ZOOM AUTOJOINER GUI

Computer Science Project

AUGUST 31, 2021

ADVAITH MENON, DAKSH BAGRECHA, NISHANT BHANDARI, PRATHAM JAIN
@11c-csproject

Contents

Introduction

TL;DR A tool to join Zoom meetings quickly. Comes with a GUI. Prior configuration needed.

Goals

- To serve as a meeting notice board (see Todo)
- To remind that it is time to join a meeting
- To aid in typing long meeting IDs and passcodes

Use Scenario

You are working on an assignment, and you are so focused that you forget that your meeting is now. When you realise that you had the meeting, you open WhatsApp Web to get the passcode and ID. It takes 10 minutes to load, only to show you that it can't connect to the phone. Now you scramble about for your cell phone. You are furious as Fingerprint Unlock fails. Finally, you copy the Meeting ID to the Zoom interface, enter the passcode wrongly once, and join the meeting. Once you join, you realise that you have joined in your mother's account.

Now you found this. You just add your meeting in the intuitive interface and BAM! When it is time, you will find your cursor automatically moving, and you will now remember about the meeting, and you can get ready while we do the clicking for you!

What **not** to use this program for

- Overloading Zoom Servers with Join Meeting requests
- Attending your classes/meetings on your behalf
- Other uses that do not fit in to the goals of this program

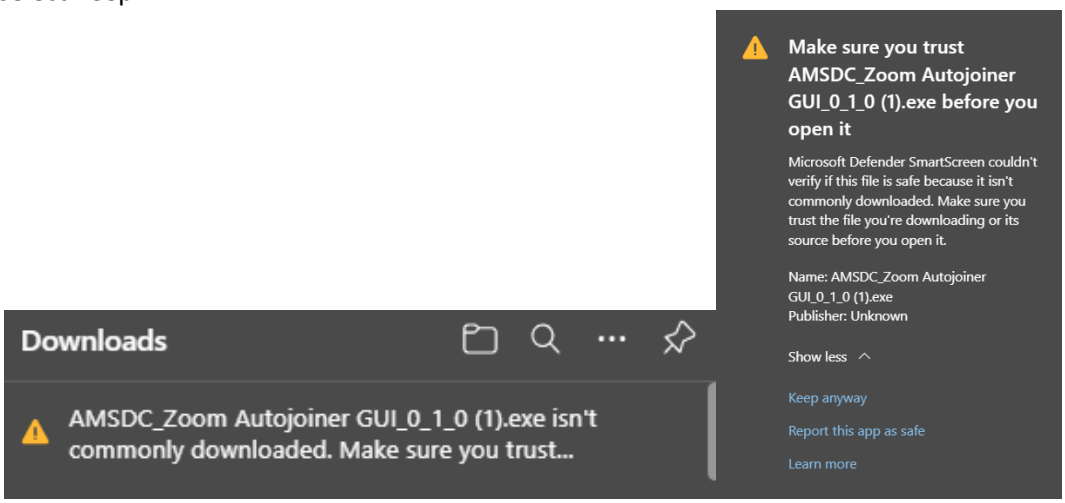
Installation

Download

The installers are updated regularly in Zoho WorkDrive. They are also uploaded GitHub Releases.

Download from Zoho WorkDrive

1. Go to the download link: https://workdrive.zohopublic.in/folder/gc2944a6c06e96e4543f3b55066196a4bd566
2. Download the latest version of the installer. Usually this is the topmost one.
3. If you get a SmartScreen warning when downloading using Edge, prese the three dots and select Keep.

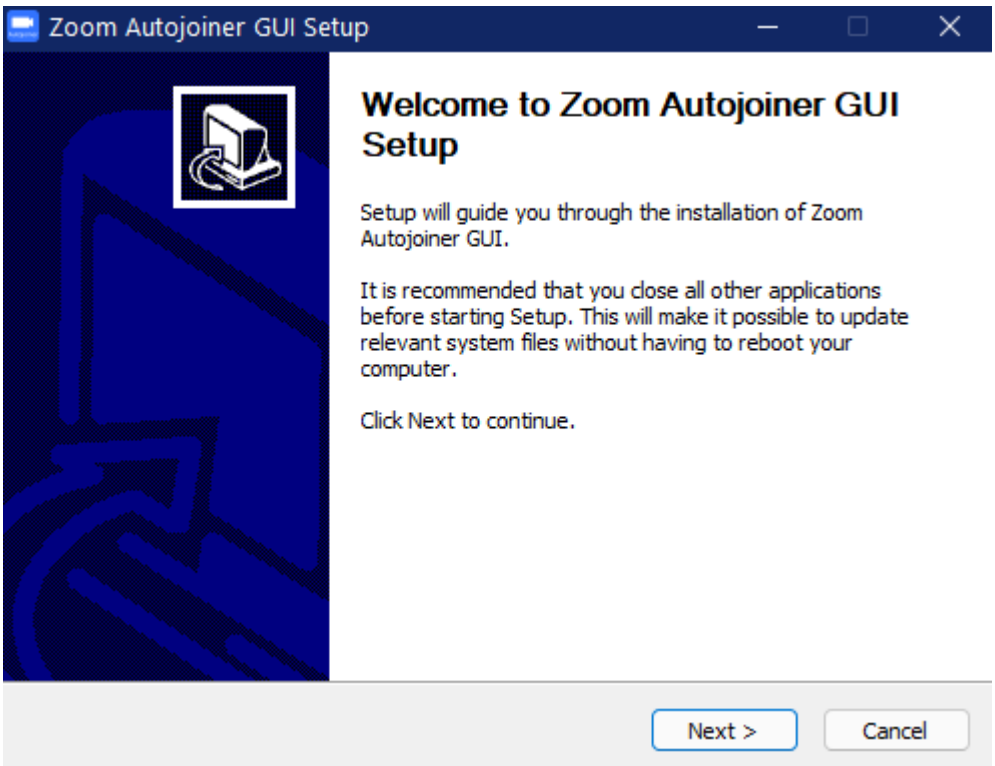


Download from Google Drive

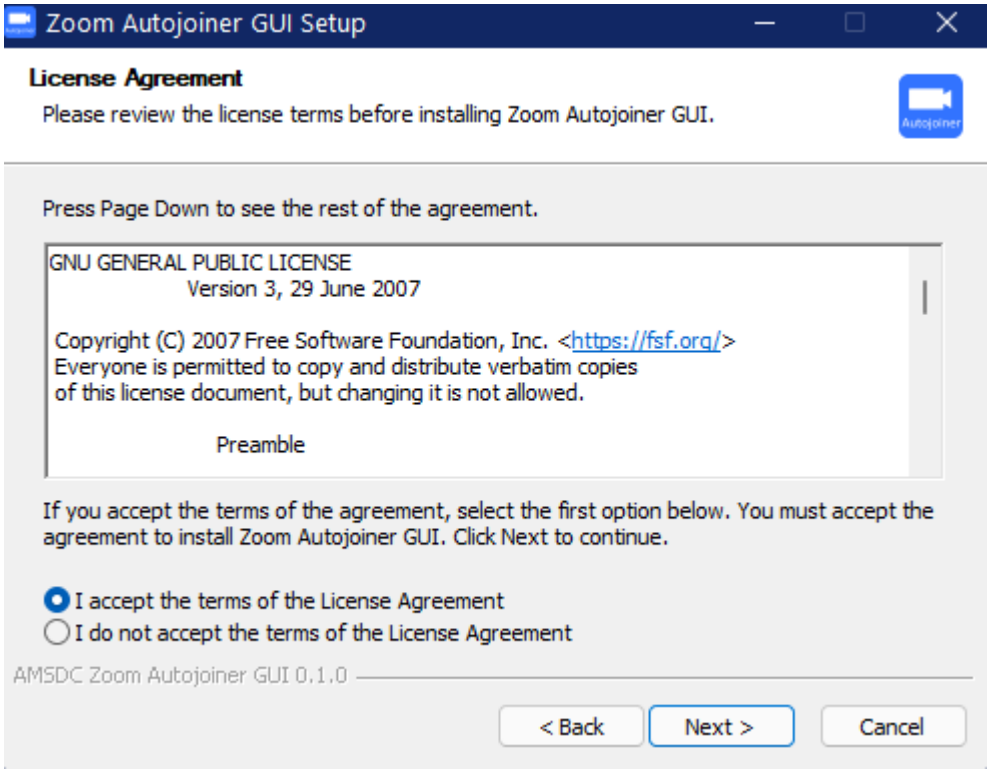
1. Go to https://drive.google.com/drive/folders/1T3kQsiWhsyb9G9BPuDHYziJjoMjGB8yk?usp=sharing
2. Follow the same steps as above.

