

Developer Manual

ChatVUE

Authors: Thomas Hansknecht, Noah Davis, Scott Conwell,
Jiaqi Liu

Sponsored By Nicholas Edwards



Tunnel VUE

Table of Contents

1.1 Overview.....	4
1.2 Set the Dependencies.....	4
1.3 Retrieving the Project.....	5
1.4 Setting up Visual Studio.....	7
2. Developing for Installers.....	9
2.1 Windows Installer.....	9
2.1.1 Overview.....	9
2.1.2 Configuring Visual Studio.....	9
2.1.3 Establishing Build Properties.....	12
2.1.4 Changing specific variables.....	13
2.1.5 Changing the user Interface.....	14
2.1.6 Customizing the Dependency Requirements.....	21
2.2 Mac Installer.....	22
2.2.1 Version 1 Installer.....	22
2.2.1.1 Creating the dmg file.....	23
2.2.1.2 Extra Applications used for install.....	25
2.2.2 Version 2 Installer.....	27
2.2.2.1 Creating the .pkg file.....	28
2.2.2.2 Signing the Installer.....	29
2.2.2.3 The build command.....	30
2.2.2.4 The directory structure.....	31
2.2.2.4.1 The build files.....	32
2.2.2.4.2 The source files.....	33
2.2.2.4.3 The post install sequence.....	36
2.3 Linux Installer.....	39
2.3.1 Red Hat Installer.....	40
2.3.1.1 Configuring the RPM Build.....	40
2.3.2 Debian Installer.....	43
2.3.3 Customizing Install Location.....	44
3. Developing for Chat UI.....	45
3.1 Setting up the keyfiles.....	45
3.2 ChatUI Project.....	46

3.3 Admin controls from UI.....	56
4. Key Server Admin.....	56
5. Contributions.....	57
6. Extra Resources.....	57

1.1 Overview

Welcome to ChatVUE! Before looking through this development manual, you might find it useful to first look through the Spring 2021 manual, as it gives more details about the more fundamental parts of the project, such as CORECryptography and CORENet. The main tasks that we aimed to accomplish during our semester was to create an Installer for all ChatVUE components, a GUI for connecting to Chat Servers and sending and receiving messages, as well as a GUI for administering the Key Server. While we made progress towards these goals, there is still plenty of work to be done before they are finished.

1.2 Set the Dependencies

This project uses .NET Core 5.0. It also uses the following NuGet packages:

- CliWrap 3.3.0 (<https://www.nuget.org/packages/CliWrap/3.3.0>)
- Microsoft.Win32.Registry 5.0.0
(<https://www.nuget.org/packages/Microsoft.Win32.Registry/5.0.0>)
- System.Management 5.0.0
(<https://www.nuget.org/packages/System.Management/5.0.0>)

The test suite code uses the following NuGet packages:

- coverlet.collector 1.3.0
(<https://www.nuget.org/packages/coverlet.collector/1.3.0>)
- Microsoft.NET.Test.Sdk 16.7.1
(<https://www.nuget.org/packages/Microsoft.NET.Test.Sdk/16.7.1>)
- xunit 2.4.1 (<https://www.nuget.org/packages/xunit/2.4.1>)
- xunit.runner.visualstudio 2.4.3
(<https://www.nuget.org/packages/xunit.runner.visualstudio/2.4.3>)

1.3 Retrieving the Project

The ChatVUEClient Github repository is the main way to develop for this project. There should be a zip of the project on pCloud with our final commits if anything goes awry with Github. The project can be cloned from Github using the instructions shown in the README. As of the end of our tenure, there are three different branches that contain the totality of the work that we have done this semester. The Key Server admin work is present in the scooter branch, Installer work is present in avaloniaUI, and the work on optimizer keys, chat gui, and fingerprinting are present in the Optimizer-Code branch. The master branch includes the final code submitted by the Spring 2021 group.

To clone the repository simply visit the Git-Hub Repository which can be found at <https://github.com/edwanic/ChatVUE-Client>

From there clone the appropriate branch which can be done with the -b flag. For instance if we want to clone the branch avaloniaUI then we will use the command

```
$ git clone https://github.com/edwanic/ChatVUE-Client -b avaloniaUI
```

Note: Permission will be needed to access and modify this private Git-Hub Repository.

With the Git-Hub repository it's easy to see the changes made to the code and view new code added by the previous teams in branches.

Layout of the Git-Hub Location

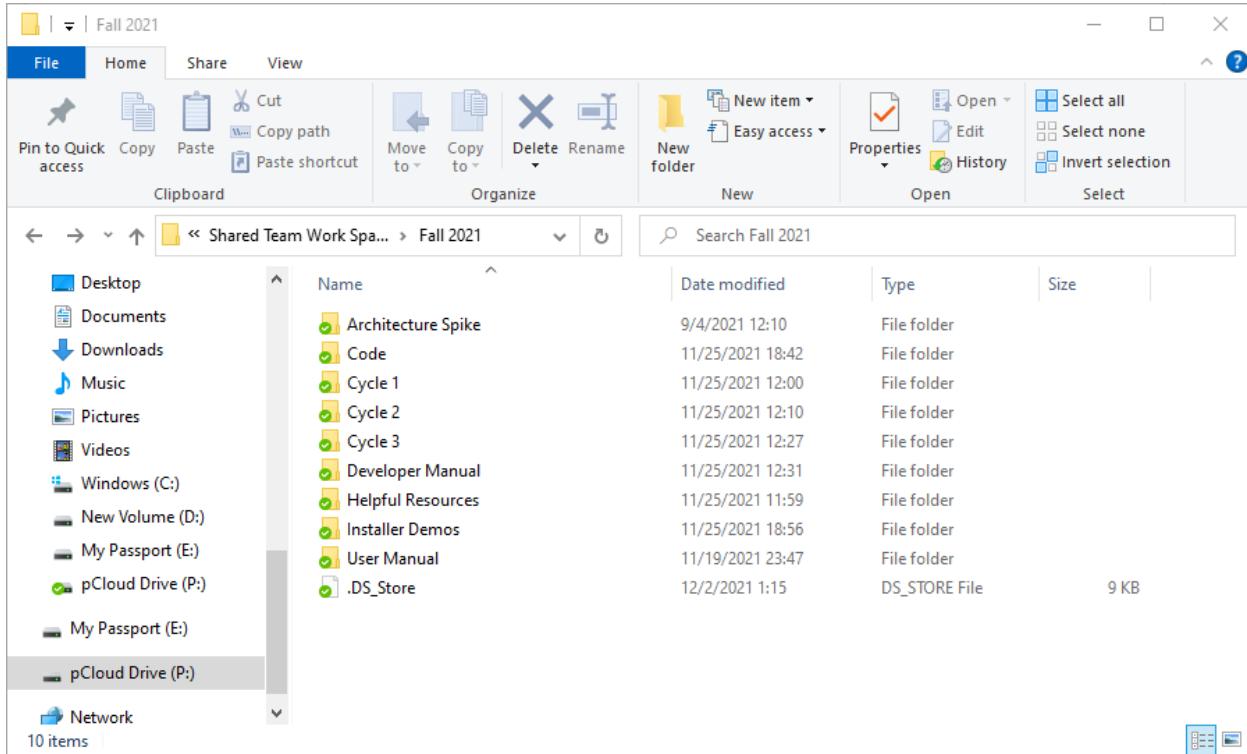
The screenshot shows the GitHub repository page for `edwanic / ChatVUE-Client`. The repository is private. Key statistics shown include 12 issues, 1 pull request, 1 project, and 0 forks. The repository is described as a central repo for the ChatVUE client/server application. It features a README and a license. The code section lists recent commits by `tfh007`, including updates to README.md, .actions/chatvue/build, .github/workflows, CORECryptography, CORECryptographyTests, CORENet, ChatClient, ChatServer, ChatUI, Database, Docs, InstallerApplication, KeyServer, .gitignore, ChatServer-MSSQL.sql, ChatVUE.sln, LICENSE.md, and README.md. The repository has 12 branches and 0 tags. The sidebar also displays information about releases, packages, contributors (jaller200, ingeonus, Omine100, tfh007), and languages (C# 99.5%, TSQ 0.5%).

There are certain parts of the Project mainly the Mac Installer 1 dmg file in addition to several installer scripting files that are not found on the Git-Hub repository linked above. To access the remaining files please visit the PCloud file location that has been set up by Nicholas Edwards.

A backup of the Git-Hub repository is also available in case anything happens to the Git-Hub repository. The backup Git-Hub repository and all other Fall 2021 files can be found in the PCloud under Auburn Class 10/Team Resources/Shared Team Work Space/Fall 2021/.

In addition the pCloud location contains Installer Demo files which contain already developed installers.

Layout of the pCloud file location



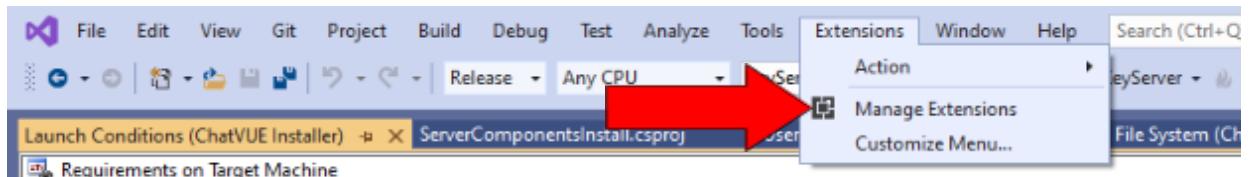
1.4 Setting up Visual Studio

Visual Studio is the preferred IDE for development of this project. To work on the ChatUI GUI the Avalonia for Visual Studio extension is needed from the marketplace. To work on Windows Installer the Microsoft Visual Studio Installer Projects extension is needed from the marketplace. The additional Visual Studio set up requirements for Windows Installer development are discussed in Chapter 2.1.2.

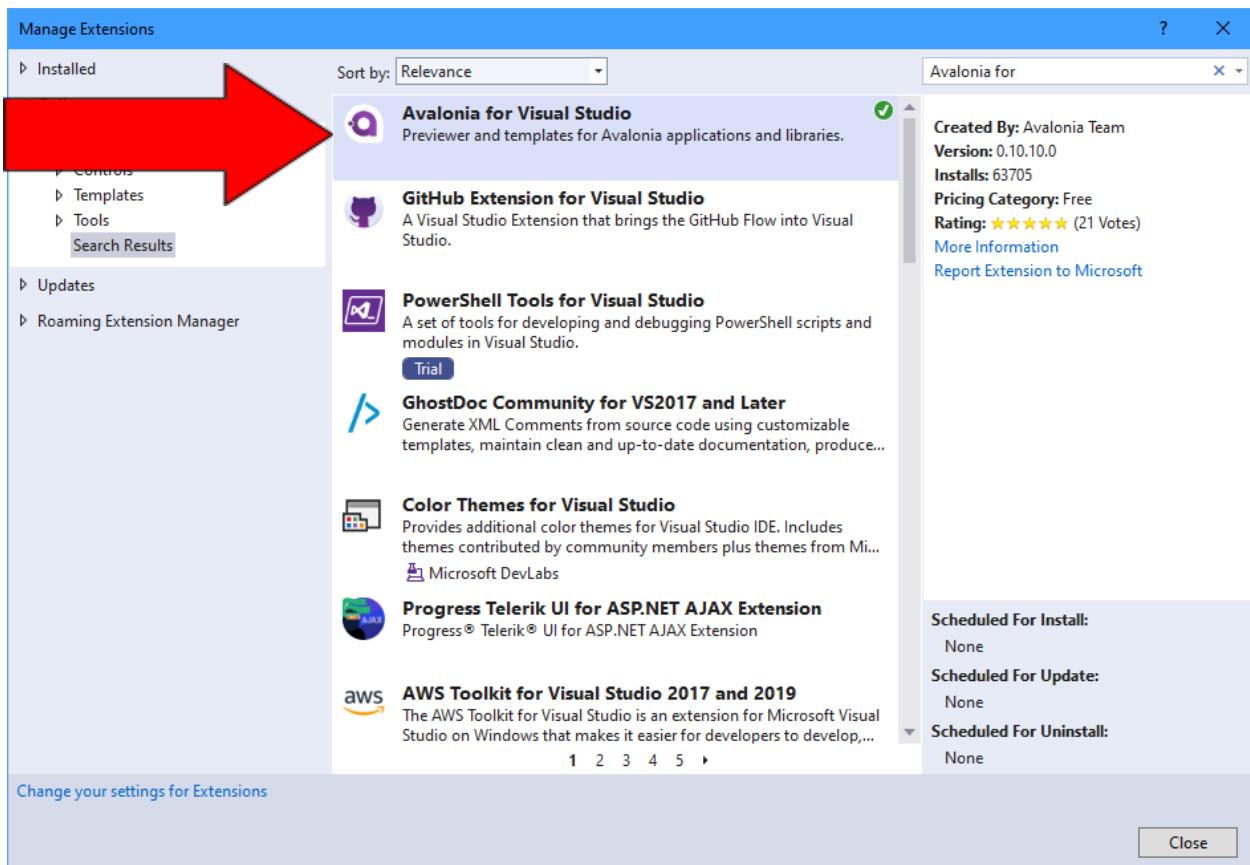
To develop for the ChatUI there is a visual studio project called ChatUI.csproj. Since ChatUI.csproj is dependent on the ChatVUE.sln project it is recommended to develop the ChatUI from the ChatVUE.sln visual studio project.

Note: In the event the ChatUI project does not show up under the ChatVUE.sln simply add the ChatUI.csproj to the solutions under the ChatUI project.

To install the Avalonia extension within visual studio simply launch visual studio then click Extensions from the toolbar followed by Manage Extensions.



Finally search for Avalonia for Visual Studio and install it to Visual Studio.



2 Developing For Installers

This project has installer applications to help install the needed dependencies, build all project files and assist the user in making the server and client applications functional. The installation process in the beginning is different for every OS. Once dependencies, ChatVUE files have been placed in the appropriate directories, and shortcut applications are created the install process between Mac and Windows becomes the same as we transition to the AvaloniaUI for the rest of the install which is made exclusively for assisting users and admins in setting up, managing, and connecting to servers.

2.1 Windows Installer

2.1.1 Overview

For Windows the installer is built as an msi file to be easily installable on Windows. The optional shortcuts and server components which are given to the user as optional additions are only visual at this time and do not currently interact with the backend components

The windows installer works by building all of the project files specified in the folders of the (ChatClient,ChatServer,ChatUI, and KeyServer,InstallerInterface) This process is done automatically by the installer and no scripting needs to be done by another application.

