development teams with test automation and static code analysis tools to verify functionalities and ensure industry standards are followed, including security. A Continuous Delivery (CD) pipeline will be developed to build and deploy the source code to the desired target environment.

The DevSecOps team will use this TDD as a point of reference for specifications during the implementation phases. This document shall represent the overall blueprint to systematically realize the design of the CI/CD pipeline.

## 1.2 Scope

The DevSecOps CI/CD pipeline will serve to manage the integration and deployment of the MemorEZ application. This document will be the basis of the design and implementation of this infrastructure.

Given the current pace and cadence of modern software development efforts, it is no longer effective or conducive to market expectations and timelines to rely on manual development operations processes. The CI/CD pipeline will therefore execute automated operations activities related to the building, testing, analyzing, and deployment of applications in a highly integrated and monitored solution.

The pipeline will enable the MemorEZ development teams to deliver higher quality software more efficiently and frequently. The system is to be a collection of distributed software services hosted on the web configured specifically for the technical needs of the MemorEZ development teams. Implementation shall encompass the configuration and setup of each pipeline component to deliver the core functionalities of integrating and deploying the application components.

## 1.3 Definitions, Acronyms, and Abbreviations

“By developing security as code, we will strive to create awesome products and services,

provide insights directly to developers, and generally favor iteration over trying to always

come up with the best answer before a deployment. We will operate like developers to

make security and compliance available to be consumed as services. We will unlock and

unblock new paths to help others see their ideas become a reality.”

Second principal of the DevSecOps Manifesto, <https://www.devsecops.org>

The below list defines DevSecOps project terms and acronyms.

* TDD - Technical Design Document, describing the design of the entire CI/CD pipeline and its individual components.
* CI/CD - Continuous integration and continuous delivery, or CI/CD pipeline, a set of operating principles, practices, and tools used to deliver code often and reliably.
* Continuous Delivery - Creating a repeatable and reliable process for delivering software in order to deliver high-value software to our users quickly.
* DSO - DevSecOps, Development, Security & Operations, or the set of practices that combines software development, IT security, and IT operations in the integration and deployment of software applications.
* Git – The project’s chosen version control system for source code.
* Repo - Source code repository to store code and documentation for software hosted on the web, such as GitHub.
* MemorEZ - The application under development, a mobile application to assist people with short-term memory loss.
* Google Play - A digital distribution service operated and developed by Google. It serves as the official app store for certified devices running on the Android operating system, allowing users to browse and download applications developed with the Android software development kit (SDK) and published through Google.
* App Store - A platform developed and maintained by Apple Inc., for mobile apps on its iOS and iPadOS operating systems. The store allows users to browse and download approved apps developed within Apple's iOS Software Development Kit.

## 1.4 References

DevSecOps Manifesto. (2020). Retrieved February 5, 2022, from

<https://www.devsecops.org>

Pham, I., Leung, V. (2021, March 30). DevSecOps TDD. University of Maryland Global Campus.

# 2. Software Architecture Design

**Continuous Integration/Continuous Delivery (CI/CD) and DevSecOps with GitHub**

The product suite GitHub created will help the DSO team navigate towards the new modern way of software delivery. With GitHub, we have a tool suite that we can use and have a secure, fully traceable, and easy‑to‑use toolset that we don't have to integrate, host, or maintain ourselves.

Highlights of the GitHub tools that the Spring ’22 DevSecOps team will incorporate into the production of the MemorEZ application are described in the following sections.

**Figure 1:**  **Overall Architecture Diagram.** GitHub Actions pipeline to App Store and

Google Play.

|  |
| --- |
| Image |

## 2.1 Automation

The process DSO will use for creating an automated workflow based on changes that

occur in the repository or issues that are tracked.

### 2.1.1 Code Repository

All development teams (RememberAll and FlutteringMind) are required to utilize the

GitHub code repository. The DSO team is responsible for the creation of the project and

appropriate repositories. Each team will have access to the appropriate repository with a

set of policies created by the DSO team. Please see the structure of the repository below.

**Figure 2: Branch Overview.** Read from the bottom up.

* The master branch is protected, which requires the approval of the DSO team for the merge from the development branch in order to check for code quality and security. Direct merge to master branch from the feature branch is not prohibited.
* The development branch is restricted to the lead developer or one required review from the team member.
* The feature branch allows developers to create a new branch for each feature being worked on.