Installation

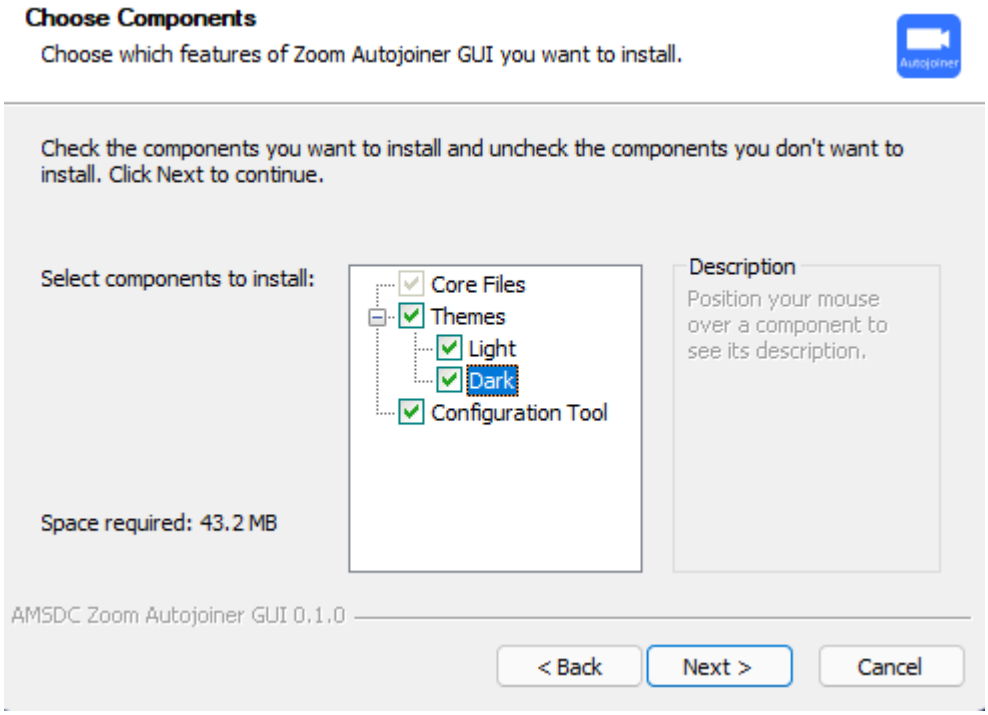
1. First configure the tool: Adding the screenshots
2. Double click on the EXE file.
3. Press *Next*



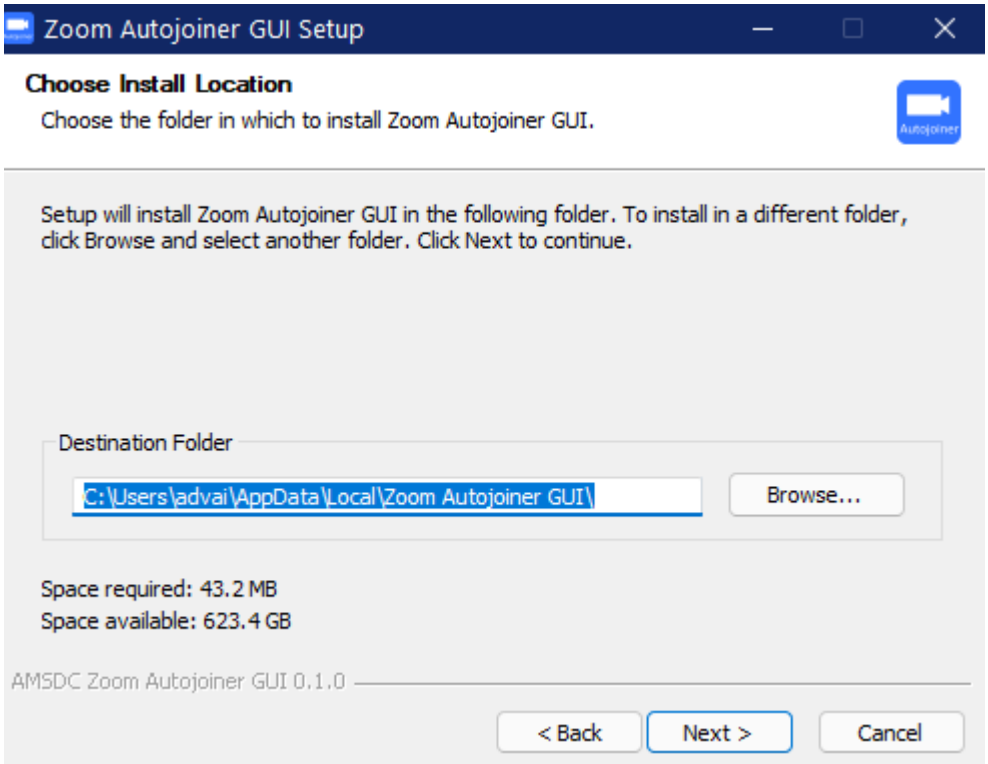
4. Accept the license and press Next



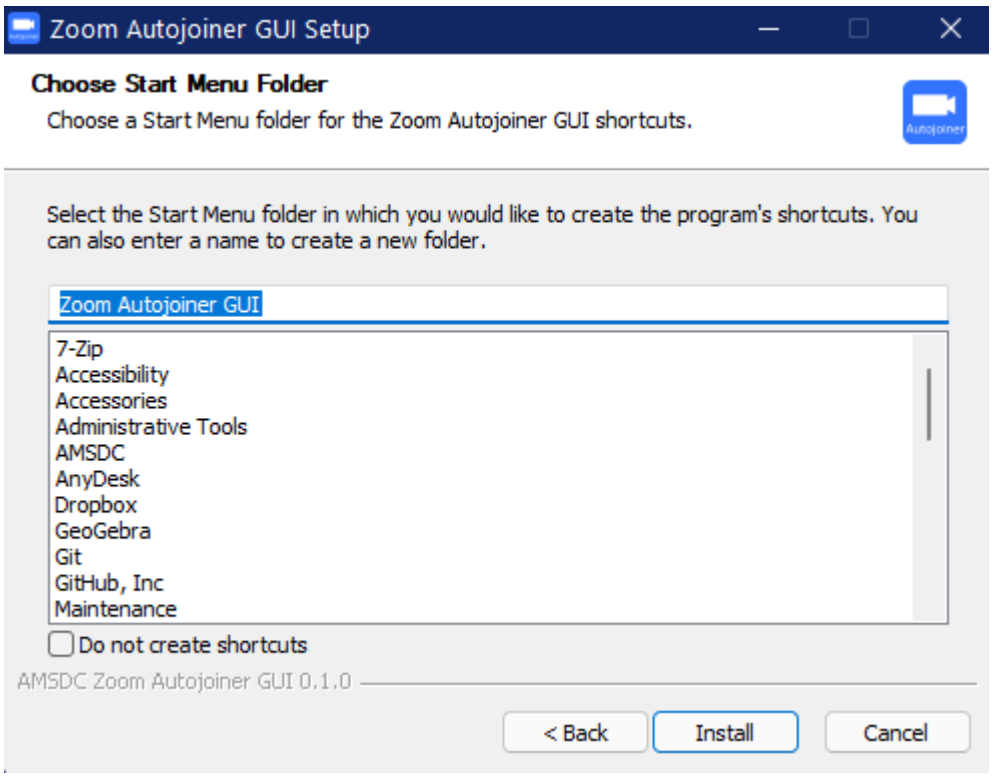
5. Ensure all the components are selected and press Next



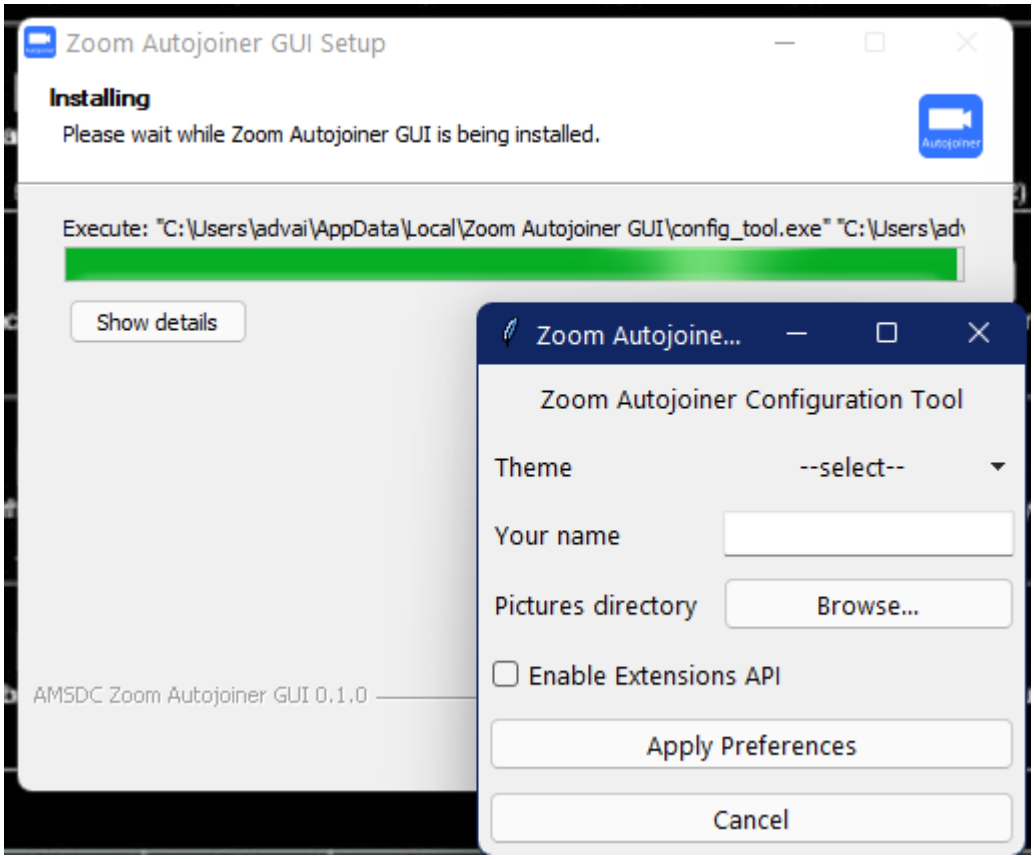
6. Leave installation location as is and press Next



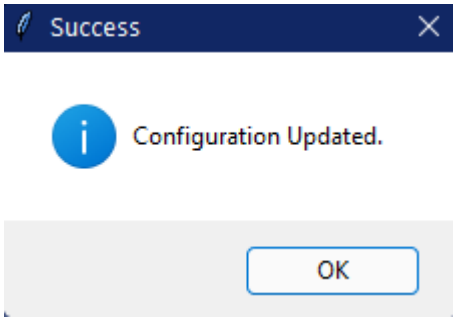
7. Create a start menu shortcut if needed. Note that this shortcut will not be deleted by the uninstaller.