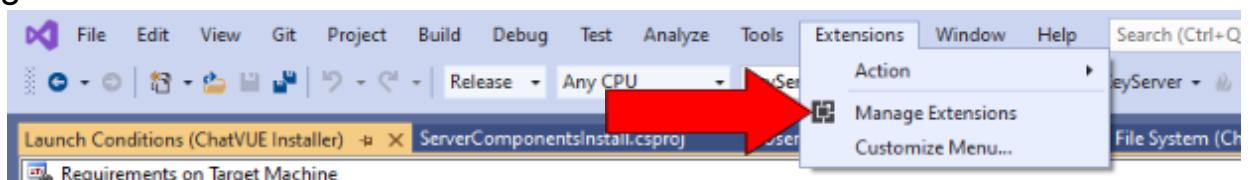
2.1.2 Configuring Visual Studio

To develop for the Windows Installer there is a visual studio project called ChatVUE Installer.sln

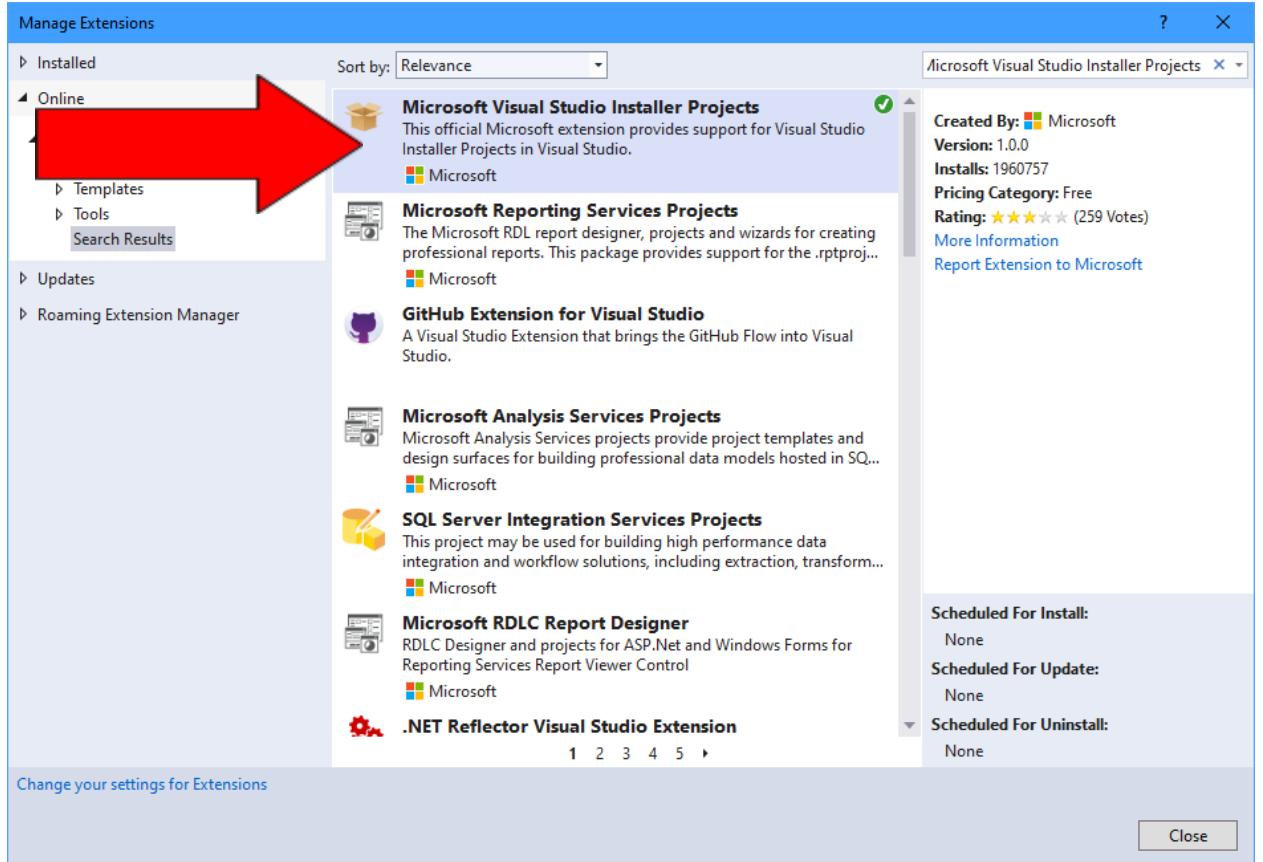
PC > Documents > ChatVUE-Client > InstallerApplication > InstallerV0.2 > setup Files > ChatVUE Installer

Name	Date modified	Type	Size
.vs	11/9/2021 18:01	File folder	
ChatVUE Installer	11/19/2021 12:10	File folder	
LicenseFiles	11/19/2021 0:42	File folder	
ChatVUE Installer.sln	11/19/2021 1:29	Visual Studio Solu...	9 KB
UpgradeLog.htm	10/20/2021 18:29	Chrome HTML Do...	38 KB
UpgradeLog2.htm	11/9/2021 17:21	Chrome HTML Do...	29 KB
UpgradeLog3.htm	11/9/2021 17:24	Chrome HTML Do...	29 KB

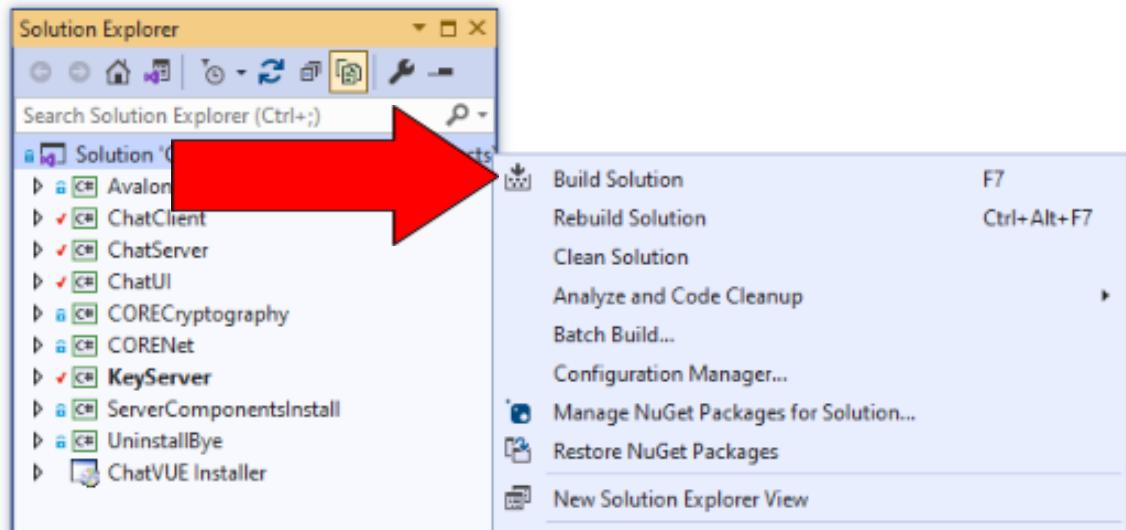
Since ChatVUE Installer.sln was created using a visual studio extension this extension will need to be installed to develop and build the ChatVUE Installer application. To install the extension within visual studio simply launch visual studio then click Extensions from the toolbar followed by Manage Extensions.



Finally search for Microsoft Visual Studio Installer Projects and install it to Visual Studio.



After installing the extension you are now able to develop and build the installer. Simply launch the **ChatVUE Installer.sln** and under the main solution right click (secondary click) and then select build/rebuild Solution.

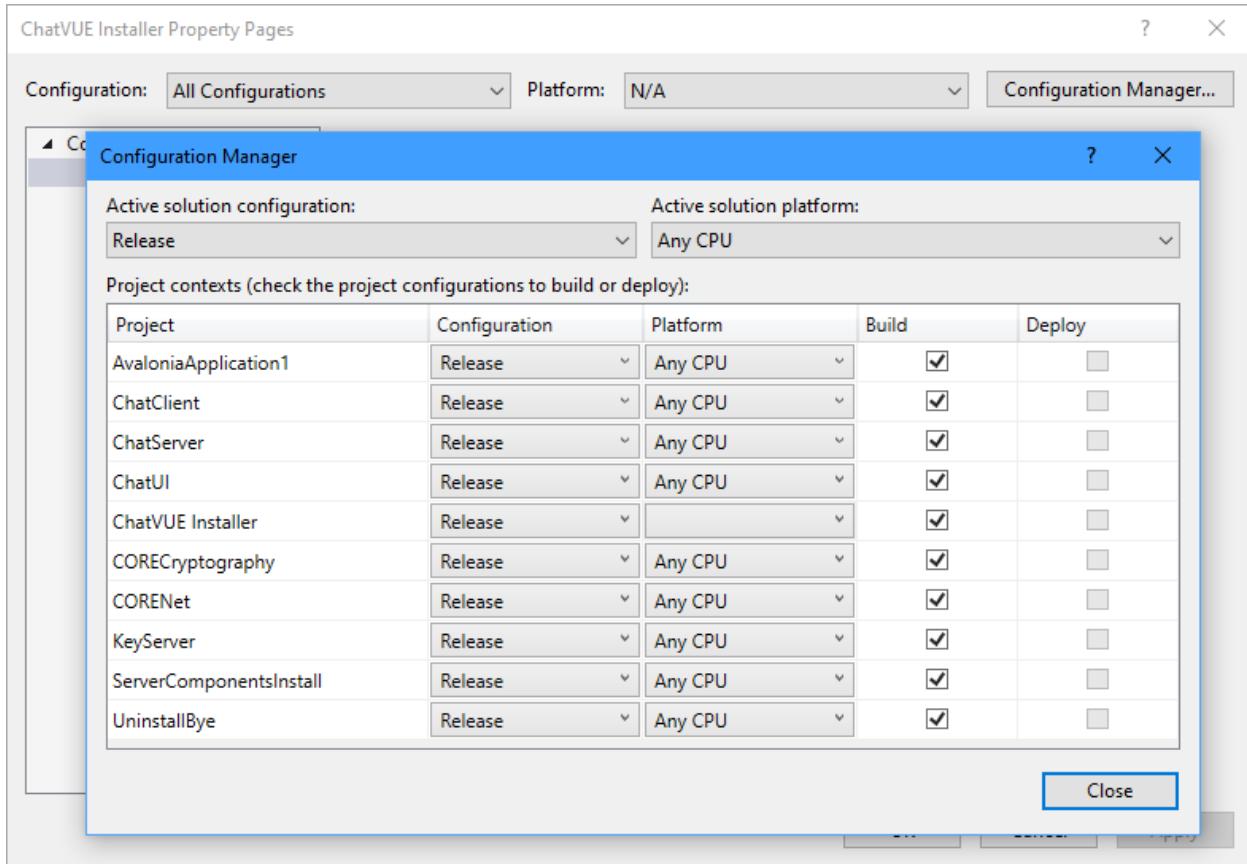


Note: It sometimes works to simply build the ChatVUE Installer solution

(The final solution) but this is risky and often causes compilation errors. To ensure the installer builds successfully it is best to build the entire solution using the process shown above.

2.1.3 Establishing Build Properties

The majority of the build properties are accessed by right clicking (secondary clicking) the ChatVUE Installer solution. From here click on the Properties option. Next Click on the Configuration Manager. Afterwards there are several different options for changing/updating the build properties including changing CPU configurations, deployment options, as well as specifying Debug/Release Properties.



A more custom way of affecting the build properties is by making manual changes to the ChatVUE Installer.vdproj file which can be found at

ChatVUE-Client/InstallerApplication/InstallerV0.2/setup Files/ChatVUE Installer/ChatVUE Installer.

PC > Documents > ChatVUE-Client > InstallerApplication > InstallerV0.2 > setup Files > ChatVUE Installer > ChatVUE Installer

Name	Date modified	Type	Size
Debug	11/9/2021 18:23	File folder	
Release	11/19/2021 12:10	File folder	
ChatVUE Installer.vdproj	11/19/2021 1:29	VDPROJ File	77 KB
setup.exe	10/28/2021 20:22	Application	540 KB
To edit Company info examine the vdpro...	11/9/2021 18:31	Text Document	0 KB

2.1.4 Changing specific variables

To change specific variable the developer will need to make changes to the ChatVUE Installer.vdproj file which can be found at

ChatVUE-Client/InstallerApplication/InstallerV0.2/setup Files/ChatVUE Installer/ChatVUE Installer. See 2.1.3 for more details. Within the ChatVUE Installer.vdproj file developers can change any number of variables from the Company/Manufacturer name of the installer. Changing the text that displays in the Installer, Changing build properties, and a lot more.

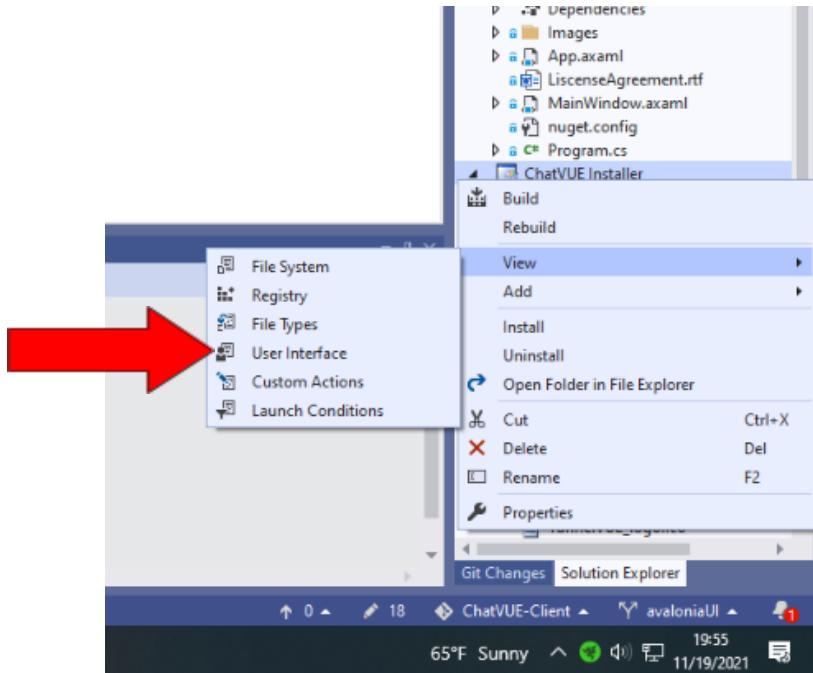
Warning: When making changes to the .vdproj file the developer needs to be careful as the visual studio solution will not make changes to this file but the project does rely on it. Thus if the developer breaks the file the entire installer solution will be broken as well so it is imperative to either make a backup of the .vdproj file or be careful when making changes to it.