### 2.1.2 GitHub Actions

|  |
| --- |
| Diagram  Description automatically generated |

The automation engine DSO will employ for automating any tasks we have in our

project.

### 2.1.3 Continuous Integration

When there is a commit on the main branch in source control and we run our build tools

to verify that we can integrate the change into the codebase without any issues. This then

results in a set of artifacts that we can use to run our software in production or run tests.

GitHub supports many languages and many tools that can be used to do the continuous

integration. Examples are Java, Ruby, and Python. GitHub is language agnostic, and can

host any type of source code we want and build our own automation around it to produce

the required executable artifacts.

**Figure 3:**  **GitHub Actions iOS CI/CD.** Development .yaml file for iOS CI/CD.

### 2.1.4 Continuous Delivery

GitHub also provides deployment automation through the use of GitHub Actions and the

concept of environments. GitHub Actions also provides automation to very easily deploy

|  |
| --- |
| Image |

to Google Play, the App Store, and to a website. With the build automation and

deployment automation, the DSO team can now realize both Continuous Integration

and Continuous Delivery or deployment.

**Figure 4:**  **GitHub Actions Android CI/CD.** Development .yaml file for Android CI/CD

### 2.1.5 GitHub Secrets

GitHub secrets are configured in the publish profile for deploying code to the App Store

and Google Play and then the secret is accessed in the workflow using the secrets

context. The secret name is available like a property on the secrets context. Secrets in

GitHub are encrypted environment variables that are created at the repository level or at

the organization level. When created at the organization level, you can share secrets with

multiple repositories under the organization, which reduces duplication.

Any sensitive value in your workflow should be stored in a secret so users with read

|  |
| --- |
| Image |

access to a repo, especially a public repo, can't see the value of your secrets. And secrets

can be anything, but you'll often use them for passwords, URLs, and API keys to outside

services, like deployment keys to the App Store and Google Play.

Once you create a secret, you can no longer access its value from the user interface. You

can only update the value, which overwrites the previous value. When you want to use a

secret in a workflow, you access it from the secrets context. When you want to make the

secret available to an action, you set the secret as an input or an environment variable.

GitHub automatically redacts the value of secrets printed to the logs, but that relies on

GitHub searching for common encodings of the values.

## 2.2 Package Management

The management of the breakdown of software into multiple components and host packages that are created for reuse inside or outside of the GitHub organization is referred to as package management. GitHub Packages allows sharing of project dependencies with the Spring 2022 repository or publicly. Because the project has a dependency on packages, it is important to be able to trust and understand their code.

### 2.2.1 Package Registry

GitHub Packages makes it easy to use the same familiar GitHub interface to find

public packages anywhere on GitHub, so GitHub can function as a package registry for

the project.

### 2.2.2 Package Manager

GitHub Packages is compatible with common package management clients to allow

consuming and publishing packages with a choice of tools. GitHub Packages is

compatible with package managers such as RubyGems, Maven and Gradle, that are the

two package managers for Java.

### 2.2.3 Container Registry

From complete applications to command line utilities, containers are another form of

distributing code. GitHub Packages also allows us to publish and distribute container

images. Once published in public or in private, these images can be used from anywhere,

including a local development environment, a step to execute in our continuous

integration or continuous deployment workflow with GitHub Actions.

### 2.2.4 Identity and Permissions

The project uses GitHub for hosting source code, Flutter/Dart for the app development,

and Gradle for the build tool. There are at least three different sets of user credentials and

permissions maintained, Git, Flutter, and Gradle repositories. With GitHub Packages, we

can use a single set of credentials across the source code repository. Packages

published through GitHub inherit the visibility and permissions assigned at the

repository. Therefore, if we have a team member that needs to have read access to the

packages, we can just give them read access to the repository.

## 2.3 Software Supply Chain

The software supply chain helps in maintaining the code base that is secure by default, and is always kept up to date. The Spring 2022 project software is built on top of software of many others by using all kinds of packages we can find in open source or from vendors.

### 2.3.1 Dependencies

All the dependencies are used will need to be kept up to date since the open source community or vendors will release new versions of the components used pretty frequently. When software has not been touched for a few months, the number of updates for packages can get high and sometimes they're not always fully compatible.

### 2.3.2 Vulnerabilities

We on DSO want to know if there are known vulnerabilities that we want to mitigate as soon as possible. And we want to know if all our packages are still the latest and greatest. Keeping software up to date will save a lot of work compared to delaying such updates and then run into a pile of technical debt that can build up. Therefore to keep things secure and safe, we need to keep all our dependencies up to date, and we need to scan for known vulnerabilities in our software. GitHub provides such tools and some of them for free.