8. Press Install. After a few seconds, the configuration tool will launch.

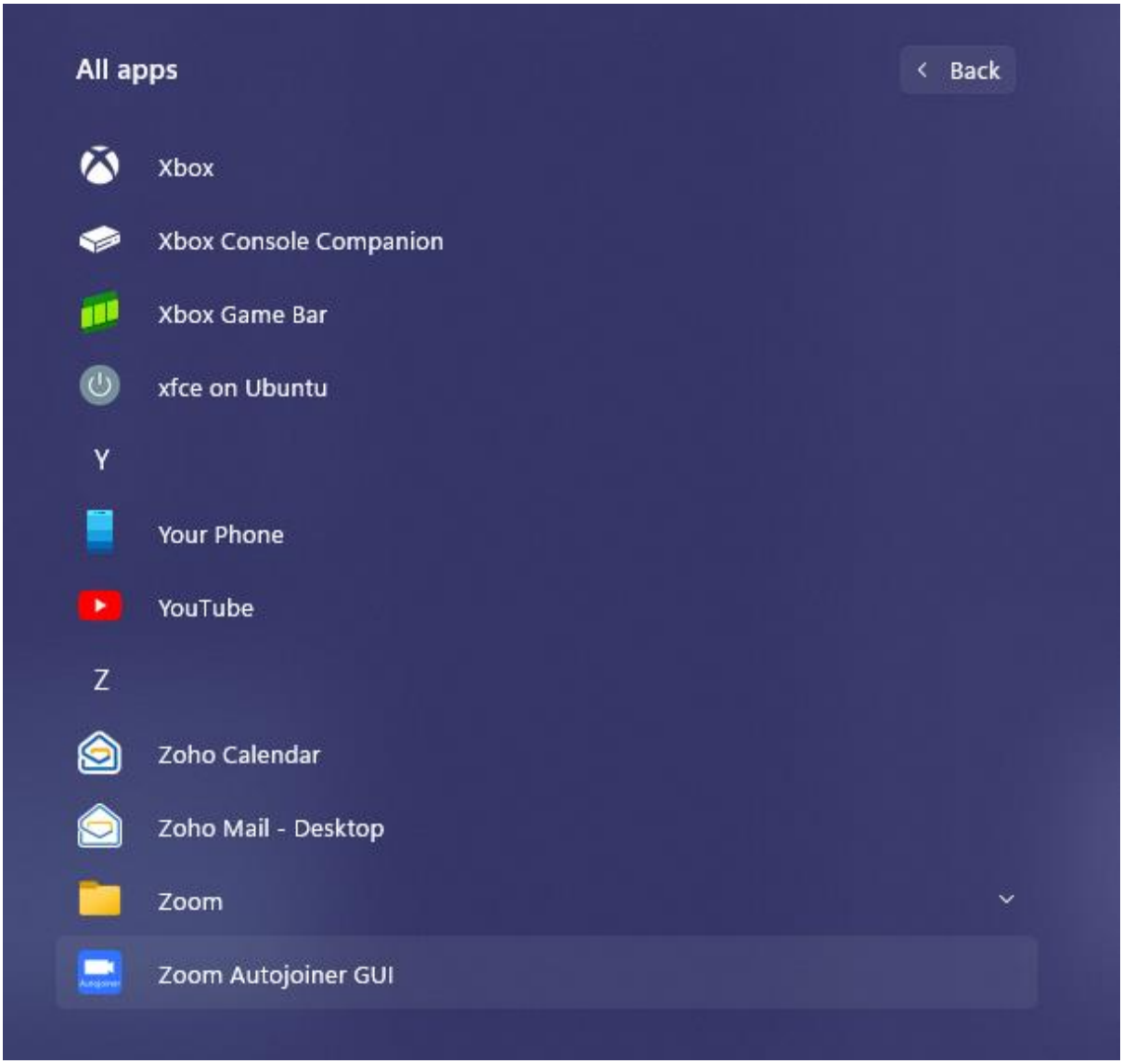


9. Fill up the form and click on Apply Preferences.



10. Click on Finish.
11. Zoom Autojoiner will be in your start menu if you chose that option, or else, it will be in *%USERPROFILE%\AppData\Local\Zoom Autojoiner GUI\ZoomAutojoinerGUI.exe* You can

launch this from Run



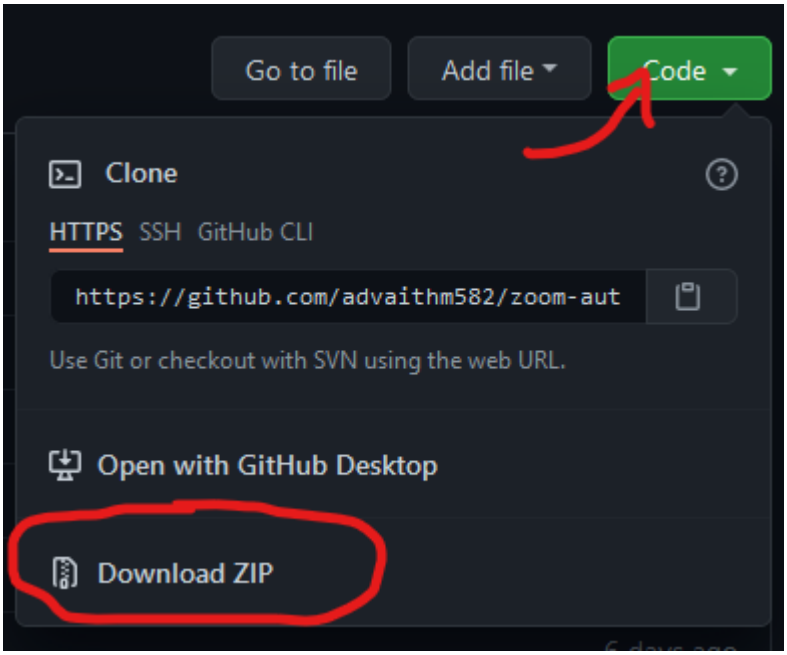
Developer Installation

Python installation

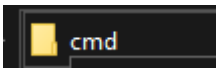
1. Go to https://www.python.org/ftp/python/3.9.6/python-3.9.6-amd64.exe.
2. In the installation options, choose Advanced.
3. Ensure Add to PATH, Tcl/TK is selected, and press Next.
4. Once setup is finished, disable max path length limit.

GitHub Download

1. In the repository, select the green 'Code' button and click 'Download ZIP'. Unzip the folder once done.



Installing dependencies

2. In the folder, in the address bar, type 'cmd' 

3. In Command Prompt, type *pip install -e .*
4. Wait for the installation to complete.

Creating the logs folder

ZAJ needs a place to store application logs. Inside the *zoom_autojoiner_gui* folder, create another folder called *logs*.

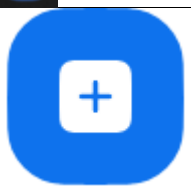
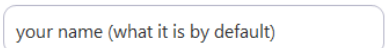
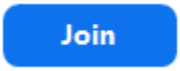
If the folder exists, delete the *.gitkeep* file.

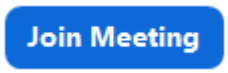
Configuration

Adding the screenshots

Add the following screenshots in a folder *images* in *zoom_autojoiner_gui*. Don't forget to pin Zoom to the taskbar.

If you used the GUI installer, add the images in any folder and select that folder in *ConfTool.exe*.

File Name	Description of the screenshot	Example
zoom_taskbar.png	A picture of Zoom in the taskbar.	
join_btn.png	A picture of the blue Join button in the Zoom home screen. to the right of orange New Meeting.	
name_box.png	A picture of your name in the Join meeting box. Below the Meeting ID prompt.	
join_btn_after_mtg_id.png	The Join Meeting button in the Meeting ID page.	

Join_meeting_btn.png	The Join Meeting button in the Enter Passcode page.	

Configuration file

- Application Configuration Sample File
- Extensions Configuration Sample File

You do not need to modify configuration files when using GUI installer.

Launching the Application

Use the Desktop/Start Menu Shortcut when installed via graphical installer.

If installed in developer mode, double-clicking __init__.py will do the trick. Or else, create a bat file to do the same:

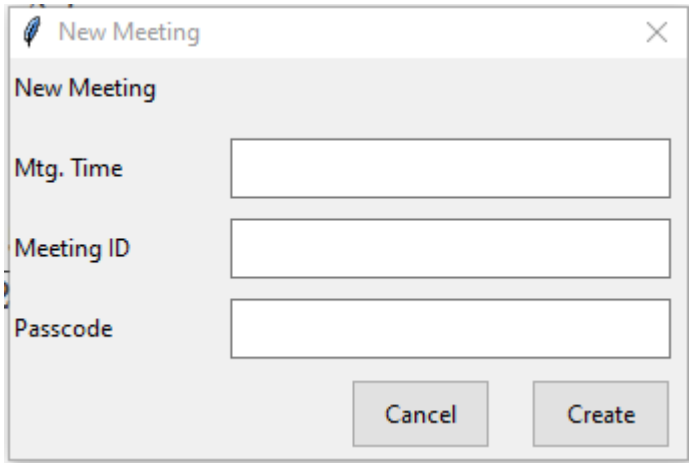
For windows:

```
CD zoom_autojoiner_gui
py __init__.py
```

Or else, compile using PyInstaller.