As an example we want to change the Product Name and Manufacturer. We can easily do this in the .vdproj file by changing the Variable ProductName as well as the Variable Manufacturer.

```
570      [
571      "Name" = "8:Microsoft Visual Studio"
572      "ProductName" = "8:ChatVUE"
573      "ProductCode" = "8:{1EEE3BA0-BB96-49D2-85E6-A20BD46ABEA6}"
574      "PackageCode" = "8:{F98E75A5-C20F-4DB3-897C-76F89E6693B5}"
575      "UpgradeCode" = "8:{9021EBAD-0B4D-4108-89CD-BF92689A6A0F}"
576      "AspNetVersion" = "8:4.0.30319.0"
577      "RestartWWWService" = "11:FALSE"
578      "RemovePreviousVersions" = "11:FALSE"
579      "DetectNewerInstalledVersion" = "11:TRUE"
580      "InstallAllUsers" = "11:FALSE"
581      "ProductVersion" = "8:1.0.0"
582      "Manufacturer" = "8:TunnelVue"
583      "ARPHELPTELEPHONE" = "8:"
584      "ARPHELPLINK" = "8:"
585      "Title" = "8:ChatVUE"
586      "Subject" = "8:"
```

2.1.5 Changing the user Interface

To make changes to the user interface we use visual studio or the .vdproj file. See 2.1.3 for more details on the .vdproj file. With the visual studio implementation we use the User Interface menu. Simply right click (secondary click) on the ChatVUE Installer solution and click User Interface to customize the UI that is present in the Installer interface.

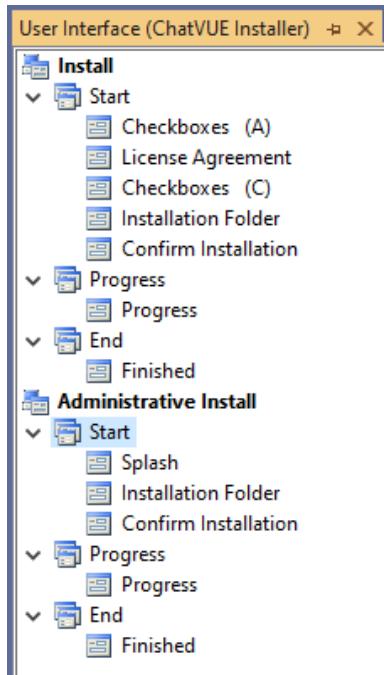
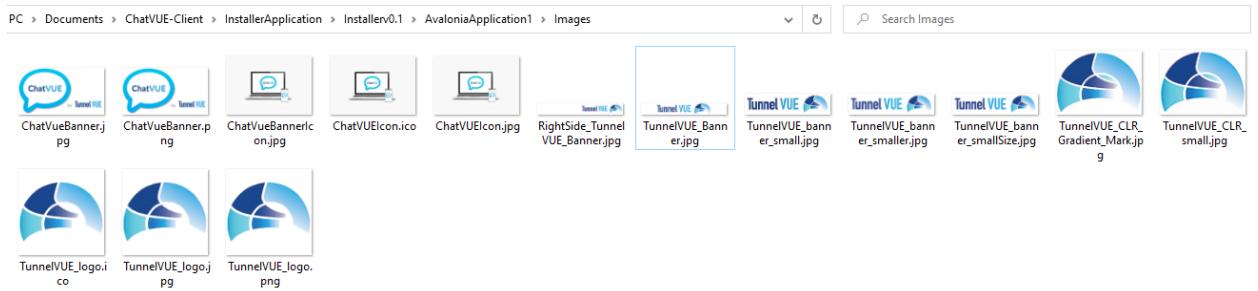


From the User Interface menu elements have different properties that can be used. Such as windows having Titles, body text, etc. To view the properties of a particular element simply right click (secondary) click on an element and select Properties Window on that element. To add new elements to the user interface simply right click (secondary click) on the Start element

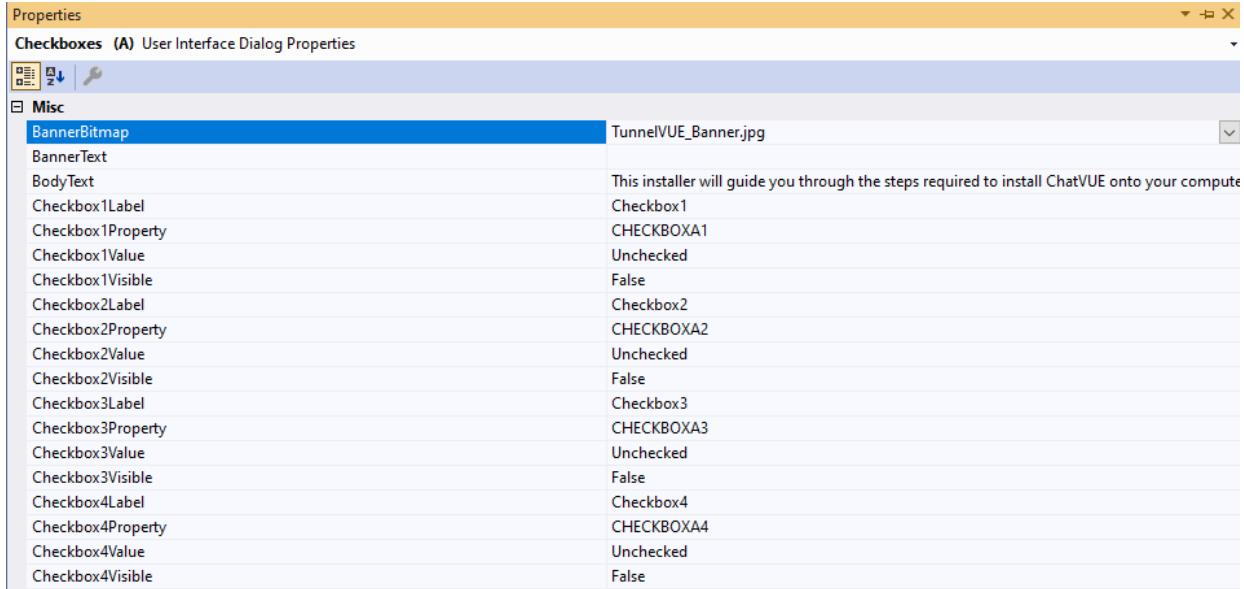
Note: There are two separate Start elements. One is for the normal installer while the other is for administrative only installs. The majority of attention has gone towards building the Regular Install since this is the Install that will appear to a normal user.

To change the graphics in the User Interface one can look at the properties Window of a given page and make changes to the BannerBitmap. The exact size of the Banner bitmap needs to be 500x70 pixels where 500 is the Width and 70 is the Height. The Banner bitmap also needs to be in a .jpg formula as png does not seem to work here.

**Currently the image graphics for the Windows installer are located at:
ChatVUE-Client/InstallerApplication/Installerv0.1/AvaloniaApplication1
/images**



As an example here are the properties of the current built Checkboxes (A) window



In addition to the Visual Studio Installer there is a post install sequence which is known as the **AvaloniaApplication1**. When Examining the ChatVUE Installer.sln file several of the solutions will be the **Installerv0.1** folder which is found in the **InstallerApplication/Installerv0.1** directory.

InstallerApplication > Installerv0.1		Search Installerv0.1		
Name		Date modified	Type	Size
.vs		10/13/2021 18:44	File folder	
AvaloniaApplication1		11/18/2021 23:10	File folder	
ChatVueDesktopInstaller		9/18/2021 0:48	File folder	
ExtraHelpInstall		11/19/2021 0:56	File folder	
UninstallBye		11/19/2021 1:49	File folder	
Installerv0.1.sln		9/17/2021 22:26	Visual Studio Solu...	2 KB

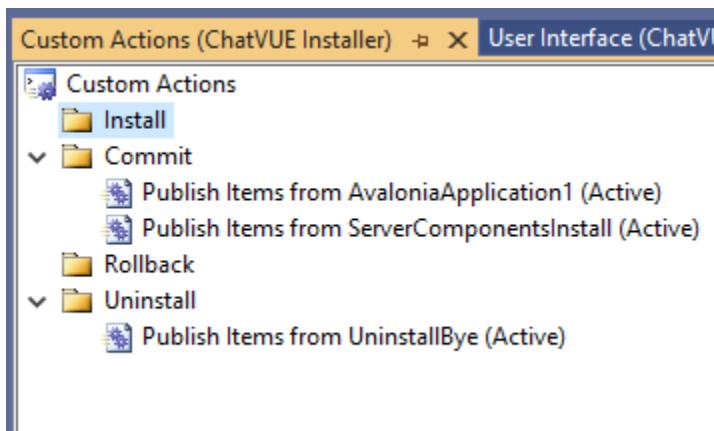
AvaloniaApplication1 is a misleading folder as it actually contains the **ServerComponentsInstall** project which is the UI for server based installs after all of the project files are built and the initial install is complete. Similar to **ExtraHelpInstall** the **ServerComponentsInstall** project is simply help documentation to advise admins on future steps for getting their system up and running with the ChatVUE and KeyServer projects. Currently there is

no backend attached to this project and it is strictly a visual application that will launch as a commit application (meaning when the installer is run the ServerComponentsInstall application will run during the install) The install process will not complete until all windows of the ServerComponentsInstall application are closed.

ChatVueDesktopInstaller is a folder that is again not currently in use by the installer. One of the ideas for this folder is to have it store the actual installer msi files. However, the folder current contains nothing except to blank Debug and Release folders.

ExtraHelpInstall project is the UI for client based installs after all of the project files are built and the initial install is complete. Currently the ExtraHelpInstall project is a Avalonia UI that simply gives help documentation to advise users on future steps for getting their system up and running. Currently there is no backend attached to this project and it is strictly a visual application that will launch as a commit application (meaning when the installer is run the ExtraHelpInstall application will run during the install) The install process will not complete until all windows of the ExtraHelpInstall application are closed.

To add or remove projects from the user interface one can use visual studio and after right clicking (secondary clicking) the ChatVUE Installer hovering the mouse over View followed by clicking on custom actions. From here we can add custom scripts on entire projects to be displayed during the install/rollback/uninstall process



The **Install folder** will display/run the items put here after the install is finished

The **Commit folder** will display/run the items put here during the install process.

It is worth noting that with Commit the displayed/run items must execute correctly and not terminate with an error. If a Commit item is prematurely terminated or ends with an error the install will be reset and the install will be canceled while prompting an error message to the user.

The **Rollback folder** will display/run the items put here whenever the user tries to Rollback/Modify the install

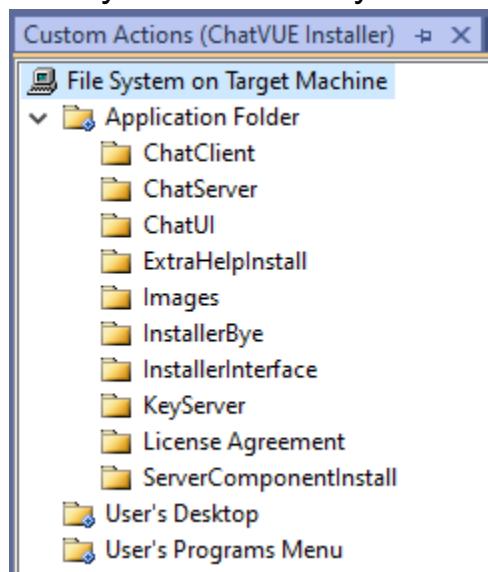
The **Uninstall folder** will display/run the items put here whenever the user tries to uninstall an already installed installation of ChatVUE.

It is worth noting that similar to Commit the uninstall will be prematurely terminated and canceled if an item put here does not exit normally.

Currently the way the Prevent Uninstall button works during the Uninstall sequence is that when the user clicks that button the avalonia application UninstallBye terminates on an System.Environment.Exit(1); which returns an abnormal exit thus the uninstall sequence fails and the uninstall is terminated as a result.

2.1.6 Customizing the User File Structure

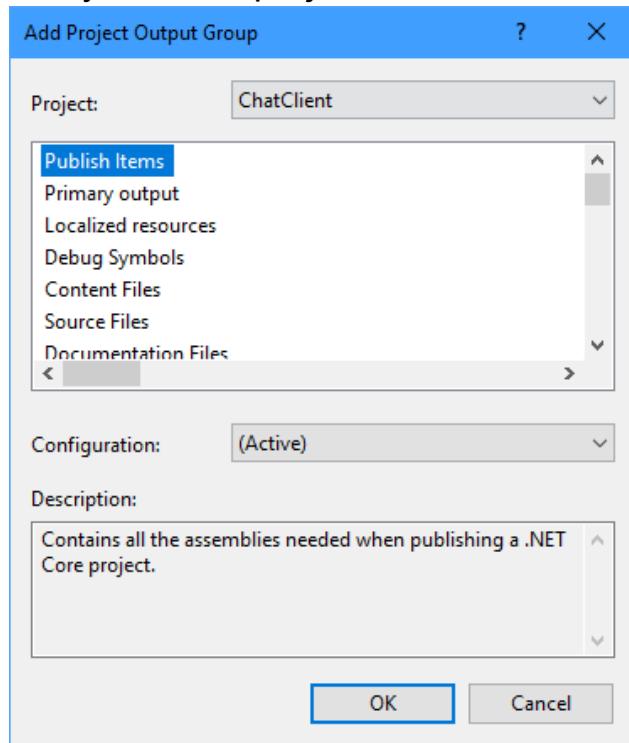
The file structure that is created on the users system can be customized in visual studio. Simply right click (secondary click) on the ChatVUE Installer solution. Then hover the mouse over the View button. Finally click on the File System. Currently the File structure is divided into several folders.



Application Folder specifies what files are placed in the Users program Files directory. Depending on the Architecture this could be Program Files x86 or simply Program Files (for 64 bit installs) The way this structure is set up currently is that all the project solutions are placed in their respective folders. For example to store the files necessary to install the ChatClient solution we have a folder named ChatClient. Within that folder we have an item called Publish items from ChatClient which as the name suggests builds the project solution and only puts the published items onto the users system.

To publish items of a solution simply open or create any folder and from within that folder right click (secondary click) then hover the mouse over add. Finally select Project output. Now the developer is presented with the list of options and a list of available projects the installer has access to. The developer can then select the project solution that they would like to add to the installer application.