# 3. Deployment Platforms

Developers of the MemorEZ app will want to deploy the application to Google Play and Apple App Store. In the sections that follow, we will describe how to set up a Google Play Store, create an application, and prepare an account to accept an Android app. We will also build an Android and deploy it to Google Play Store. DSO will describe setting up an App Store, creating a developer account in Apple, and also build and deploy our iOS app.

To upload an Android app, we need to have a Google Developer account under the Google Play console. Two separate Google Play accounts have been paid for and created by Andrew Nicolette and Robert Wren for the purposes of this course. After we successfully created our accounts, we are able to log in to the Google Play console dashboard.

To upload an iOS app, you need to join Apple Developer Program. An App Store account have been paid for and created by Robert Wren for the purposes of this course. After successfully create this account, Robert Wren is able to log in to both the developer account and App Store Connect dashboard.

## 3.1 Deploy to Google Play

DSO will create an application in the Google Play Store, and we will complete the application process to make it ready to upload the Android MemorEZ app. Then, we prepare the app for code signing and build our Android MemorEZ app using Flutter, and upload that to the Google Play Store and publish it to our internal tester and to production.

### 3.1.1 Build and Release the Android App

After logging into the Google Play console, we need to create the first app. First, fill in several fields such as App name, Default language, and if the app is free or a paid version and accepting developer program policies and US export laws. In the dashboard of the app, follow the several steps to set up the application and complete the app access. This is an important step when it comes to providing credentials such as login access to the app. Add instructions on how Google can access your content and review your app.

In the next step, be clear about advertisement in the app. Make sure to indicate which of these options are suitable for the app.

In the next step, follow the questionnaire, which shows what is the rating for the app. There are several questions, such as what is the category of this app or how you communicate in the app.

Define the target audience. Follow all the steps and answer all the questions that are appropriate for the application.

Next, provide a category and contact details. When the app is approved and available on Google Play, then it will be shown to all users.

In the last step, provide all the information that is going to be used in the public facing of the app in Google Play. Information such as app details, descriptions, icons, graphics, and screenshots, must at least three screenshots for phone and one per each tablet size, 7 in and 10 in. At this point, the app is ready to upload as an Android artifact that is created in the Google Play console.

Now, make the application ready for a production build. First, generate the launcher icons. To do that automatically, add flutter\_launcher\_icons package to dev\_dependencies. Specify the Android launcher\_icon name and image\_path to the icons that you want to generate launcher\_icon from. Run the flutter\_launcher\_icons command to generate all the icons.

To publish on the Play Store, give the app a digital signature. To do that, create a key store. Follow all the questions, answer, add the password, and make sure to keep this file somewhere very safe. This is the private key. Very important not to publish it to a public source control. Then, under the android folder, create key.properties file and add all of the properties that are necessary for Gradle to build the application, including storePassword, keyPassword, keyAlias, and storeFile.

Open the build.gradle file under the src folder under app. Then, add a code block before the android block with the keystore information from the properties file. This loads the key.properties file into the keystore properties object.

Then, add signing configuration info before buildTypes block, and change signingConfig from debug to release. In the defaultConfig, make sure the applicationId is a unique ID that can be used in Google Play.

Next, there are two options. Either build an apk or an appbundle. Building with appbundle is preferred. Once the flutter build appbundle is completed, the app‑release.aab file appears under build, app, outputs, bundle/release. This is the production app and is ready to upload to Google Play.

There are several options in Google Play to release the app. The fastest way to release is using internal testing. Internal testing can have up to 100 internal testers. When a new release is created for the first time, there will be a Play App Signing notification. At this point, the aab file or apk file is able to be uploaded.

After a successful upload, the release name, which is a build number plus a version number, is shown. It is important to have versioning on your app. It is absolutely important to have a unique build number. Whenever you upload a new artifact, it must have a new build number.

Create a mailing list for the internal testers. To do that, go to the Testers tab in Internal testing, create a mail list, add users with their emails, and, at the end, send them a URL that they can join to the program and start testing the application. Almost the same process can be followed for closed testing, as Google calls it the alpha track, or open testing, as they call it beta track. At any point, you can promote one release to another higher track, for instance from internal to closed, from closed to open, or at the end to production. After creating a release for production or promoting one version, it will be rolled out to Google Play, and it will be accessible to everyone.