Adding a meeting

To add a meeting, go to Meetings -> Add meeting.



Field	Value
Mtg. Time	The time of the meeting in ISO format e.g 2021-01-01 17:00:30
Meeting ID	11 – digit Meeting ID
Passcode	Meeting Passcode

Updating/Deleting a meeting

Click *Edit Meeting* on the desired meeting and follow same procedure as above.

To delete a meeting, select the Delete option.

What to do in case the Autojoiner fails

- Use the *Join Meeting* button to join the meeting.

- In case the autojoiner fails multiple times, try one of the following:
    - Recalibrate the autojoiner by retaking the screenshots.
    - Ensure Zoom is pinned to the taskbar.
    - Open the error log (the bottom most one) and search for *traceback* to find the error.
    - Report a bug, and if you can fix it, fork, fix and open a PR.
- ZAJ doesn't work when the computer is sleeping.

Code

GitHub Link: https://github.com/11c-csproject/autojoiner

```
zoom_autojoiner_gui
    zoom_autojoiner_gui.constants
    zoom_autojoiner_gui.controllers
    zoom_autojoiner_gui.dialogs
    zoom_autojoiner_gui.extensions
    zoom_autojoiner_gui.models
    zoom_autojoiner_gui.views
    zoom_autojoiner_gui.ZoomAutojoinerGUI
```

zoom_autojoiner_gui.constants

```
# This file is part of Zoom Autojoiner GUI.

# Zoom Autojoiner GUI is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.

# Zoom Autojoiner GUI is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with Zoom Autojoiner GUI.  If not, see
<https://www.gnu.org/licenses/>.

"""Configuration Constants

This file reads run-time config constants.
"""

import json
import logging
import configparser

logger = logging.getLogger(__name__)
try:
```

```
    logger.info("Attempting to load config...")
    # with open("config/config.json", "r") as cfg_file:
    #     cfg = json.loads(cfg_file.read())
    #     ICON_FILE = cfg["ICON_FILE"]
    #     DB_URL = cfg["DB_URL"]
    #     PYAG_PICS_DIR = cfg["PYAG_PICS_DIR"]
    #     MY_NAME = cfg["MY_NAME"]
    #     THEME_FILE = "themes/" + cfg["THEME_FILE"]

    # Parse the config file.
    config = configparser.ConfigParser()


    # read it
    config.read("config/application.ini")
    # Tkinter configuration
    ICON_FILE = config["tkinter"]["icon"]
    THEME_FILE = "themes/" + config["tkinter"]["theme"] + ".thm.json"

    # Database configuration
    DB_URL = config["database"]["uri"]

    # Autojoiner configuration
    PYAG_PICS_DIR = config["autojoiner"]["pictures_dir"]
    MY_NAME = config["autojoiner"]["name"]

    """The Extensions Config Variable"""
    EXTENSIONS = config["extensions"]

except Exception as e:
    logger.error("Failed to load config, exiting...", exc_info=True)
    exit(1)
else:
    logger.info("Config loaded.")
```