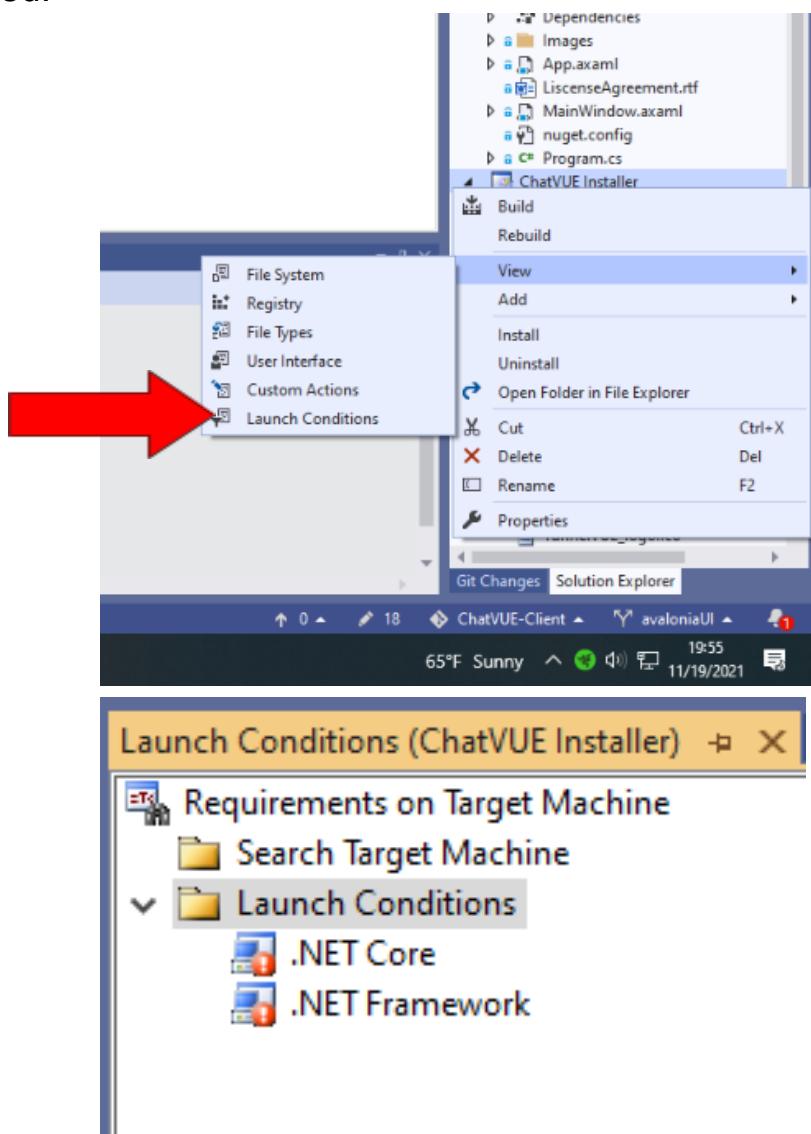
Note: To add solutions to this list one simply needs to add the desired solution they have to the Solution Explorer of the main Installer application project. Now the developer can perform the options listed below with their newly inserted project.



2.1.7 Customizing the Dependency Requirements

Before the installer runs it checks that the user has the appropriate .NET 5.0 framework installed to their system. When the user does not have .NET 5.0 framework installed the installer will terminate and prompt the user to install the appropriate version of .NET 5.0.

Customizing the dependencies is done in visual studio using the Launch conditions menu. Simply right click (secondary click) on the ChatVUE Installer solution and click Launch conditions to customize the dependencies that are forced to be installed before ChatVUE can be installed.



License Agreement folder:

Contains the TunnelVue License files (Not here currently) During the install a license agreement is required to be accepted by the users. This license agreement file will be stored in this folder location and should be called LicenseAgreement.rtf

User's Desktop, User's Programs Menu folder: (Files that are placed in user's directories)

Specifies what shortcut files to create. Currently the Chat UI shortcut is sent to both the user's desktop as well as the user's programs menu location.

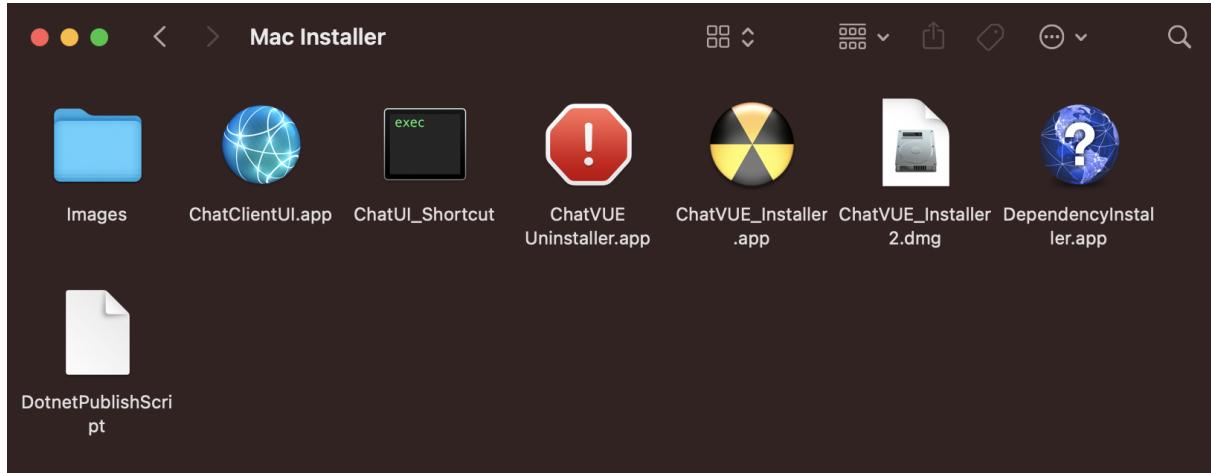
Uninstall Process

The uninstaller is built into the install application. If the user runs the installer while ChatVUE is already installed to their system then the user will be asked if they want to repair or remove ChatVUE on their system.

2.2 Mac Installer

2.2.1 Version 1 Installer

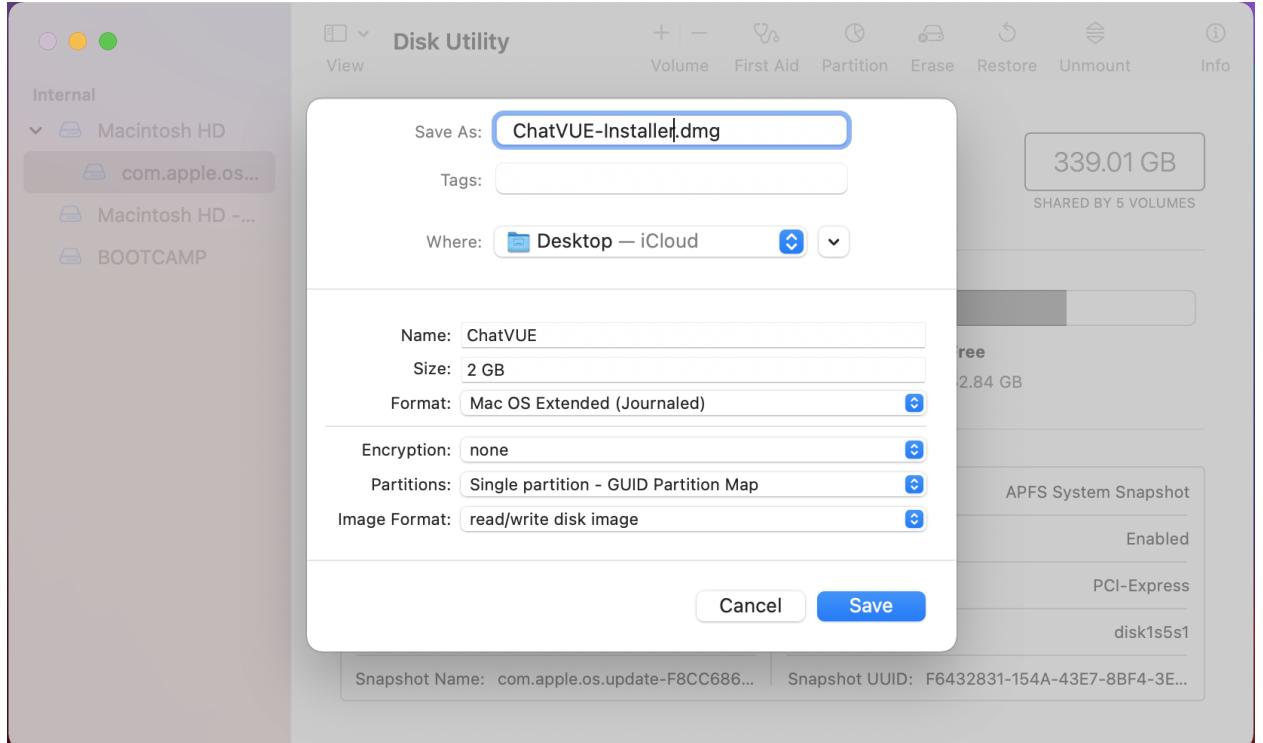
For Mac, the original installer is built in a dmg file. Currently the dmg file is large at around 5GB. Compression is recommended here. The dmg file can be compressed to around 500 MB but it seems to compress a dmg you have to create a brand new copy hence adding an excess of storage to store any additional files that need to be added in the future. The Mac installer consists of three main components. The dmg file, application files, and scripting files.



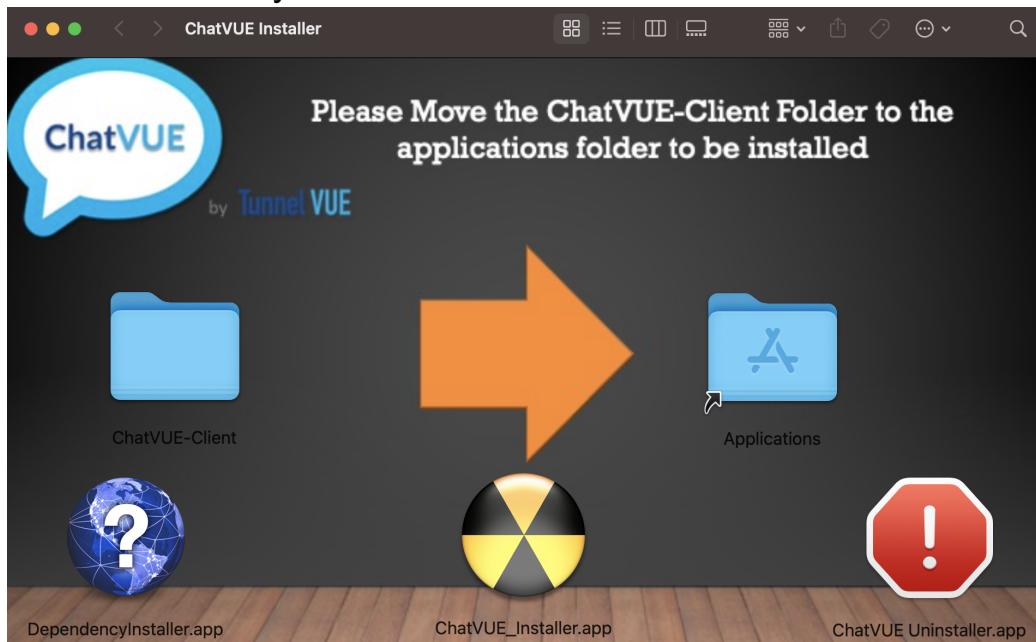
2.2.1.1 Creating the dmg file

The dmg file is made to be visually appealing and informative to the user. The dmg file has a shortcut to the applications folder which is where the ChatVue files need to be placed. There are three applications the user is presented with for the install. The names of the applications are meant to be informative and easy for the user to understand. All three applications were written using Apple automator combined with shell scripting.

To create a dmg file open the Disk Utility application. From there in the top menu click on File. Next hover the mouse over New Image. Finally select Blank Image. Adjust the properties as needed to create the appropriate dmg. The properties used to create the current developed dmg are the following aside from the Size property.



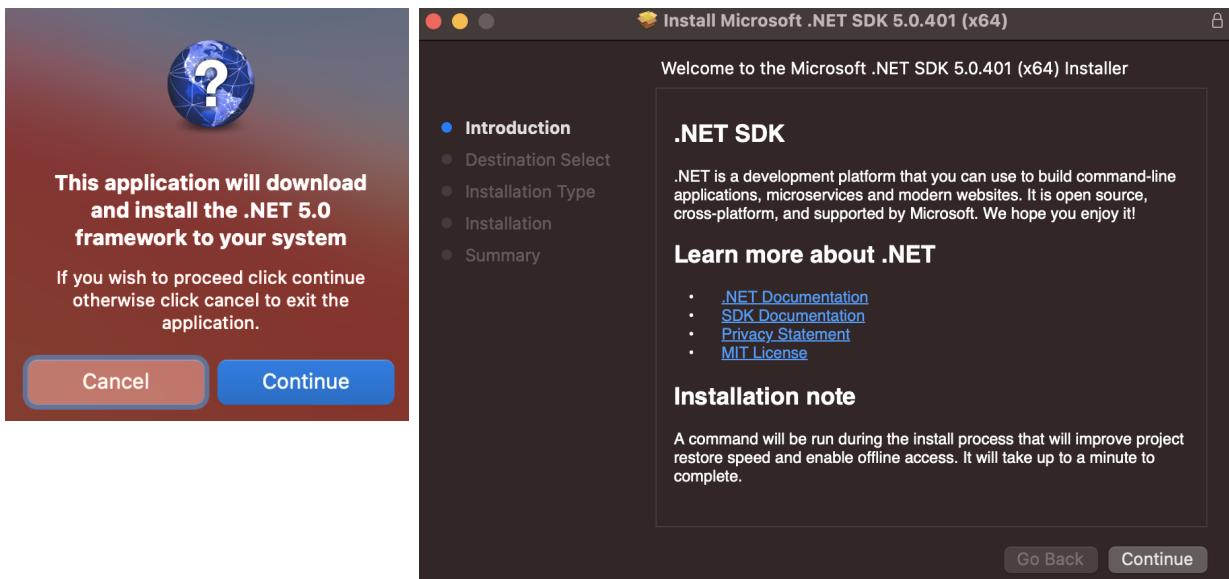
Dmg supposedly offer support to lock the dmg files so that when a user receives it they can not modify any of the properties but this implementation does not exist currently



2.2.1.2 Extra Applications used for install

DependencyInstaller.app (automator application)

This application when executed downloads the .NET 5.0 sdk for the user to install then it runs the .NET 5.0 sdk installer automatically and minimizes all other windows so the process is more obvious to the user. The .NET 5.0 sdk is grabbed from Microsoft servers and the normal .NET 5.0 sdk installer is presented to the user which was created by Microsoft.