## 3.2 Deploy to App Store

DSO will create an application in the App Store and complete the whole process, and we will prepare this application to be able to upload the MemorEZ iOS app. Then, we prepare code signing. After building our iOS app, we will upload our artifacts to App Store Connect.

### 3.2.1 Build and Release the iOS App

After logging into the developer account, go to Certificates, IDs, & Profiles. Then,

register the one identifier for the application. Select Explicit in Bundle ID. And type the

unique bundle ID. At the end, register it.

Next, go to App Store Connect. Go to My Apps and create the app. Fill all the necessary

information and select the bundle ID just created in the previous step. A unique SKU,

which could be similar to the bundle ID, can also be added.

Now that the app is created, complete the necessary information. Start with TestFlight.

TestFlight makes it easy to invite users to test the app and collect valuable feedback

before releasing the app on the App Store. Fill all the required information.

Once all is done, save, and go back to App Store tab. There is some information that

will needed to be completed, Pricing and Availability, App Privacy, and Rating and

Reviews. Make sure to upload all the screenshots according to the Apple guidelines. And

finally, provide promotional text, a description, keywords, support, and marketing URLs

for each version to be released, and it will be available publicly on the App Store.

Now that App Store Connect is prepared for both TestFlight and the production release,

prepare the app and build it. Generate icons using the flutter\_launcher\_icons package to

automatically generate all the necessary icons. Pass correct values to flutter\_icons

configuration. And run the flutter\_launcher\_icons command.

On the Mac computer, ensure Xcode is installed and configured. In the ios folder, open

Runner.xcworkspace. In Xcode, click on Runner. Select runners under TARGETS. Open

the General tab. Type a proper display name for your app. Add a unique bundle identifier.

This is the one that we have added in the Identifiers tab in developers account. Modify

the version and build number every time a new bundle will be uploaded to the App Store.

The build number must be unique. Go to Signing & Capabilities. In most cases, if you

enable automatic signing, it should be enough for to build and sign the application.

Select your team and you add your bundle identifier, the rest must be automatically done

by Xcode.

Next, move to the Build Settings tab and search for the iOS deployment target and build

the iOS app. If automatically signing code does not work as expected, resolve it

manually. In the developer account, go to Certificates and add a new one. Select Apple

Distribution in the list if using Xcode higher than 11; otherwise, iOS Distribution.

In the next step, upload a certificate signing request. To generate a CSR file,

open Keychain Access on the Mac computer. Go to Certificate Assistant and request a

certificate from a certificate authority. Complete the form and check Saved to disk.

Continue and save this file on your disk. Now upload the CSR file that was just

generated. Continue and download this file to disk. After the file is saved, open and add it

to the keychain.

Next, go to Profiles in developers account. Add a new profile. Since a distribution

provisioning profile is needed, select in the App Store under Distribution. In the

next step, select the App ID that this profile is associated with. Select the distribution

certificate that was just generated. And finally, generate the profile and download it to

disk. After this file is saved, double‑click and open it. Then, go back to Xcode, and this

profile name will appear.

Now build the iOS app. Once build is done, open Xcode and go to the Product menu.

Select Archive. Once the archive is done successfully, then it will appear in the Archives

panel. In the Archives panel, select the latest build and validate the app. Follow the steps

in the validation panels. Once all of the panels are ready, the Validate button will be

enabled. Hit the Validate button and wait until the application is successfully validated.

Next, distribute this version of the app. Select App Store Connect, and then select

Upload. The steps are similar to validation. Follow those steps until the Upload button is

enabled. It may take a few minutes to upload the file. Do not interrupt this panel, and wait

until the app is successfully uploaded. Go back to TestFlight tab in App Store Connect,

the status of the build should be processing. An email is sent when the build has

completed processing. Once the processing is completed, it's time to add this build for

testing or release it to production.

This build can now be added to the internal group testers. TestFlight testing internal or

externalgroups will receive an invitation. Once users receive this invitation, they should

follow the instructions provided in the invitation. And once they have download the

TestFlight application from the App Store, they can put the redeem code and install the

app on their phones.

After the app has been tested properly, then it can be promoted to production. Go to the

version that was created, select the build version to promote to production, and hit the

Submit for Review button. The review process may take a few days. Once it's approved,

you should be able to find the app on the App Store.