zoom_autojoiner_gui.controllers

```
# This file is part of Zoom Autojoiner GUI.

# Zoom Autojoiner GUI is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.

# Zoom Autojoiner GUI is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
```

```python
# You should have received a copy of the GNU General Public License
# along with Zoom Autojoiner GUI.  If not, see
<https://www.gnu.org/licenses/>.

import json
import time
import platform
import logging
from datetime import datetime
from typing import Any, Union

import pyautogui
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, Query

from zoom_autojoiner_gui.models import Meetings
from zoom_autojoiner_gui.constants import DB_URL, MY_NAME


logger = logging.getLogger(__name__)


class TkinterTheme():
    """
    TkinterTheme Class

    This class deals with the styling of the GUI
    interface. Theme files end in the extension
    `.thm.json`. These sre simple JSON files.
    An example is gilen in the constant
    DEFAULT_THEME.

    Incase the theme file fails to open (maybe
    file not found?), the class defaults to the
    DEFAULT_THEME str (or docstring to be more
    specific.)

    There are a few bugs in this class, which
    occur when there is a space in the font name.
    I am not sure how to fix the same, hence
    leaving as is.
    """
    DEFAULT_THEME: str = """
            {
                "title" : {
                    "fg" : "white",
                    "bg" : "gray",
                    "font" : {
```

```
            "name" : "Lato",
            "size" : "20",
            "style" : "bold"
        },
        "padding" : {
            "x" : "5",
            "y" : "5"
        },
        "border" : {
            "width" : null,
            "relief" : null
        },
        "sticky" : "EW"
    },
    "table_header" : {
        "fg" : "white",
        "bg" : "#142E54",
        "font" : {
            "name" : "Lato",
            "size" : "12",
            "style" : "bold"
        },
        "padding" : {
            "x" : "5",
            "y" : "5"
        },
        "border" : {
            "width" : 2,
            "relief" : "groove"
        },
        "sticky" : "NSEW"
    },
    "table_content" : {
        "fg" : "black",
        "bg" : "white",
        "font" : {
            "name" : "Lato",
            "size" : "8",
            "style" : "bold"
        },
        "padding" : {
            "x" : "5",
            "y" : "5"
        },
        "border" : {
            "width" : 2,
            "relief" : "groove"
        },
```

```
                "sticky" : "NSEW"
            }
        }
        """
    def __init__(self, theme_file_uri:str) -> None:
        try:
            with open(theme_file_uri, "r") as file_handle:
                self.json = json.loads(file_handle.read())
        except:
            logger.warning("Failed to load theme file, using default...",
                exc_info=True)
            self.json = json.loads(self.DEFAULT_THEME)
        else:
            logger.info("Loaded theme file")

    def get_styling(self, style_name: str) -> dict:
        """get_styling

        Gets the style to be used for given style name.

        Args:
            style_name: The name of the style.

        Returns:
            dict: The dict of TK styling to be unpacked and passed.
        """
        if style_name in self.json:
            the_dict = {
                    "fg" : self.json[style_name]["fg"],
                    "bg" : self.json[style_name]["bg"],
                    "padx" : self.json[style_name]["padding"]["x"],
                    "pady" : self.json[style_name]["padding"]["y"],
                    "borderwidth" : self.json[style_name]["border"]["width"],
                    "relief" : self.json[style_name]["border"]["relief"],
                    "font" : (self.json[style_name]["font"]["name"]
                        + " "
                        + self.json["table_header"]["font"]["size"]
                        + " "
                        + self.json["table_header"]["font"]["style"])
                }
            return the_dict
        else:
            # no style available
            return {}


class DatabaseHandler():
    """DatabaseHandler
```

```
    This class handles database related stuff. It's like a waiter in a
    restaurant - it handles communications between the customer (view)
    and the cook (database). In technical terms, it is a controller in
    the MVC architecture.

    Args:
        database_uri:
            The URI of the database, in SQLAlchemy format.
    """
    def __init__(self, database_uri: str) -> None:
        # engine = create_engine(DB_URL)
        engine = create_engine(database_uri)
        Session = sessionmaker(bind=engine)
        self.__db_session = Session()

    def add_mtg(self, meeting_id: str, meeting_password: str,
            meeting_time: datetime, meeting_provider: str = "ZM",
            auto_commit: bool = True) -> None:
        """add_mtg

        Adds a meeting to the database.

        Args:
            meeting_id: The Meeting ID
            meeting_password: Mtg. passcode
            meeting_time: Datetime of meeting
            meeting_provider: Meeting Provider. Defaults to "ZM".
            auto_commit: Whether to autosave changes. Defaults to True.
        """
        mtg = Meetings(mtg_provider=meeting_provider, mtg_id=meeting_id,
            mtg_password=meeting_password, mtg_time=meeting_time)
        self.__db_session.add(mtg)
        if auto_commit:
            self.commit_changes()

    def delete_mtg(self, rec_id: int, auto_commit: bool = True) -> None:
        """delete_mtg

        Deletes a meeting from the database.

        Args:
            rec_id: The Record ID in database
            auto_commit:
                Whether to auto commit changes. Defaults to True.
        """
        to_delete = self.__db_session.query(Meetings).filter_by(id=
            rec_id).one()
        self.__db_session.delete(to_delete)
```

```
        if auto_commit:
            self.commit_changes()

    def update_mtg(self, db_id: int, meeting_id: str, meeting_password: str,
            meeting_time: datetime, meeting_provider: str = "ZM",
            auto_commit: bool = True) -> None:
        """update_mtg

        Update meeting data in the database.

        Args:
            db_id (int): The Record ID in database.
            meeting_id (str): The Meeting ID.
            meeting_password (str): The Meeting password.
            meeting_time (datetime.datetime): The time of the meeting.
            meeting_provider (str, optional):
                Meeting provider code. Defaults to "ZM".
            auto_commit (bool, optional):
                Whether to autosave changes. Defaults to True.
        """
        to_update = self.__db_session.query(Meetings).filter_by(id=
            db_id).one()
        to_update.mtg_provider = meeting_provider
        to_update.mtg_id = meeting_id
        to_update.mtg_password = meeting_password
        to_update.mtg_time = meeting_time
        if auto_commit:
            self.commit_changes()

    def get_mtg_data_to_list(self) -> list[dict[str, Any]]:
        """get_mtg_data_to_list

        Queries meeting data from SQL database and outputs it as a
        list cum dict

        Returns:
            list: List with dict of mtg data.
        """
        output_list = [] # Output list
        query = self.__db_session.query(Meetings).order_by(
            Meetings.mtg_time)
        for record in query:
            mtg_data = {
                "id" : record.id,
                "mtg_provider" : record.mtg_provider,
                "mtg_id" : record.mtg_id,
                "mtg_password" : record.mtg_password,
                "mtg_time": record.mtg_time
```

```
        }
        # print(type(record.mtg_time))
        output_list.append(mtg_data)
    return output_list

def get_single_mtg_data_to_list(self, record_id: str) -> dict[str, Any]:
    """get_single_mtg_data_to_list

    Queries only one meeting from SQL database and outputs it as a
    dict

    Note:
        The method name is redundant, as it was imported from
        `cryptocurrency-portfolio`_. It will be changed before the
        release of the 1.x.x series. To ensure backwards
        compatibility, this method will be made to call the dict
        method.

    Args:
        record_id (str): ID of record in database.

    Returns:
        dict[str, Any]: Dict containing meeting data of record ID.

    .. _cryptocurrency-portfolio: https://tinyurl.com/48a7y5cw
    """
    # output_list = [] # Output list
    record = self.__db_session.query(Meetings).filter_by(id=record_id) \
        .one()
    output_list = {
        "id" : record.id,
        "mtg_provider" : record.mtg_provider, # mtg_provider
        "mtg_id" : record.mtg_id, # No. of mtg owned
        "mtg_password" : record.mtg_password,
        "mtg_time": record.mtg_time # Price per mtg at time of purchase
    }
    # output_list.append(mtg_data)
    return output_list

def get_mtg_with_time(self, time: datetime) -> list[Query]:
    """get_mtg_with_time

    Get the meeting data, using time as a filter.
    Used for checking if it is time to join the meeting.

    Note:
        This API is not used and is kept for the future. The reason
        is that this heavily depends on the seconds parameter, and
```

```
            if the autojoiner checks for meetings every second, it will
            consume too many resources.

            If you have a solution to this problem, please fork the
            repo and feel free to open a PR.

        Args:
            time: The time filter.

        Returns:
            list[Query]:
                It is a normal SQLAlchemy object which has a list, and
                in each an object.
        """
        query = self.__db_session.query(Meetings).filter(Meetings.mtg_time \
            == time).order_by(Meetings.mtg_time)
        # logger.debug(type(query))
        return query

    def truncate_table(self, auto_commit: bool = True) -> None:
        """truncate_table

        Clear all data in the table.
        This API is yet to be implemented. As always, this method was
        taken from `cryptocurrency-portfolio`_.

        Args:
            auto_commit:
                Whether to autosave changes. Defaults to True.

        .. _cryptocurrency-portfolio: https://tinyurl.com/48a7y5cw
        """
        # Remove all meetings
        self.__db_session.query(Meetings).delete()
        if auto_commit:
            self.commit_changes()


    def commit_changes(self) -> None:
        """Make changes reflect in database"""
        self.__db_session.commit()


class ATLParser():
    """
    AuTomation Language (ATL) parser

    This class parses ATL code, which ends in the file extension
    .atl. These are simple JSON files. For the format, see
```

```
    specification.atl in the 'scripts' folder.

    ATL code will be used internally by Zoom Autojoiner to enable
    joining of meetings on different platforms (OSes).

    Note:
        The ATL parser will be implemented as an extension and not in
        the ZAJ core. Infact, this idea may be DROPPED itself, as it
        can impact performance.
    """
    pass


class Autojoiner():
    """
    Autojoiner

    This class contains functions that will help in automatically
    joining Zoom meetings.

    Note:
        This class may be moved to an extension in the next major
        release.

    Args:
        image_dir: The directory where images are stored.
    """
    def __init__(self, image_dir: str = "") -> None:
        self.__dbh = DatabaseHandler(DB_URL) # dbh is DB handle
        self.IMG_DIR = image_dir # e.g /usr/share/

    def get_image_path(self, filename: str) -> str:
        """get_image_path

        Get the path to the image with correct slash for the OS.

        Note:
            It will be more efficient if os.path.join is used instead.

        Args:
            filename (str): The name of the file.

        Returns:
            str: The full directory path.
        """
        # Get the path of the image from the filename.
        # Get the directory slash.
        # Why couldn't Windows be like the rest? :(
        OS_SLASH = "\\" if platform.system() == "Windows" else "/"
```

```python
    # Get the directory/folder.
    if self.IMG_DIR[-1] == "\\" or self.IMG_DIR[-1] == "/":
        img_directory = self.IMG_DIR
    else:
        img_directory = self.IMG_DIR + OS_SLASH

    # Get Filename
    if filename[0] == "\\" or filename[0] == "/":
        final_filename = filename[1:]
    else:
        final_filename = filename

    # Return the file directory.
    return img_directory + final_filename

def check_for_meeting(self) -> Union[dict, bool]:
    """check_for_meeting

    Checks if there is a meeting at the current time.

    Returns:
        Optional[dict, bool]:
            Returns list of meetings, if a meeting is present at
            that time.
    """
    logger.debug("Check For Meeting Block entered")
    mtgs_list = self.__dbh.get_mtg_data_to_list()
    logger.debug(str(mtgs_list))
    now = datetime.now()
    now_string = now.strftime("%d-%m-%y %H:%M")
    logger.debug("Nowstring %s", now_string)
    return_str = ""
    return_dict = {}
    for mtg_dict in mtgs_list:
        mtg_date = mtg_dict["mtg_time"].strftime("%d-%m-%y %H:%M")
        logger.debug("Mtg date string %s", mtg_date)
        if mtg_date == now_string:
            logger.debug("True")
            return_str = True
            return_dict = mtg_dict
        else:
            logger.debug("False")
            return_str = False

    logger.debug("Return Dict %s", str(return_dict))
    return return_dict if return_str else False
```

```python
    def join_zm_mtg(self, id: str, password: str) -> None:
        """join_zm_mtg

        Joins a zoom meeting

        Args:
            id (str): Meeting ID
            password (str): Meeting Passcode
        """
        try:
            # IMG_DIR = self.IMG_DIR
            # (start_x, start_y) =
pyautogui.center(pyautogui.locateOnScreen(IMG_DIR + "start.png"))
            # print(start_x, start_y)
            pyautogui.click(self.get_image_path("zoom_taskbar.png"))
            time.sleep(0.75)
            pyautogui.click(self.get_image_path("join_btn.png"))
            time.sleep(0.75)
            pyautogui.write(id, interval=0.25)
            time.sleep(0.75)
            pyautogui.click(self.get_image_path("name_box.png"))
            time.sleep(0.25)
            pyautogui.hotkey('ctrl','a')
            time.sleep(0.25)
            pyautogui.press('backspace')
            time.sleep(0.25)
            pyautogui.write(MY_NAME, interval=0.25)
            time.sleep(0.75)
            pyautogui.click(self.get_image_path("join_btn_after_mtg_id.png"))
            time.sleep(5)
            pyautogui.write(password, interval=0.25)
            time.sleep(0.75)
            pyautogui.click(self.get_image_path("join_meeting_btn.png"))
        except:
            logger.error("Failed to join meeting", exc_info=True)
        else:
            logger.info("Joined Meeting successfully")

# Todo:
# Finish theming class
# After that finish Autojoiner class
```