ChatVUE_Installer.app (automator application)

This application when executed will first confirm with the user that .NET 5.0 sdk is installed. After this confirmation the application automatically runs the DotnetPublish script. There is a check in place to ensure the user has the shell open. If the user does not have a shell window open the DotnetPublish script will automatically open a new shell window. From there the script will build all of the .NET projects including (ChatClient, ChatServer, ChatUI, and KeyServer). The projects files are built using the following command:

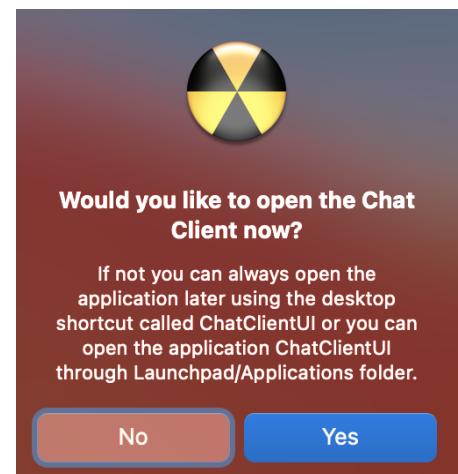
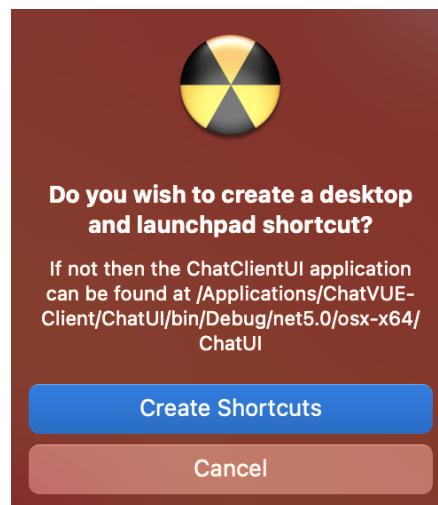
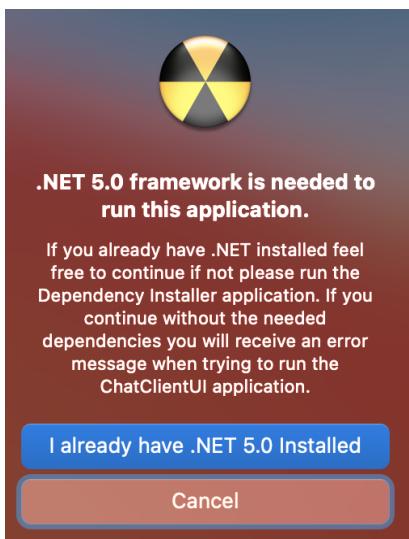
```
$ dotnet publish -r osx-x64
```

Currently there is no visual process to explain what is happening to the user during the .NET project files build process . This feature needs to be added otherwise the user will be either unaware or unsure of what is

happening. For testing and debugging purposes I have made the shell window visible but ideally there is a progress bar with details explaining to the user the current tasks being performed and the shell window is hidden in the background to not intimidate the user. During the install the user has the option of creating a desktop and launchpad shortcut to the ChatUI application. Without a shortcut the user will need to manually run the ChatUI dll file which is located in

/Applications/ChatVUE-Client/ChatUI/bin/(Debug or release)/net5.0/osx-x64/ChatUI

Since this directory is hard to find the user is warned of the difficulties when they choose to not add shortcuts. If shortcuts are made the user is given the option of opening the ChatUI at that moment which is asked as a question to the user.



ChatVUE UnInstaller.app (automator application)

This application when executed will send a confirmation message to the user to make sure that they want to uninstall ChatVUE from their system. If the user wants to, this application will first display a notification telling the user to wait for the uninstall process to complete. Then the application will run several apple scripts that first remove all ChatVUE files from the users system and then finally remove all shortcuts that were added including any desktop and launchpad shortcuts. After the process completes the user will get a notification informing them that the files and applications were deleted successfully.



ChatClientUI.app (automator app)

This application serves as an application link to the ChatUI dll file. Since the ChatUI dll file is not executable except by using shell we use an automator application to link to and open the ChatUI dll file. This application is the one that is placed in the Mac launcher and desktop.

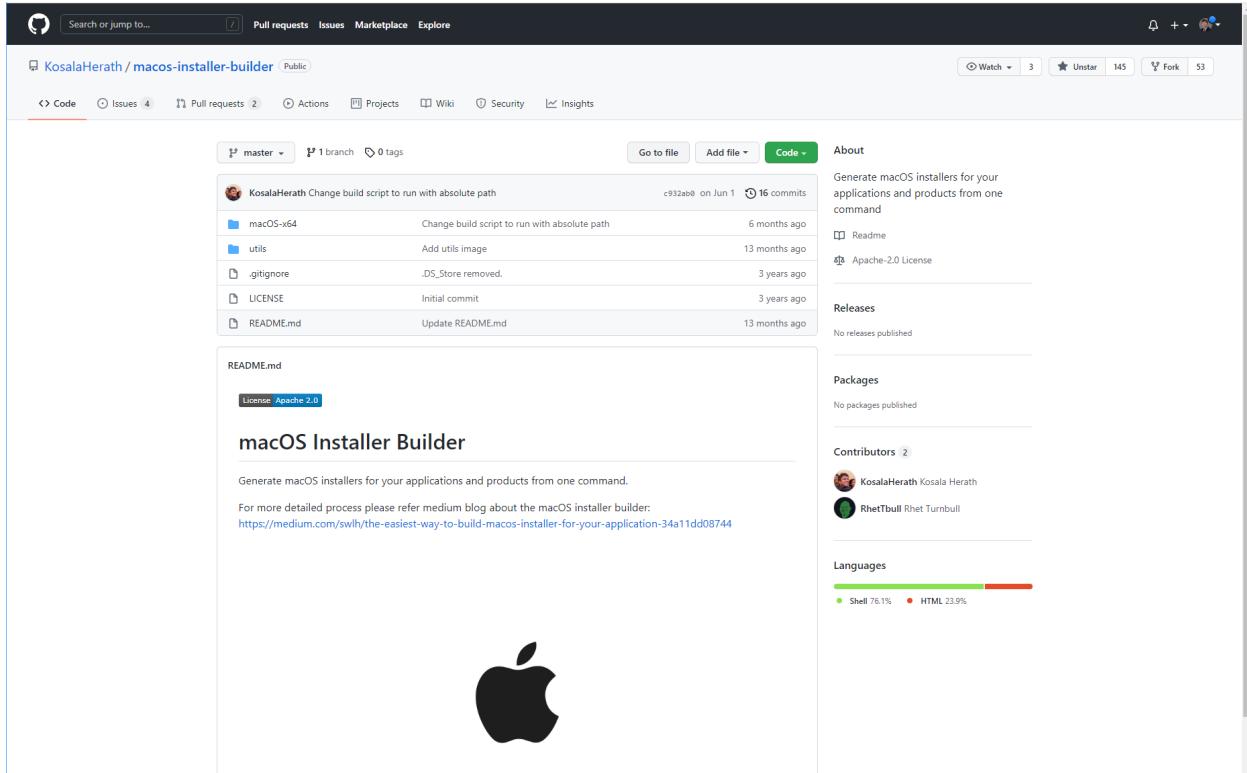
ChatUI_Shortcut (Shell script)

This script serves as a shell link to the ChatUI dll file. Since the ChatUI dll file is so difficult to find, this script can be placed in the main directory of ChatVUE so the user can easily run the ChatUI dll file directly using a shell window.

2.2.2 Version 2 Installer

For the version 2 Mac installer it is constructed in the folder labeled ChatVUE-Client/InstallerApplication/Mac Installer/macos-installer-builder. The template for the macos-installer-builder was created by Kosala Sananthana who has a blog on the code he wrote which can be found at <https://medium.com/swlh/the-easiest-way-to-build-macos-installer-for-your-application-34a11dd08744>

There is also a github repository for the installer template which can be found at <https://github.com/KosalaHerath/macos-installer-builder>



The basic idea is that we use Apple's .pkg builder through the bash command line while the scripting that Kosala has done streamlines the process. The current implementation has been heavily modified to include the needed ChatVUE files and the interface has been heavily modified as well to use TunnelVUE graphics in addition to custom pages.

Warning: The installer does not build inside of the ChatVUE-Client/... directory. To actually build the installer the macos-installer-builder folder needs to be placed in its own directory. It was found that the Documents directory works here. Thus putting the macos-installer-builder into the ~/USER/Documents folder will suffice and will allow the build command to function. The \$USER is simply a bash variable that means the current User's name.

2.2.2.1 Creating the .pkg file

First navigate the macos-installer-builder/macOS-x64 folder. Next open the terminal at this location and type the command

```
$ bash build-macos-x64.sh ChatVUE 0.0.1
```

Note: You may need appropriate permissions to run this shell script. To allow executing of the script type in terminal/bash

```
$ chmod build-macos-x64.sh u+x
```

Using this command we allow the user of the system to execute the script without needing to be the root user.

After running the build-macos-x64.sh script with the above parameters of ChatVUE and 0.0.1 then answering the questions asked during the build the developer should be shown a screen similar to the one below saying that the build was successful.



```
macOS-x64 -- bash -- 123x27
~/Documents/macos-installer-builder/macOS-x64 -- bash
(base) Thomass-MacBook-Pro-3:macOS-x64 thomashansknecht$ bash build-macos-x64.sh ChatVUE 0.0.1
[REDACTED] [REDACTED]
[REDACTED] [REDACTED]
[REDACTED] [REDACTED]
Application Name : ChatVUE
Application Version : 0.0.1
[2021-11-20 15:44:56] [WARN] Apache Maven was not found. Please install Maven first.
[2021-11-20 15:44:56] [WARN] Ballerina was not found. Please install ballerina first.
[2021-11-20 15:44:56] [INFO] Installer generating process started.
[2021-11-20 15:44:56] [INFO] Cleaning /Users/thomashansknecht/Documents/macos-installer-builder/macOS-x64/target directory.
[2021-11-20 15:44:56] [INFO] Application installer generation process started.(3 Steps)
[2021-11-20 15:44:56] [INFO] Application installer package building started.(1/3)
[2021-11-20 15:44:56] [INFO] Application installer product building started.(2/3)
Do you wish to sign the installer (You should have Apple Developer Certificate) [y/N]n
[2021-11-20 15:44:56] [INFO] Skipped signing process.
[2021-11-20 15:44:56] [INFO] Application installer generation steps finished.
[2021-11-20 15:44:56] [INFO] Installer generating process finished
(base) Thomass-MacBook-Pro-3:macOS-x64 thomashansknecht$
```

After the build is complete the developer can find the resulting .pkg file at macOs-x64/target/pkg as well as the template .pkg file at macOs-x64/target/package

2.2.2.2 Signing the Installer

During the build process at step 2 the developer will be asked if they want to sign the installer. The question will appear similar to the following image:

```
macOS-x64 bash build-macos-x64.sh ChatVUE 0.0.1 -- 123x23
~/Documents/macos-installer-builder/macOS-x64 -- bash build-macos-x64.sh ChatVUE 0.0.1
(base) Thomass-MacBook-Pro-3:macOS-x64 thomashansknecht$ bash build-macos-x64.sh ChatVUE 0.0.1
[ 0%] [ 25%] [ 50%] [ 75%] [ 100%]
[ 0%] [ 25%] [ 50%] [ 75%] [ 100%]
[ 0%] [ 25%] [ 50%] [ 75%] [ 100%]
Application Name : ChatVUE
Application Version : 0.0.1
[2021-11-20 15:44:56] [WARN] Apache Maven was not found. Please install Maven first.
[2021-11-20 15:44:56] [WARN] Ballerina was not found. Please install ballerina first.
[2021-11-20 15:44:56] [INFO] Installer generating process started.
[2021-11-20 15:44:56] [INFO] Cleaning /Users/thomashansknecht/Documents/macos-installer-builder/macOS-x64/target directory.
[2021-11-20 15:44:56] [INFO] Application installer generation process started.(3 Steps)
[2021-11-20 15:44:56] [INFO] Application installer package building started.(1/3)
[2021-11-20 15:44:56] [INFO] Application installer product building started.(2/3)
Do you wish to sign the installer (You should have Apple Developer Certificate) [y/N]? 
```

The developer can answer the question by (pressing the n key for no or the y key for yes in the terminal/bash window followed by the enter key) More documentation regarding why and what signing the installer will do can be found on Kosala Sananthana's blog which can be found here:

<https://medium.com/swlh/the-easiest-way-to-build-macos-installer-for-your-application-34a11dd08744>

The main purpose of signing the installer will be to make the installer appear as trusted on the user's computer. However not signing the installer appears fine and at worst will give more access permission warnings when the user installs ChatVUE to their system. If the developer wants to sign the .pkg file they will need an Apple Developer Certificate.

2.2.2.3 The build command

Let's explain exactly what the build command is and what it does. The exact build command is located in the commandToBuildHere file which can be found at ChatVUE-Client/Installer Application/Mac Installer/macos-installer-builder/.

The actual command is:

```
$ bash build-macos-x64.sh ChatVUE 0.0.1
```

The **bash** specifies that we want to run the command under the bash language.

The **build-macos-x64.sh** is the shell script we use to build the installer .pkg file.

The **ChatVUE** specifies the name of the installer and the output name it will be saved as.

The **0.0.1** specifies the version of the install.

Note: ChatVUE as the directory name and version 0.0.1 as the version number are currently expected during scripting. If a developer wants to change these values then searching for anytime 0.0.1 as well as ChatVUE is used in all of the .app scripting files will be necessary as these values need to be changed to allow for a different directory structure. Currently there are global variables that specify the name and version number in the postinstall script which is found at

macos-installer-builder/macOS-x64/darwin/scripts but these global variables are not specified in any of the .app files located in the macos-installer-builder/macOS-x64/application folder. So all .app files at this location will need to be updated with the appropriate directory values as needed.

2.2.2.4 The directory structure

Let's discuss the directory structure of the macos-installer-builder. First navigate to the macos-installer-builder folder. Next we want to make sure all of our files are present. Let's examine the directory structure here.

Name	Date Modified	Size	Kind
LICENSE	Oct 22, 2021 at 6:39 PM	11 KB	Unix Executable File
README.md	Oct 22, 2021 at 6:39 PM	690 bytes	Markdown Document
macOS-x64	Today at 1:30 PM	--	Folder
build-macos-x64.sh	Oct 23, 2021 at 1:06 AM	6 KB	Shell Script
commandToBuildHere	Today at 1:28 PM	38 bytes	Unix Executable File
application	Nov 18, 2021 at 10:39 PM	--	Folder
darwin	Oct 28, 2021 at 9:26 PM	--	Folder
target	Today at 1:30 PM	--	Folder
darwin	Today at 1:30 PM	--	Folder
darwinpkg	Today at 1:30 PM	--	Folder
package	Today at 1:30 PM	--	Folder
pkg	Today at 1:30 PM	--	Folder
utils	Oct 22, 2021 at 6:39 PM	--	Folder

LICENSE - contains the Apache License

README.md - contains more info regarding the template and how it works. Currently this README.md is the readme that Kosala originally created and simply contains references that were used and a simple definition of the git-hub project he created as well as encouraging giving suggestions and feedback on improving the installer.

macOS-x64 - This folder is the most important as it contains all of the needed files during and after the installation process

2.2.2.4.1 The build files

macOs-x64/build-macos-x64.sh - This is the shell script that actually creates the installer. Not much has been changed in this script when

compared with the original. However, making changes to this script will fundamentally change how the installer is created and what features and interface windows are added. This script also specifies the install directory. Currently the directory is set to the Library folder under the /Library folder which allows the Installer to install for all users of the system.

macOs-x64/commandToBuildHere - This file simply contains the bash ... command mentioned earlier to build and construct the installer. To use the file one can simply cat into it from the terminal using \$cat commandToBuildHere or open the file in one's favorite text editor. The file is exclusively used to help the developer remember what the actual build command is so that the developer does not have to look up the command all the time.

2.2.2.4.2 The source files

macOs-x64/application - This folder contains all of the .app files as well as the project solution files “ie. the source code” which is placed inside of the ChatVUE-Client folder and this folder contains all the needed source code to build using .NET which is placed within the application folder. Let's take a closer look at what this directory should have inside.