zoom_autojoiner_gui.dialogs

```python
# This file is part of Zoom Autojoiner GUI.

# Zoom Autojoiner GUI is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
```

```
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.

# Zoom Autojoiner GUI is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with Zoom Autojoiner GUI.  If not, see
<https://www.gnu.org/licenses/>.

import tkinter as tk
from tkinter import ttk, messagebox
import tkinter.font as tkFont
import datetime
import platform
from zoom_autojoiner_gui.controllers import DatabaseHandler
from zoom_autojoiner_gui.constants import DB_URL

class NewMeetingDialog(tk.Toplevel):
    def __init__(self, tk_frame_handle = None, tk_root_element = None):
        """This class shows the New Meeting Dialog box."""
        # DB handle
        self.__dbh = DatabaseHandler(DB_URL)

        # TK Root Element
        if tk_root_element:
            # Attach it to var
            self.tk_root_element = tk_root_element
        else:
            # Get Root element from frame
            self.tk_root_element = None

        # Toplevel Initialization
        try:
            # print("HI")
            super().__init__(self.tk_root_element)
        except:
            super().__init__()

        # TK Frame
        if tk_frame_handle:
            # If given use it
            self.tk_frame_handle = tk_frame_handle
        elif self.tk_root_element:
            # Take from root element
            self.tk_frame_handle = self.tk_root_element.__meeting_list_frame
```

```
        else:
            # None it
            self.tk_frame_handle = None

        #setting title
        self.title("New Meeting")
        #setting window size
        width=338
        height=200
        screenwidth = self.winfo_screenwidth()
        screenheight = self.winfo_screenheight()
        alignstr = '%dx%d+%d+%d' % (width, height, (screenwidth - width) / 2,
(screenheight - height) / 2)
        self.geometry(alignstr)
        self.resizable(width=False, height=False)
        # For modal dialog
        self.grab_set()
        try:
            # if platform.system() == "Windows": self.attributes('-
toolwindow', True)
            self.transient(self.tk_root_element)
            # self.attributes('-topmost', True)
        except:
            pass

        # Meeting Time Label
        self.GLabel_3=ttk.Label(self)
        self.GLabel_3["text"] = "Mtg. Time"
        self.GLabel_3.place(x=0,y=40,width=101,height=30)

        # Title Label
        self.GLabel_276=ttk.Label(self)
        self.GLabel_276["justify"] = "center"
        self.GLabel_276["text"] = "New Meeting"
        self.GLabel_276.place(x=0,y=0,width=350,height=30)

        # Meeting Time Entry
        self.DateTimeEntry=ttk.Entry(self)
        self.DateTimeEntry.place(x=110,y=40,width=220,height=30)

        # Meeting ID label
        self.GLabel_378=ttk.Label(self)
        # ft = tkFont.Font(family='Times',size=10)
        self.GLabel_378["text"] = "Meeting ID"
        self.GLabel_378.place(x=0,y=80,width=100,height=30)

        # Meeting ID entry
        self.MeetingIDEntry=ttk.Entry(self)
```

```
        self.MeetingIDEntry.place(x=110,y=80,width=220,height=30)

        # Meeting Passcode Label
        self.GLabel_48=ttk.Label(self)
        self.GLabel_48["text"] = "Passcode"
        self.GLabel_48.place(x=0,y=120,width=101,height=30)

        # Meeting Password Entry
        self.MeetingPasscodeEntry=ttk.Entry(self)
        self.MeetingPasscodeEntry.place(x=110,y=120,width=220,height=30)

        # Create Meeting Button
        self.CreateMtgButton=ttk.Button(self)
        # CreateMtgButton["bg"] = "#f0f0f0"
        # ft = tkFont.Font(family='Times',size=10)
        # CreateMtgButton["font"] = ft
        # CreateMtgButton["fg"] = "#000000"
        # CreateMtgButton["justify"] = "center"
        self.CreateMtgButton["text"] = "Create"
        self.CreateMtgButton.place(x=260,y=160,width=70,height=35)
        self.CreateMtgButton["command"] = self.CreateMtgButton_command

        # Cancel New Meeting Button
        self.CancelButton=ttk.Button(self)
        # CancelButton["bg"] = "#f0f0f0"
        # ft = tkFont.Font(family='Times',size=10)
        # CancelButton["font"] = ft
        # CancelButton["fg"] = "#000000"
        # CancelButton["justify"] = "center"
        self.CancelButton["text"] = "Cancel"
        self.CancelButton.place(x=170,y=160,width=70,height=35)
        self.CancelButton["command"] = self.CancelButton_command

    def CreateMtgButton_command(self):
        try:
            datetimeobj=datetime.datetime.strptime(self.DateTimeEntry.get(),
"%Y-%m-%d %H:%M:%S")
            self.__dbh.add_mtg(self.MeetingIDEntry.get(),
self.MeetingPasscodeEntry.get(), datetimeobj)
        except Exception as e:
            messagebox.showerror("Error", "An exception has occured.\nError
Details:\n%s" % (str(e)))
        else:
            messagebox.showinfo("Information", "Meeting Added.")
            try:
                self.tk_frame_handle.reload_table()
            except:
```

```
                messagebox.showinfo("Information", "Failed to refresh table
data. Please refresh manually.")
            self.destroy()


    def CancelButton_command(self):
        self.destroy()

class EditMeetingDialog(tk.Toplevel):
    def __init__(self, record_id, tk_frame_handle = None, tk_root_element =
None):
        # DB handle
        self.__dbh = DatabaseHandler(DB_URL)

        # TK Root Element
        if tk_root_element:
            # Attach it to var
            self.tk_root_element = tk_root_element
        else:
            # Get Root element from frame
            self.tk_root_element = None

        # Toplevel Initialization
        try:
            # print("HI")
            super().__init__(self.tk_root_element)
        except:
            super().__init__()

        self.record_id = record_id

        # TK Frame
        if tk_frame_handle:
            # If given use it
            self.tk_frame_handle = tk_frame_handle
        elif self.tk_root_element:
            # Take from root element
            self.tk_frame_handle = self.tk_root_element.__meeting_list_frame
        else:
            # None it
            self.tk_frame_handle = None

        # Get values from DB
        try:
            mtg_data = self.__dbh.get_single_mtg_data_to_list(self.record_id)
        except Exception as e:
            messagebox.showerror("Error", "An exception has occured.\nError
Details:\n%s" % (str(e)))
```

```
            self.destroy()

        #setting title
        self.title("Edit Meeting")

        #setting window size
        width=338
        height=200
        screenwidth = self.winfo_screenwidth()
        screenheight = self.winfo_screenheight()
        alignstr = '%dx%d+%d+%d' % (width, height, (screenwidth - width) / 2,
(screenheight - height) / 2)
        self.geometry(alignstr)
        self.resizable(width=False, height=False)
        # For modal dialog
        self.grab_set()
        try:
            # if platform.system() == "Windows": self.attributes('-
toolwindow', True)
            self.transient(self.tk_root_element)
            # self.attributes('-topmost', True)
        except:
            pass

        # Title Label
        self.GLabel_276=ttk.Label(self)
        self.GLabel_276["justify"] = "center"
        self.GLabel_276["text"] = "Edit Meeting (Record ID %d)" %
(self.record_id)
        self.GLabel_276.place(x=0,y=0,width=350,height=30)

        # Meeting Time Label
        self.GLabel_3=ttk.Label(self)
        self.GLabel_3["text"] = "Mtg. Time"
        self.GLabel_3.place(x=0,y=40,width=101,height=30)

        # Meeting Time Entry
        self.DateTimeEntry=ttk.Entry(self)
        self.DateTimeEntry.insert(0, mtg_data["mtg_time"])
        self.DateTimeEntry.place(x=110,y=40,width=220,height=30)

        # Meeting ID label
        self.GLabel_378=ttk.Label(self)
        self.GLabel_378["text"] = "Meeting ID"
        self.GLabel_378.place(x=0,y=80,width=100,height=30)

        # Meeting ID entry
        self.MeetingIDEntry=ttk.Entry(self)
```

```
        self.MeetingIDEntry.insert(0, mtg_data["mtg_id"])
        self.MeetingIDEntry.place(x=110,y=80,width=220,height=30)

        # Meeting Passcode Label
        self.GLabel_48=ttk.Label(self)
        self.GLabel_48["text"] = "Passcode"
        self.GLabel_48.place(x=0,y=120,width=101,height=30)

        # Meeting Password Entry
        self.MeetingPasscodeEntry=ttk.Entry(self)
        self.MeetingPasscodeEntry.insert(0, mtg_data["mtg_password"])
        self.MeetingPasscodeEntry.place(x=110,y=120,width=220,height=30)

        # Update Meeting Button
        self.UpdateMtgButton=ttk.Button(self)
        # UpdateMtgButton["bg"] = "#f0f0f0"
        # ft = tkFont.Font(family='Times',size=10)
        # UpdateMtgButton["font"] = ft
        # UpdateMtgButton["fg"] = "#000000"
        # UpdateMtgButton["justify"] = "center"
        self.UpdateMtgButton["text"] = "Update"
        self.UpdateMtgButton.place(x=260,y=160,width=70,height=35)
        self.UpdateMtgButton["command"] = self.UpdateMtgButton_command

        # Delete New Meeting Button
        self.DeleteMtgButton=ttk.Button(self)
        # CancelButton["bg"] = "#f0f0f0"
        # ft = tkFont.Font(family='Times',size=10)
        # CancelButton["font"] = ft
        # CancelButton["fg"] = "#000000"
        # CancelButton["justify"] = "center"
        self.DeleteMtgButton["text"] = "Delete"
        self.DeleteMtgButton.place(x=170,y=160,width=70,height=35)
        self.DeleteMtgButton["command"] = self.DeleteMtgButton_command

        # Cancel New Meeting Button
        self.CancelButton=ttk.Button(self)
        self.CancelButton["text"] = "Cancel"
        self.CancelButton.place(x=80,y=160,width=70,height=35)
        self.CancelButton["command"] = self.CancelButton_command

    def UpdateMtgButton_command(self):
        """Update meeting data."""
        try:
            datetimeobj=datetime.datetime.strptime(self.DateTimeEntry.get(),
"%Y-%m-%d %H:%M:%S")
            self.__dbh.update_mtg(self.record_id, self.MeetingIDEntry.get(),
self.MeetingPasscodeEntry.get(), datetimeobj)
```

```
        except Exception as e:
            messagebox.showerror("Error", "An exception has occured.\nError
Details:\n%s" % (str(e)))
        else:
            messagebox.showinfo("Information", "Meeting Updated.")
            try:
                self.tk_frame_handle.reload_table()
            except:
                messagebox.showinfo("Information", "Failed to refresh table
data. Please refresh manually.")
            self.destroy()


    def CancelButton_command(self):
        """Close the Window"""
        self.destroy()

    def DeleteMtgButton_command(self):
        """Delete meeting"""
        result = messagebox.askquestion("Warning", "Deleted meetings are
DELETED FOREVER.\nThis action is IRREVERSIBLE!!. \nProceed?")
        if result == "yes":
            try:
                self.__dbh.delete_mtg(self.record_id)
            except Exception as e:
                messagebox.showerror("Error", "An exception has
occured.\nError Details:\n%s" % (str(e)))
            else:
                messagebox.showinfo("Information", "Meeting Deleted.")
                try:
                    self.tk_frame_handle.reload_table()
                except:
                    messagebox.showinfo("Information", "Failed to refresh
table data. Please refresh manually.")
                self.destroy()




if __name__ == "__main__":
    app = NewMeetingDialog()
```

zoom_autojoiner_gui.extensions

```
# This file is part of Zoom Autojoiner GUI.

# Zoom Autojoiner GUI is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
```

```
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.

# Zoom Autojoiner GUI is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with Zoom Autojoiner GUI.  If not, see
<https://www.gnu.org/licenses/>.

import logging
import sys
import os
import json
import tkinter as tk
from importlib import import_module
from configparser import ConfigParser

# from zoom_autojoiner_gui.views import (
#     MainWindow,
#     ApplicationMenuBar,
#     MeetingListFrame
# )
from zoom_autojoiner_gui.constants import EXTENSIONS


logger = logging.getLogger(__name__) # This creates logger for this file.


#: bool : Whether extension API is enabled or not.
enabled = EXTENSIONS.getboolean("enabled")


class ExtensionHandler():
    """ExtensionHandler

    This class deals with the handling of ZAJ
    Python Extensions.

    Args:
        config: The Extensions Config dict.

    Returns:
        NoneType
    """

    #: tuple : The tuple of permissions.
```

```
permissions = (
    "main_window",
    "menu_bar",
    "meeting_list_frame"
)

def __init__(self, config: ConfigParser) -> None:
    self.basic_config = config
    self.config = ConfigParser()

    # real_path = os.path.realpath(__file__)
    # dir_path = os.path.dirname(real_path)
    dir_path = '' # not package path -- makes more sense

    logger.debug("DIR path %s" % dir_path)

    # Extension configuration
    self.config.read(os.path.join(dir_path, "config",
        self.basic_config['config'] + '.ini'))

    # Extension DIR

    self.extensions_dir = os.path.join(dir_path, self.basic_config['dir'])

    self.extensions = {}

    logger.debug(os.path.join(dir_path, "config",
        self.basic_config['config'] + '.ini'))

def get_ext(self) -> list:
    """get_ext

    Gets the enabled extensions from
    the extension configuration file.

    Returns:
        The list of enabled extensions.
    """
    return json.loads(self.config['enabled']['extensions'])

def get_extension_permission(self, ext_name: str,
        permission_name: str) -> bool:
    """get_extension_permissions

    Get the permission for an extension.

    Args:
        ext_name: The name of an extension.
```

```
        permission_name: The name of a permission to check.

    Returns:
        A boolean value.
        If the permission is granted, it returns True. Or else it
        returns False.
    """
    return self.config.getboolean(ext_name, permission_name,
        fallback=False)

def load_extensions(self) -> bool:
    """load_extensions

    Load all the extensions.

    Returns:
        True if everything went fine
        False if even one extension failed

    Note:
        If one extension failed, others are still executed.
        Like a parallel circuit, if one bulb fuses others don't.
    """
    all_ext_ran = True

    sys.path.insert(0, self.extensions_dir)

    for extension in self.get_ext():
        try:
            self.extensions[extension] = import_module(extension)
        except:
            logger.error(f"Failed to load extension {extension}!",
                exc_info=True)
            all_ext_ran = False

    return all_ext_ran

def give_extensions_prefs(self) -> bool:
    """give_extensions_prefs

    Give extensions their preferences. This is stored in the same
    section where their permissions are stored. The preferences are
    passed to a function in the extension, `get_prefs` where they
    are given the ConfigParser Object of their section in an argument
    `prefs_dict`.

    Returns:
        True if everything went fine
```

```
            False if even one extension failed

        Note:
            An extension should have implemented the module level function
            `get_prefs` in order for this to work.
            If one extension failed, others are still executed.
            Like a parallel circuit, if one bulb fuses others don't.
        """
        all_ext_ran = True

        for extension in self.get_ext():
            try:
                self.extensions[extension].get_prefs(prefs_dict=self.config[ex
tension])
            except:
                logger.error(f"Failed to give obj to ext {extension}!",
                    exc_info=True)
                all_ext_ran = False

        return all_ext_ran

    def give_extensions_objects(self, main_window: tk.Tk = None,
            menu_bar: tk.Menu = None,
            meeting_list_frame: tk.Frame = None) -> bool:
        """give_extensions_objects

        Give extensions the currently used Tkinter objects. It also depends
        on the permissions granted to the extensions. For example, if an
        extension was not given meeting_list_frame permission, then a None
        object will be passed to it instead.

        Args:
            main_window: The Main window of the ZAJ.
            menu_bar: Application menu Bar
            meeting_list_frame: Meeting lsit frame object.

        Returns:
            True if everything went fine
            False if even one extension failed

        Note:
            An extension should have implemented the module level function
            `set_objects` in order for this to work.
            If one extension failed, others are still executed.
            Like a parallel circuit, if one bulb fuses others don't.
        """

        # Whether all extensions ran successfully
```

```
        all_ext_ran = True

        for extension in self.extensions:
            try:
                objects = {}
                for permission in self.permissions:
                    if self.get_extension_permission(extension, permission):
                        objects[permission] = locals()[permission]
                        # logger.debug(f"{objects}, {locals()}")
                    else:
                        objects[permission] = None
                self.extensions[extension].set_objects(
                        objects["main_window"],
                        objects["menu_bar"],
                        objects["meeting_list_frame"]
                    )
                logger.debug(f"EXT_NAME{extension}\nOBJECTS:{objects}")
            except:
                logger.error(f"Failed to set extension {extension}!",
                    exc_info=True)
                all_ext_ran = False

        return all_ext_ran

    def run_extensions(self) -> bool:
        """run_extensions

        Run all the extensions.

        Returns:
            True if everything went fine
            False if even one extension failed

        Note:
            If one extension failed, others are still executed.
            Like a parallel circuit, if one bulb fuses others don't.
        """

        all_ext_ran = True

        for extension in self.extensions:
            try:
                self.extensions[extension].main()
            except:
                logger.error(f"Failed to run extension {extension}!",
                    exc_info=True)
                all_ext_ran = False
```

```
        return all_ext_ran

class ExtensionAPI():
    """ExtensionAPI

    The Extension API object can be used for:
    1. Adding new Autojoiners

    Args:
        name (str): Name or the extension as in __name__.
    """
    extension_handle = None
    def __init__(self, name: str) -> None:
        pass


def load_extensions():
    if EXTENSIONS.getboolean("enabled"):
        # if extensions are enabled
        global ext_class
        ext_class = ExtensionHandler(EXTENSIONS)
        if ext_class.load_extensions():
            ext_class.run_extensions()
```

zoom_autojoiner_gui.models

```
# This file is part of Zoom Autojoiner GUI.

# Zoom Autojoiner GUI is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.

# Zoom Autojoiner GUI is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with Zoom Autojoiner GUI.  If not, see
<https://www.gnu.org/licenses/>.

from sqlalchemy import create_engine
from sqlalchemy import (
    Column,
    Integer,
    String,
    REAL,
    DateTime
```

```
)
from sqlalchemy.orm import declarative_base
from sqlalchemy.orm import sessionmaker

from zoom_autojoiner_gui.constants import DB_URL


engine = create_engine(DB_URL)
# Session = sessionmaker(bind=engine)
Base = declarative_base()

class Meetings(Base):
    """Meetings

    Class containing the Meetings table.
    """
    __tablename__ = 'meetings'

    id = Column(Integer, primary_key=True)
    mtg_provider = Column(String)
    mtg_id = Column(String)
    mtg_password = Column(String)
    mtg_time = Column(DateTime)
    def __repr__(self):
        return "<Meeting(mtg_provider='%s', mtg_id='%s', mtg_password='%s')>"
\
            % (self.mtg_provider, self.mtg_id, self.mtg_password)

Base.metadata.create_all(engine)
```