Name	Date Modified	Size	Kind
macOS-x64	Today at 1:30 PM	--	Folder
build-macos-x64.sh	Oct 23, 2021 at 1:06 AM	6 KB	Shell Script
commandToBuildHere	Today at 1:28 PM	38 bytes	Unix Executable File
application	Nov 18, 2021 at 10:39 PM	--	Folder
ChatClientUI.app	Oct 28, 2021 at 11:46 PM	7.3 MB	Application
DotnetInstall.app	Nov 18, 2021 at 10:23 PM	3.4 MB	Application
DependencyInstaller.app	Oct 23, 2021 at 4:42 PM	3.4 MB	Application
ChatVUE Uninstaller.app	Yesterday at 4:25 PM	3.3 MB	Application
DotNetBuild.sh	Oct 23, 2021 at 3:58 AM	869 bytes	Shell Script
.gitkeep	Oct 22, 2021 at 6:39 PM	2 bytes	Unix Executable File
ChatVUE-Client	Nov 18, 2021 at 10:39 PM	--	Folder

macOs-x64/darwin - This folder contains all of the necessary parts that create the user interface as well as provides the process used after the installation. This folder also decides on dependency requirements as well as an after install confirmation. Lets dig deeper into these files and folders and what they do/are.

Name	Date Modified	Size	Kind
darwin	Oct 28, 2021 at 9:26 PM	--	Folder
Distribution	Oct 23, 2021 at 1:05 AM	2 KB	Unix Executable File
Resources	Oct 28, 2021 at 9:26 PM	--	Folder
TunnelVUE.icns	Oct 28, 2021 at 9:25 PM	728 KB	Apple icon image
banner.png	Oct 28, 2021 at 8:41 PM	143 KB	PNG image
LICENSE.txt	Oct 23, 2021 at 4:17 PM	11 KB	Plain Text Document
conclusion.html	Oct 23, 2021 at 5:54 PM	3 KB	HTML text
uninstall.sh	Oct 23, 2021 at 12:32 AM	2 KB	Shell Script
welcome.html	Oct 23, 2021 at 4:47 PM	1 KB	HTML text
welcome.css	Oct 23, 2021 at 3:10 PM	133 bytes	CSS
TunnelVUE.iconset	Oct 28, 2021 at 9:26 PM	--	Folder
scripts	Oct 23, 2021 at 7:48 PM	--	Folder
postinstall	Oct 23, 2021 at 7:48 PM	870 bytes	Unix Executable File

macOs-x64/darwin/Distribution - This file sets install requirements such as the required MacOS version as well as handling the installation verification. This verification validates that files are placed in the appropriate place and that all conditions were met during the install process. If the developer wants to change the install location this file will need to be modified otherwise the Distribution will search for files in the Library folder and since it does not find them it assumes the install failed and notifies the user.

In addition the Distribution file specifies the layout of the installer. To add or remove interface elements to the installer this file needs to be modified. The original file was created by Kosala

macOs-x64/darwin/Resources - This folder contains all graphical elements present during and after the install. This folder also contains the uninstall.sh script which was created by Kosala and has not been implemented currently but provides a template for creating a shell based uninstaller which could be looked into with future versions of the installer. The TunnelVUE.iconset is what creates the application icon for the ChatClientUI.app. To create and use iconset files refer to this blog created by OWC Chris S:

<https://eshop.macsales.com/blog/28492-create-your-own-custom-icons-in-10-7-5-or-later/>

The .html files located at macOs-x64/darwin/Resources are the visual pages that display during the install. As an example the welcome.html file is what is displayed as the first page of the installer while the conclusion.html is displayed as the final page of the installer and only displays after the postinstall is marked as completed and after the Distribution determines that all files and folders were correctly installed. The html files here seem to only be grayscale models as adding colors and css properties to these files does not currently work.

macOs-x64/darwin/Resources/LICENSE.txt is the actual license file that is required to be read during the installation. To change or update the license modify this file but keep the same name as LICENSE.txt

2.2.2.4.3 The post install sequence

macOs-x64/darwin/scripts - This folder contains the post install instructions. Currently the only file present in the post install sequence is the postinstall file. The postinstall file specifies the commands to run during the scripting portion of the install. Currently the postinstall calls the .app files to handle installing dependencies in addition to the scripting for building and publishing .net solutions. Currently there is a bug with the postinstall which is that the check for dotnet version does not work as shell does not recognize the location for dotnet and thus the dependencyinstaller.app will always be run during the post install

sequence. Another issue is that it was unknown how to make the post install wait for the termination of the DotnetInstall.app thus the postinstall simply waits a few minutes before concluding and saying the install was successful. **It is recommended** that future developers improve this postinstall process.

macOs-x64/target - This folder is essentially temporary as it is changed every time the installer is built. Also, the folder can simply be empty before the build process using the \$bash... command discussed earlier and the build process will remain identical either way. One way to think of the target versus darwin is that the darwin is what is used to create the installer and the target is the result after building the installer. Thus the two directories, darwin and target, will share a similar structure as well as identical files.

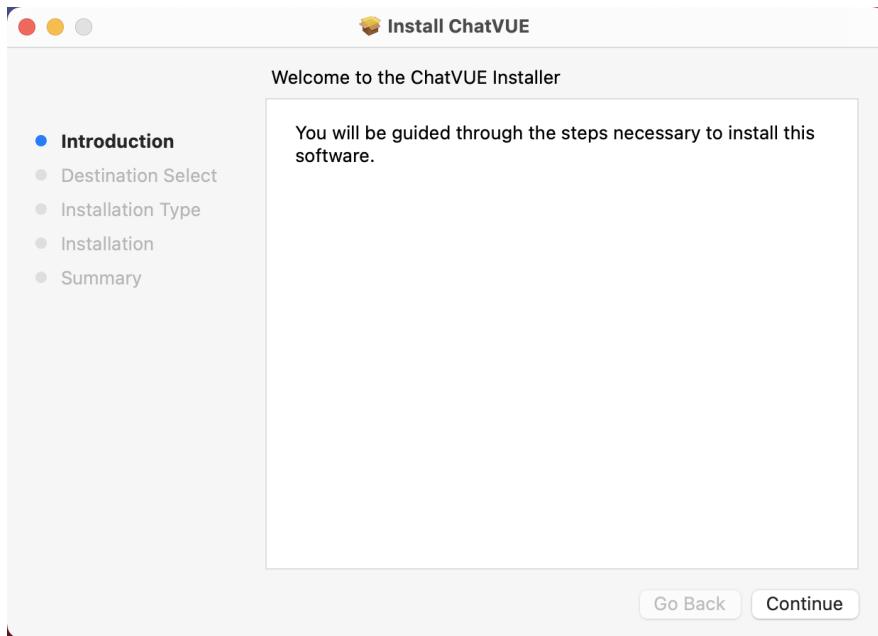
****Be careful** - It is easy to accidentally develop in the target folder as it contains a similar directory structure and identical files as the darwin folder after the installer is constructed. **However, any changes and modifications to this folder will be completely erased after a build.** So it is best not to touch this folder unless one is trying to retrieve the .pkg files.

macOS-x64		Today at 1:30 PM	--	Folder
build-macos-x64.sh		Oct 23, 2021 at 1:06 AM	6 KB	Shell Script
commandToBuildHere		Today at 1:28 PM	38 bytes	Unix Executable File
application		Nov 18, 2021 at 10:39 PM	--	Folder
darwin		Oct 28, 2021 at 9:26 PM	--	Folder
target		Today at 1:30 PM	--	Folder
darwin		Today at 1:30 PM	--	Folder
darwinpkg		Today at 1:30 PM	--	Folder
package		Today at 1:30 PM	--	Folder
ChatVUE.pkg		Today at 1:29 PM	20.4 MB	Installer package
pkg		Today at 1:30 PM	--	Folder
ChatVUE-m...x64-0.0.1.pkg		Today at 1:29 PM	20.7 MB	Installer package

macOs-x64/target/darwin - This folder contains the same files present in the darwin under macOs-x64/darwin. However, the files here are created/modified during the build process.

macOs-x64/target/darwinpkg - This folder contains the same source files that are present in the application folder found at macOs-x64/application. However, the files here are created/modified during the build process and the directory structure here will match where the directories are placed on the target's (the person that uses the installer) machine.

macOs-x64/target/package - This folder will contain the unmodified version of the installer. In this version no custom elements are displayed and the installer is in the most basic form with the original instructions (It is believed that Apple uses these instructions as the template) However, there has been no testing done with this .pkg file and it is unknown as to exactly why it exists or what it does. This .pkg file is automatically generated after building the installer with the \$bash... command mentioned earlier. The macOs-x64/target/package installer will appear similar to:



macOs-x64/target/pkg - This folder will contain the modified version of the installer. In this version all the custom elements are displayed and the installer behaves as intended. It is recommended to use the installer that is created here and to simply discard the installer that is created in the macOs-x64/target/package directory. The .pkg file here is automatically generated after building the installer with the \$bash... command mentioned earlier. The macOs-x64/target/pkg installer will appear similar to:



2.3 Linux Installer

With the Linux OS there are multiple installers constructed for various Linux distributions. Currently support is being made for Debian and RedHat Linux. Neither package builder is working properly yet but the idea is to have the installation completed through a .deb package for Debian systems and a .rpm package for RedHat systems. Instructions should be added in the package builder in case the user has issues or wants to build the package manually. The package can be built manually by running the setup file in shell. Both automatic and manual installs will rely on shell scripting. Most Linux distributions come with shell functionality so relying on shell functionality should not be an issue for the user but in case it would be a

good idea to provide tools in the package builder to assist the user with adding shell functionality to their Linux system.

2.3.1 Red Hat Installer

The Red Hat Installer files are located at
 ChatVUE-Client/InstallerApplication/Linux Installer/RedHat Install/

Currently the Linux Installer assumes the install location to be in the `~/$USER/Downloads` directory which simply means the given user's downloads directory and using `$USER` in the bash script makes the address relative to the user that is using the system. For the current installation process the installer copies the ChatVUE-Client folder in the `~/$USER/Documents` location. If the folder ChatVUE-Client is not found in this directory or the directory itself is not found the installer will terminate and display helpful hints to the user as to what went wrong during the install process.

2.3.1.1 Configuring the RPM Build

For rpm packages the directory is as follows:

```

File Edit View Search Terminal Help
rpmbuild
  └── BUILD
      ├── DependencyInstallScript
      ├── dotnet-install-Script.sh
      ├── DotnetPublishScript
      ├── setup
      └── UninstallerScript
  └── RPMS
  └── SOURCES
      ├── Chat Client App.desktop
      └── ChatVUE-Client.tar.gz
          └── Images
              ├── ChatVueBannerIcon.jpg
              ├── ChatVueBanner.jpg
              ├── ChatVueBanner.png
              ├── ChatVUEIcon.ico
              └── ChatVUEIcon.jpg
  └── SPECS
      └── Chatclient.spec
  └── SRPMS
  6 directories, 14 files

```

The BUILD folder:

This folder includes all of the scripts that can be run during the build process.

setup (Shell Script)

This script will automatically launch the dependency installer, dotnet-install-Script.sh, and DotnetPublishScript script. This script also requires the user to accept the license which is presented to the user during the install process.

DependencyInstallScript (Shell Script)

This script first asks the user if they have .NET 5.0 sdk installed. When the answer is yes this script will run dotnet-install-Script.sh. After the .NET framework is created the DependencyInstallScript will locate the appropriate .NET 5.0 sdk package from the Default system Repository to install. The .NET 5.0 package selects the most recent version from Microsoft Servers. After the runtime environment and .NET 5.0 sdk are installed the DependencyInstallScript concludes.

dotnet-install-Script.sh (Shell Script)

This script was created by Microsoft to be an easy way of finding and installing the necessary dependencies to install the dotnet framework.

DotnetPublishScript (Shell Script)

This script navigates to the specified ChatVUE files location and builds the necessary .NET projects including (ChatClient, ChatServer, ChatUI, and KeyServer) The building is done using the command:

```
$ dotnet publish -r linux-x64 --self-contained false
```

UninstallScript (Shell Script)

It is unlikely the user will want to build the UninstallerScript to their system as this will remove everything that was built previously but the Uninstaller will be run if the install fails due to corruption. When corruption happens the best strategy is to remove all files so the user has a fresh slate for running the install again. If no fatal errors occur, the Uninstall script will not be run during the build process. The main purpose of the UninstallScript is to assist the user in removing ChatVUE from their system in the event the user uses a custom install location or for whatever reason the default package handler on Linux is unable to remove ChatVUE from the user's system.

The SOURCES folder:

This directory includes all of the source code and images that are used during the install for either backgrounds or icon files.

The SPECS folder:

This directory contains user help documentation as well as the binary execution instructions for the Linux system to follow when running the rpm package. The binary execution instructions are in ChatClient.spec while the user help documentation is in Instructions.txt

Sample Outputs from the shell console of running setup file

```
File Edit View Search Terminal Help
.....(0% Progress) [Intializing setup]
-----
Install the required .NET 5.0 sdk for ChatVUE?
-----
Continue? y or n? █
```

```
File Edit View Search Terminal Help
#####.....(10% Progress) [Installing .NET 5. ]
Last metadata expiration check: 2:39:47 ago on Wed 20 Oct 2021 01:06:59 PM CDT.
Package dotnet-sdk-5.0-5.0.207-1.el8_4.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
```

Error Handling of Linux Installation

If the user messes up the install by not following directions, such as not specifying the correct location of the ChatVUE directory on their system then a custom error message is generated informing the user of their mistake as well as halting the install process so no files get corrupted or placed in the wrong locations.

Sample error Output from the shell console of running setup file

```

File Edit View Search Terminal Help
#####
(100% Progress) [Install Terminated]
FATAL ERROR: ChatVUE files were not found on your system
TIP 1: You need to select an install location that the user: thansknecht has permissions to access

TIP 2: If you are not user: thansknecht please switch to another user with the appropriate permissions to install software to this system

TIP 3: The install location you picked may not be possible unless you are the root . Don't run the setup as root. Instead specify a location non-root users can access

TIP 4: Installing as root is risky and any-non root user on your system trying to access the ChatVUE service may receive permission denied and be unable to access the service.

TIP 5: You can try the install again by re-running the setup file found in your distribution install location
[thansknecht@localhost RedHat Install]$ █

```

2.3.2 Debian Installer

The Debian Installer files are located at
 ChatVUE-Client/InstallerApplication/Linux Installer/Debian Install/

The debian installer is similar to the Red Hat installer in that it uses the same files excluding the RPM build structure. However, the RPM Build files are also in the Debian installer. Debian systems should use a .deb format which is not currently implemented at this time. The main difference between the Debian and Red Hat install is that the two different distributions use different package managers which causes the syntax to be slightly different in the scripting files depending on whether we are using snap packages and apt for debian systems or we are using yum and dnf for Red Hat systems. The Debian installer was tested using Ubuntu 18.04. Currently the Debian installer has an issue in that installing the .net 5.0 using a snap package causes only the root to have access to .net 5.0

commands. Thus currently with the debian install root permissions are required to both complete the install as well as open the application created after the install process is complete.

One solution here is to update the project to support .net 6.0 as this newer version does not require root permissions to use after being downloaded as a snap package. The main issue though is that .net 6.0 is currently not supported with the ChatVUE project solution. In addition, changing the version of .net would mean **needing to update the .net dependency values on all platforms** including Windows, Mac OS, as well as Linux Installer applications

2.3.3 Customizing Install Location

The install location is not a global variable currently thus the files setup, DependencyInstallScript, DotnetPublishScript, UninstallerScript, and ChatUIShortcut.sh will all need to be modified with the appropriate install and retrieval locations. Most of these files can be found in both the:
ChatVUE-Client/InstallerApplication/Linux Installer/RedHat Install as well as
ChatVUE-Client/InstallerApplication/Linux Installer/Debian Install
While the remaining files are found in
ChatVUE-Client/InstallerApplication/Linux Installer

Currently the installer (which is really just a bunch of scripts) expects the install files to be located in the ~/Documents folder. If the folder ChatVUE-Client is detected in this directory the install will continue otherwise the install will terminate and explain to the user what went wrong during the install. Currently the install directory is at ~/Downloads. Which basically means in the current state the installer copies the ChatVUE-Client folder from the documents folder and places it in the downloads folder. After this process completes the installer builds the solution files and finally creates desktop and launchpad shortcuts which is done in the script DotnetPublishScript.

It is recommended to change these install and retrieval locations because installing from documents folder to downloads folder is not ideal.

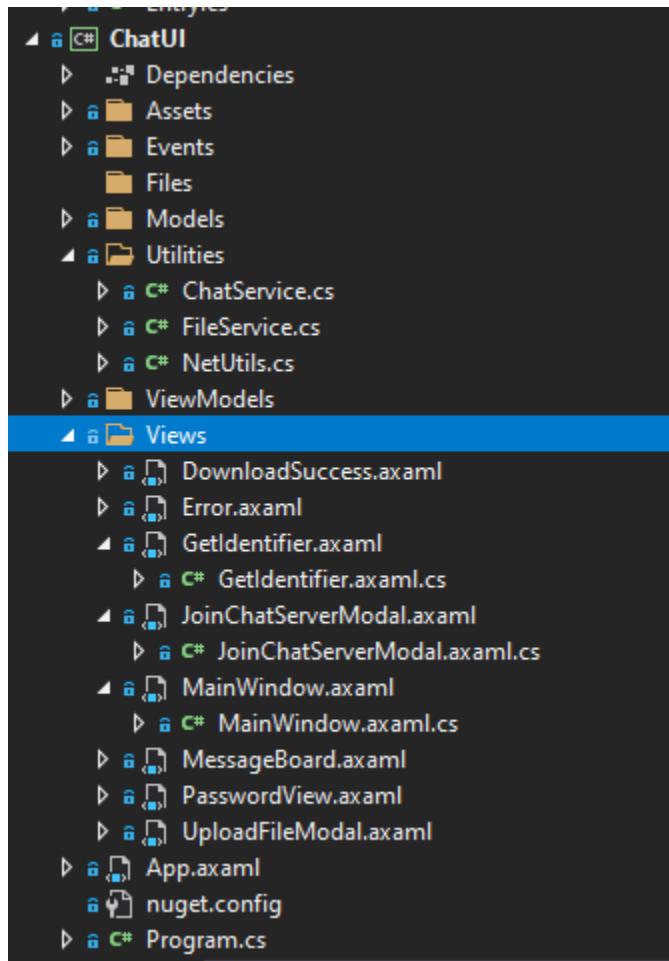
DotnetPublish creates the shortcut files at `~/.local/share/applications` as well as `~/Desktop` and these shortcut files simply refer to the `ChatVUE-Client/InstallerApplication/'Linux Installer'/ChatUIShortcut.sh` shell script which is a script that executes the dll files needed to load the Chat UI which is done using the dotnet 5 sdk. These dll files are generated earlier in the build but while the DotnetPublishScript is running.

3 Developing for Chat UI

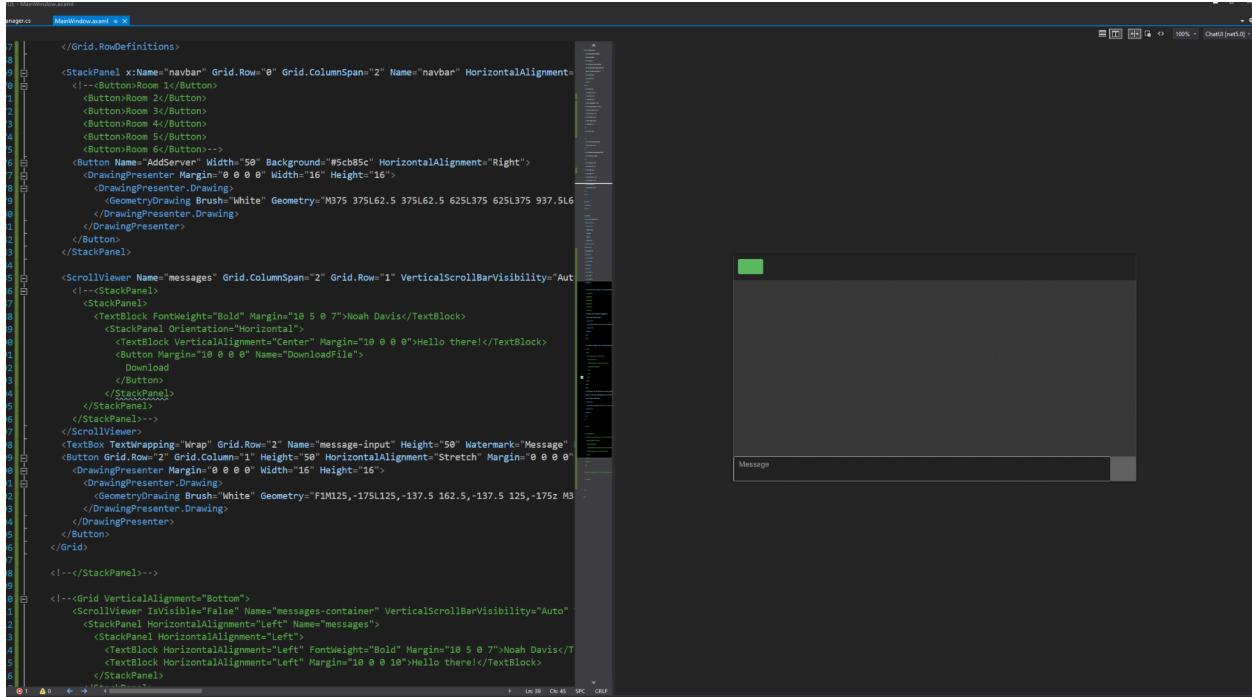
3.1 Setting up the keyfiles

One problem we had this semester was integrating optimizer code that Mr. Edwards provided us in order for the KeyServer to quickly search for keys. To generate license keys to be used by a client with the new optimizer code, you need to run the `ServerSetupConfiguration` method in `COREKeyManager.cs` within the `CORECryptography` project then place a key in the ChatServer's or ChatUI's `bin/Debug/net5.0` folder. You'll also need to change the code to accommodate the new filename as the optimizer code uses different naming conventions than the ones that the methods that read the keyfiles are searching for (in `FileService.cs`). So either change the method in `COREKeyManager.cs` to match the naming conventions of those methods or vice-versa. Once that's been done, you should be able to decrypt and brand keys once the correct passwords have been given in the corresponding GUI. Additionally, in order to communicate with a server from another network, the one running a ChatServer will probably need to set up port forwarding on their network.

3.2 ChatUI Project



The new GUI is located in the ChatUI project of the avaloniaUI branch in Github and the back-end is heavily based off of the ChatClient project that preceded our group's effort. The ChatUI project is written with the [AvaloniaUI](#) MVVM framework, but we don't really make much use of the viewmodels in the project and do most of the data binding in each view's corresponding c# file. The Main Window and its corresponding c# file contain the most integral code to the application. It contains a list of ChatServices (which are used to communicate with ChatServers) and a list of StackPanels to keep track of the messages that have been sent in each ChatServer that has been connected to.



In Visual Studio, there's a convenient designer window that can be used to design windows without always having to check how they look by debugging.

```
1 reference | nmdavis24, 7 days ago | 1 author, 5 changes
public MainWindow()
{
    Opened += onOpened;
    WindowState = WindowState.Maximized;

    InitializeComponent();

    this.AttachDevTools();

    var textbox = this.Find<TextBox>("message-input");
    var addBtn = this.Find<Button>("AddServer");
    var fileBtn = this.Find<Button>("addFiles");

    textbox.KeyDown += SendMessage;
    addBtn.Click += OnAddBtnClicked;
    fileBtn.Click += AddFileDialog;
    //this.WhenActivated(d => d(ViewModel!.ShowDialog.Reg
}
```

```

/// <summary>
/// Keydown event listener on textbox. Sends the message when
/// the user hits enter and clears the textbox
/// </summary>
/// <param name="sender"></param>
/// <param name="e">Event info i.e. which key was pressed</param>
1 reference | nmdavis24, 11 hours ago | 1 author, 6 changes
private void SendMessage(object? sender, KeyEventArgs e)
{
    // probably inefficient but checks if enter button was pressed
    // on message input textbox
    if (e.Key == Key.Return)
    {

```

In the constructor of a window class (or any other method), you can search elements of the page by name and initialize the event listeners on the page that handle actions the way you need them to. There are other ways to add listeners (like through properties in the .axaml files or using reactive elements in a.viewmodel) but I've had the most success doing them this way.

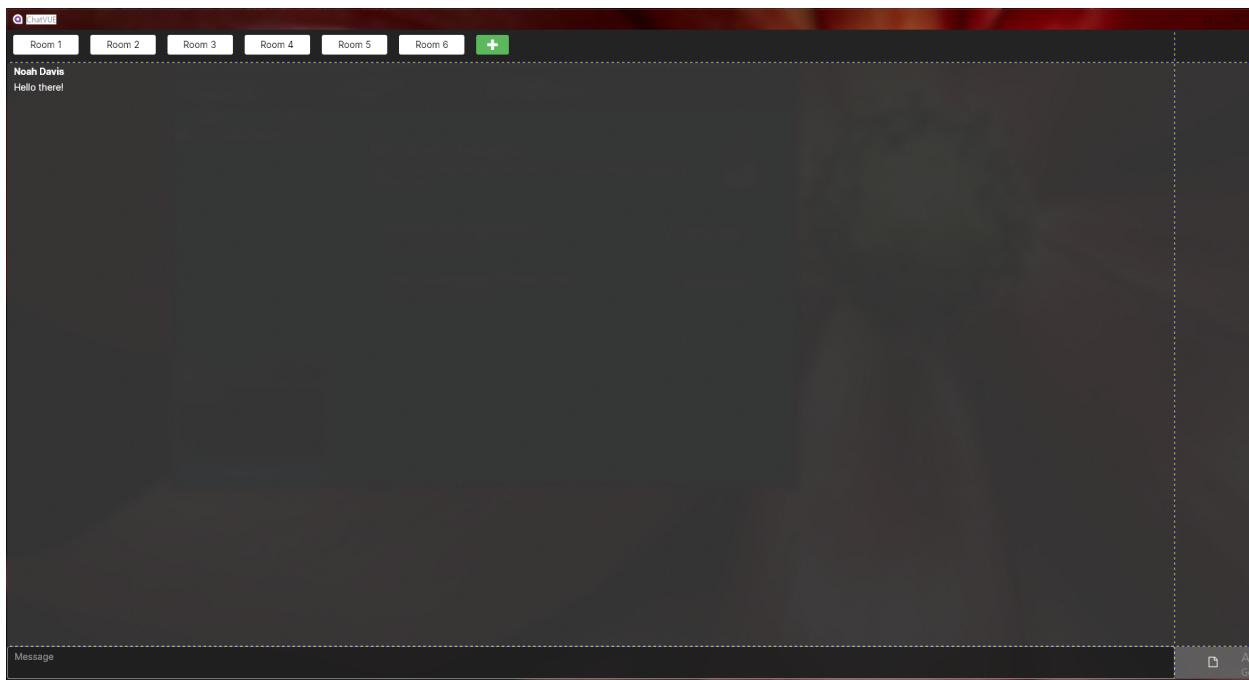
Next I'll list each major component of the ChatUI project and give a brief description of its purpose (though the description for the MainWindow may not be so brief):

- **The Events Folder:** Contains 5 classes representing Event Arguments that will be sent to the Client from the server: ConnectionFailedEventArgs.cs, ConnectionSuccessEventArgs.cs, DisconnectionEventArgs.cs, KeyReceivedEventArgs.cs, and MessageReceivedEventArgs.cs. Each of these files are used to create event handlers so that the GUI can react to messages sent from a Server.
- **Models.KeyFileModel:** A class representing a KeyFile that allows us to easily manipulate KeyFile data. Used to check if KeyFiles are branded and to store the machine's encrypted key as a byte array to be used in various methods.
- **ChatService.cs:** This class contains all of the methods necessary for a ChatClient to function. It contains the EventHandlers for the 5 events listed above as well as methods to validate user inputs when they try to join a chat server, a method to create a TCP connection with a KeyServer, and a method that sends users' messages to a ChatServer. It also contains an identifier so that we can switch

between ChatServices as needed.

- **FileService.cs:** Contains two methods: ReadKeyFile and SaveKeyFile. They're used to read and update the KeyFile stored on the user's machine.
- **NetUtils.cs:** Contains ResolveAddress method which helps us convert string IP Address inputs into IPAddress objects so that we can use the addresses to make TCP connections.
- **Views/ViewModels:** We currently have 8 Views in use to allow the GUI to function but will eventually need to add more for the Admin GUI and other functions. I'll list the views below and explain what they do.

MainWindow



The main window is a pretty basic looking page that uses grids to segment the page into 3 main sections (I added the dotted grid lines to illustrate): the navbar at the top, the chat room in the middle, and the message input bar at the bottom.

The chat room is a ScrollViewer that contains a StackPanel that holds the messages that are sent. Messages are children of the main StackPanel and are also StackPanels themselves that contain TextBlocks. This allows us to easily append messages to the chat room programmatically and maintain separation between each message. With each ChatServer connection, a StackPanel containing the messages sent in the server since

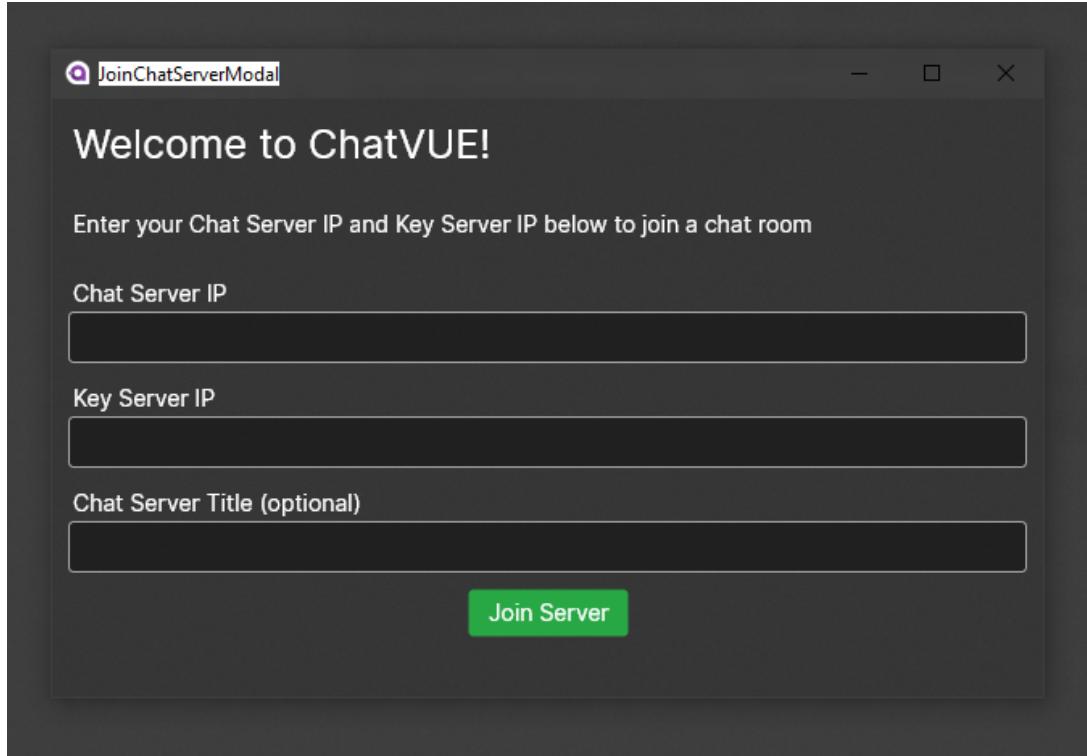
the user has connected is stored in the MainWindow's c# file and is updated even when the server isn't being displayed.

The navbar contains buttons associated with a server that can be clicked to switch between servers. The switch is done by searching the list of StackPanels for one with the identifier that is passed to the code by the button and replacing the content of the Scrollviewer with it. The green plus button pulls up a dialog window that allows users to create a connection with another server. Once this connection is created, another button is added to the navbar for the server. The styling on this page is pretty crude and will need to be improved but right now we're mostly concerned with getting this thing to function correctly. There will also need to be additional visual components added for Server Admin functions and audio/video chat once those are implemented.

The message input bar has a button on the right to handle file inputs and messages can be sent by typing something in and clicking enter.

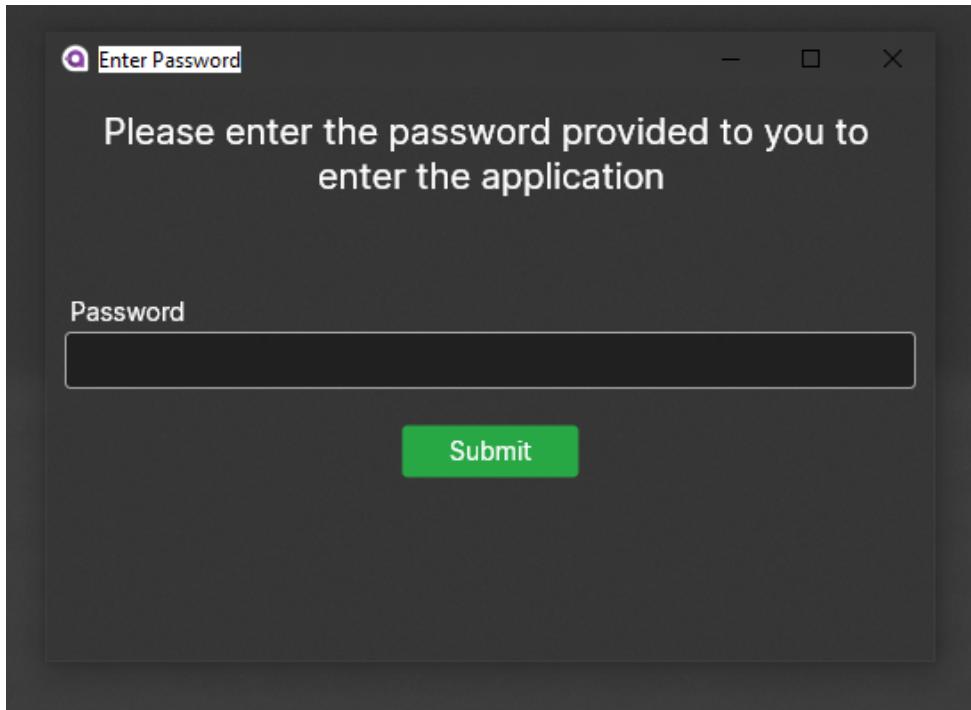
The c# file associated with this view contains all the event listeners and other methods necessary for the above functions to work properly. It also has listeners for messages from the ChatServer so that we can update the chat logs of all the servers the user is connected to. In the future we'll need to give users the ability to disconnect from a server if they please, but we have not implemented this function yet (though they can disconnect from every server if they exit the application!).

JoinChatServerModal



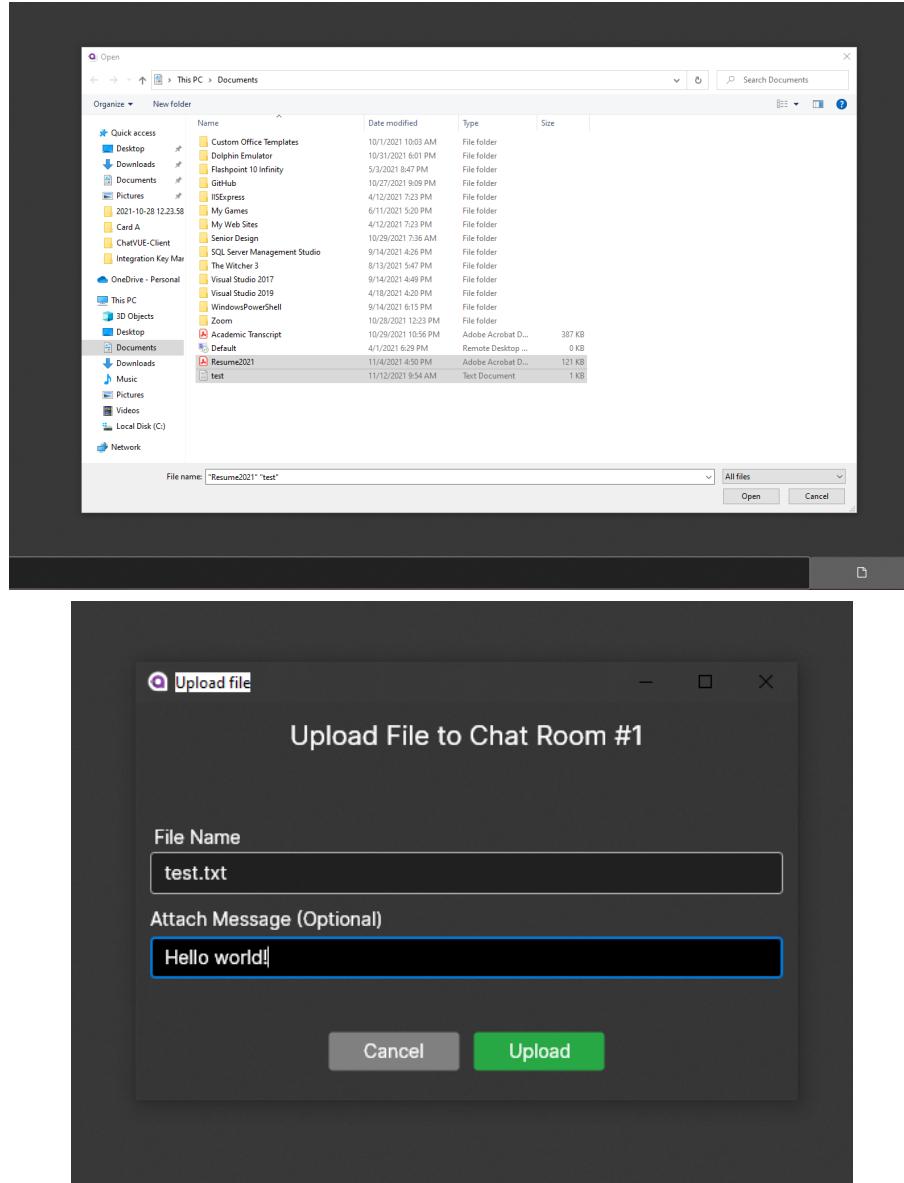
This dialog window appears on startup of the application and when the user clicks the add button on the navbar. It takes user input for the IPs needed to connect to each server and an optional input of a title for the server. If given, the title appears in the button on the navbar that is added once the connection is made; otherwise, the button's title defaults to the IP of the ChatServer. The c# file for this dialog window validates the inputs with a static method in the ChatService and notifies the user of any errors. If it's the first time the user's machine has attempted a connection to this key server (AKA their keyfile hasn't been branded), then another dialog is opened to receive the password needed to decrypt the key from the user. After this or if the file has already been branded, a method from the MainWindow is called to add the new connection and the dialog closes.

PasswordView

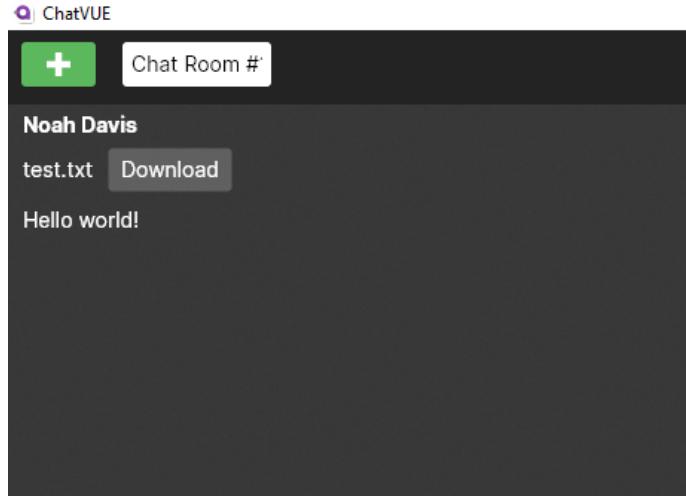


PasswordView is another dialog window that shows when a user is connecting to a KeyServer for the first time. It takes the user's password input and uses it to decrypt their keyfile and make a connection to the KeyServer. Once a connection is established, an event listener in the MainWindow updates the key with the new one sent by the KeyServer.

UploadFileModal



The UploadFileModal pops up after a user has selected files from the file explorer after clicking on the file icon button in the bottom right. The modal has hidden input fields that store the path(s) of the selected files along with a Textbox to add an additional optional message with the file.



Once the user has sent the file, this is what appears in the chat room. The user can click on the Download button to save the file to their downloads folder.

Work to be done on GUI

- Currently, the backend is in a bit of a state of disarray so the avaloniaUI branch has a ChatUI that is basically a skeleton front-end of the features needed for the application to function correctly. There is a lot of commented code that would serve to connect it to the backend and the server with some modification once the optimizer code has been properly implemented. For example, in the MainWindow.cs file, the AddConnection method only contains a parameter for the title of the chatroom being added but has three other params commented to the side and event listeners for the ChatService commented within it. For this to work with the server, you'll need to uncomment those lines and move the code below it into the OnConnectionSuccess method so that a room isn't added to the GUI and list of ChatServices until a connection has been successfully established with the server.
- Another thing that needs to be finished is the file upload/download functionality. Right now, when a file is uploaded it is not being encrypted and sent to the server and is rather being stored in the bin of the ChatUI project and being copied to the downloads folder of the

user (and this may only work on Windows). This was to serve as a model for how it should actually work so I could model the front-end since I couldn't get the back-end to cooperate. What you need to do is instead of sending the file to the bin on upload of the file, you need to encrypt the file with a single key and send it to the server to be stored in the Server's database or in a folder (probably the db). Then, when a ChatClient user clicks on the download button for the file, a request needs to be sent to the server to decrypt and send the file back to the client.

- We still don't have identifiers for messages implemented, so you can't tell who is sending what. I wrote a basic window to take in user input for an identifier called GetIdentifier.axaml and intended to use it to get a user to enter their identifier the first time they open the application but didn't get around to it. Once you take the input, you will need to store it somewhere (either on the client's config file or in the KeyServer db with its key). Then it can be used to identify who is sending a message.
- Some work should be done to make the GUI prettier, to make it more responsive to varying input/window sizes, and to add ChatVUE branding similar to the way Thomas has done with the installers.
- We discussed adding audio/video chat this semester, but obviously did not have the time to get to that.

3.3 Admin controls from UI



The AdminWindow currently contains these 3 functions. This window should be implemented in the same way as the chat server windows, where you can tab in and out of the admin window into another admin window or chat window.

4. Key Server Admin

The Key Server directory holds the back end methods for performing admin functions, as well as handling messages from the client. The server handles admin message requests from the Server in KeyServerService.cs where messages received from the client are handled. The client side, which sends the admin messages is present in ChatUI/Utilities/ChatService.cs. Any method that interacts with the Key Server database is in KeyServerSQL.cs. The GUI for interfacing with the Key Server is present in ChatUI/Views/AdminWindow.xaml. The GUI needs to be integrated into the MainWindow as well as connected to the chatService so it can send adminRequest messages to the Key Server.

5. Contributions

Thomas Hansknecht - (Header Page, Table Of Contents, Chapters 1.3-1.4, 2, 5, 6)

Jiaqi Liu - (Chapter 1)

Noah Davis - (Chapter 3.1-3.2)

Scott Conwell - (Chapters 3.3, 4)

6. Extra Resources

[How to make a .exe file execute after installation of project using setup and deployment project in c#](#)

[Connect to Server \(Database Engine\) - SQL Server Management Studio \(SSMS\) - Microsoft Docs](#)

[Docker- Install containers for SQL Server on Linux - SQL Server - Microsoft Docs](#)

[The Easiest Way to Build macOS Installer for Your Application - by Kosala Sananithana - The Startup - Medium](#)

[bash - How to change the output color of echo in Linux - Stack Overflow](#)
[How to change the output color of echo in Linux](#)

[How to actually run dotnet project on Mac](#)

[Create Your Own Custom Icons in OS X 10.7.5 or Later](#)

[applications - Open terminal via AppleScript - Ask Different](#)

[Building binary deb packages- a practical guide - Internal Pointers](#)

[Documentation To Install WSL \(Windows Subsystem for Linux\)](#)

[Application publishing - .NET - Microsoft Docs](#)

[KosalaHerath-macos-installer-builder- Generate macOS installers for your applications and products from one command](#)

[How to create a Linux RPM package - Enable Sysadmin](#)

[How to create a new database in Microsoft SQL Server](#)

[Open Port Check Tool](#)

[ChatVUE Linux Install Demo](#)