zoom_autojoiner_gui.views

```
# This file is part of Zoom Autojoiner GUI.

# Zoom Autojoiner GUI is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.

# Zoom Autojoiner GUI is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with Zoom Autojoiner GUI.  If not, see
<https://www.gnu.org/licenses/>.

import time
```

```python
import logging
import datetime
import tkinter as tk
import tkinter.font as tkFont
from tkinter import ttk, messagebox
from tkinter import N, S, E, W
from typing import Callable

from zoom_autojoiner_gui.constants import (
    ICON_FILE,
    THEME_FILE,
    DB_URL,
    PYAG_PICS_DIR,
    EXTENSIONS
)
from zoom_autojoiner_gui.controllers import (
    TkinterTheme,
    DatabaseHandler,
    Autojoiner
)
from zoom_autojoiner_gui.dialogs import (
    NewMeetingDialog,
    EditMeetingDialog
)
from zoom_autojoiner_gui.extensions import (
    load_extensions,
    ExtensionHandler
)


logger = logging.getLogger(__name__)


class ApplicationMenuBar(tk.Menu):
    """ApplicationMenuBar

    Creates the menu bar for the application.

    Args:
        root_element: The root MainWindow compatible class.
        meeting_list_frame: The MeetingListFrame compatible class.
    """
    def __init__(self, root_element: tk.Tk,
            meeting_list_frame: tk.Frame = None) -> None:
        # Initialise the TK Menu Bar
        super().__init__(root_element)

        # The Root Element of tk.Tk
        self.root_element = root_element
```

```
        # If the meeting list frame has been supplied, store
        # it, or else leave it as None
        self.__meeting_list_frame = meeting_list_frame

        # Make List to Menu
        try:
            # logger.info("Attepting to render menu bar..")
            self.make_list_to_menu([
                # Application menu
                ["Application", [
                    # ["Clear Data", None, None, None],

                    # Quit the application
                    ["Quit", lambda: root_element.destroy(), "<Control-q>",
                        lambda event: root_element.destroy()],
                    ],
                ],
                # Meetings menu
                ["Meetings", [
                    # Add meeting submenu
                    ["Add Meeting", lambda: self.launch_add_meeting_dialog(),
                        "<Control-n>", lambda event: \
                            self.launch_add_meeting_dialog()],
                    # ["Edit Meeting", None, None, None],
                    # ["Delete Meeting", None, None, None],
                    ],
                ],
                ])
        except Exception as e:
            # If an error occured while rendering the menu bar,
            # put that in the error log and exit.
            logger.error("Failed to render menu bar, exiting...",
                exc_info=True)
            messagebox.showerror("Error", ("An exception has occured.\n"
                "Error Details:\n%s") % (str(e)))
            exit(1)

        # Configure menu to be used by the application.
        root_element.config(menu=self)

    def set_meeting_list_frame(self, frame:tk.Frame) -> None:
        """set_meeting_list_frame

        Sets the frame of the meeting list.

        Args:
            frame:
```

```
            A TK frame which has similar methods to MeetingListFrame.

        Returns:
            Nothing.
        """
        self.__meeting_list_frame = frame

    def make_list_to_menu(self, main_menu: list) -> None:
        """make_list_to_menu

        Makes a list to a usuable menu.

        Args:
            main_menu:
                This list contains the menu to be made in the below
                format.
                ::
                    [
                        ["Main Menu", [
                            ["SubItem", Command, shortcut key, command],
                            ["SubItem2", COmmand, shortcut key],
                            ],
                        ],
                    ]
        """
        for menu_item in main_menu:
            menu = tk.Menu(self, tearoff = "off") # Init menu
            for submenu_item in menu_item[1]:
                # Create subitem
                if submenu_item[2] != None:
                    # Shortcut key - remove angular braces
                    skey = submenu_item[2][1:-1]
                else:
                    skey = None
                menu.add_command(label=submenu_item[0],
                    command=submenu_item[1], accelerator=skey)
                # Shortcut key
                if submenu_item[2] != None:
                    self.root_element.bind_all(submenu_item[2],
                        submenu_item[3])

            # Add menu to app
            self.add_cascade(label=menu_item[0], menu=menu)

    def launch_add_meeting_dialog(self) -> None:
        """launch_add_meeting_dialog

        Launch 'ADD MEETING' dialog
```

```
        """
        # root = tk.Tk()
        # app =
        NewMeetingDialog(tk_root_element = self.root_element,
            tk_frame_handle=self.__meeting_list_frame)
        # root.mainloop()


class MeetingListFrame(tk.Frame):
    """MeetingListFrame

    This class creates the frame that displays
    the list of meetings.

    Args:
        root_element:
            The MainWindow compatible class that is passed to the buttons.
        tk_theme_object:
            The TKTheme object used for styling elements.
        autojoiner_handle:
            The Autojoiner object used for Join
            Meeting buttons.
    """
    __components = []        # TK/TTK widgets
    __current_table_row = 1  # Current row of the table
    def __init__(self, root_element: tk.Tk,
            tk_theme_object: TkinterTheme = None,
            autojoiner_handle: Autojoiner = None) -> None:
        super().__init__(root_element)

        #: We create a Database Handler here.
        self.__dbh = DatabaseHandler(DB_URL)

        self.root_element = root_element #: The root element.

        # If the theme object is not provided, make one, else use the one
        # given
        if tk_theme_object == None:
            self.tk_theme = TkinterTheme(THEME_FILE) #: Tkinter Theme object
        else:
            self.tk_theme = tk_theme_object

        # Autojoiner
        if autojoiner_handle:
            self.__autojoiner_handle = autojoiner_handle
        else:
            self.__autojoiner_handle = Autojoiner(PYAG_PICS_DIR)

        # Sticky grid that resizes according to window size.
```

```
        # tk.Grid.rowconfigure(root_element, row, weight=1)
        # tk.Grid.columnconfigure(root_element, column, weight=1)

        # Grid
        # self.grid(row=row, column=column, sticky=N+S+E+W)

        # Widgets
        # for i in range(0, 10):
        #     for j in range(0, 10):
        #         self.create_ttk_button("Row:%d Column:%d" % (i, j), i, j)
        self.create_column_headers(["Meeting Start Time", "Meeting ID",
                "Meeting Password", "Join Meeting", "Edit/Delete Meeting"])

        # Populate table
        self.populate_table_from_db()

    def __stickify(self, row: int = 0, column: int = 0) -> None:
        """Auto resize the TK widget according to window size

        Args:
            row: The row in the TK grid system
            column: The column in TK grid.
        """
        # Should only stick vertically.
        # tk.Grid.rowconfigure(self, row, weight=1)
        tk.Grid.columnconfigure(self, column, weight=1)

    def create_ttk_button(self, text: str, row: int = 0, column: int = 0,
            command: Callable = None, sticky: str = N+S+E+W,
            stickify: bool = True) -> ttk.Button:
        """create_ttk_button

        Creates a TTK Button and adds resizing capability.

        Args:
            text: Text of the TTK button.
            row: The row in the TK grid system
            column: The column in TK grid.
            command: The command associated with the TK button.
            sticky: TK's sticky attribute
            stickify: Make the width adjust to that of parent container.

        Returns:
            The TTK Button object, for further manipulation.
        """
        # Auto resize
        if stickify: self.__stickify(row, column)
```

```python
        # Create component
        btn = ttk.Button(self, text=text, command=command)
        btn.grid(row=row, column=column, sticky=sticky)

        # Append to component list and return index
        self.__components.append(btn)
        return self.__components[-1]
        # return len(self.__components) - 1

    def create_tk_label(self, text: str, row: int = 0, column: int = 0,
            sticky :str = N+S+E+W, stickify: bool = True,
            *args: tuple, **kwargs: dict) -> tk.Label:
        """create_tk_label

        Creates a TK Label and adds resizing capability.

        Args:
            text: Text of the TK label.
            row: The row in the TK grid system
            column: The column in TK grid.
            command: The command associated with the TK label.
            sticky: TK's sticky attribute
            stickify:
                Make the width adjust to that of parent container.

        Returns:
            The TK Label object, for further manipulation.
        """
        # Auto resize
        if stickify: self.__stickify(row, column)

        # Create component
        lbl = tk.Label(self, text=text, *args, **kwargs)
        lbl.grid(row=row, column=column, sticky=sticky)

        # Append to component list and return index
        self.__components.append(lbl)
        return self.__components[-1]
        # return len(self.__components) - 1

    # Table populating functions:
    def create_column_headers(self, col_headers: list) -> int:
        """create_column_headers

        Creates the column headers.

        Args:
            col_headers: The list of column headers.
```

```
        Returns:
            A integer with the total number of columns.
        """
        col_no = 0
        for col_header in col_headers:
            theme_dict = self.tk_theme.get_styling("table_header")
            self.create_tk_label(col_header, column = col_no, **theme_dict)
            col_no += 1

        return col_no

    def create_table_row(self, record_id: int, meeting_time:
datetime.datetime,
                         meeting_id: str, meeting_password: str) -> None:
        """create_table_row

        Creates a row for the table.

        Args:
            record_id: The ID in database. Used for edit meeting dialog.
            meeting_time: The time of the meeting.
            meeting_id: The ID of the meeting.
            meeting_password: The meeting password.

        Returns:
            Nothing.
        """
        row_no = self.__current_table_row
        styling = self.tk_theme.get_styling("table_content")
        self.create_tk_label(meeting_time.strftime("%a %d %B %Y %I:%M:%S %p"),
            row=row_no, column=0, **styling)
        self.create_tk_label(meeting_id, row=row_no, column=1, **styling)
        self.create_tk_label(meeting_password, row=row_no, column=2,
**styling)
        self.create_ttk_button("Join meeting", row=row_no, column=3,
            command=lambda: self.__autojoiner_handle.join_zm_mtg(meeting_id,
                meeting_password))
        self.create_ttk_button("Edit/Delete meeting", row=row_no, column=4,
            command=lambda: EditMeetingDialog(record_id, tk_root_element = \
                self.root_element, tk_frame_handle=self))
        self.__current_table_row += 1

    # Controller/View Interface
    def populate_table_from_db(self) -> None:
        """populate_table_from_db

        Populate the table from the Database.
```

```
        """
        try:
            # logger.info("Attempting to load meeting data from DB...")
            meetings = self.__dbh.get_mtg_data_to_list()
            for mtg in meetings:
                self.create_table_row(mtg["id"], mtg["mtg_time"],
                    mtg["mtg_id"], mtg["mtg_password"])
        except Exception as e:
            logger.error("Failed to load meeting data, exiting...",
                exc_info=True)
            messagebox.showerror("Error",
                "An exception has occured.\nError Details:\n%s" % (str(e)))
            exit(1)
        else:
            logger.info("Loaded meeting data successfully.")

    def reload_table(self) -> None:
        """reload_table

        Reload and rebuilt the TK Table by calling the
        applicable functions.
        """
        for cell in self.winfo_children():
            cell.destroy()
        self.create_column_headers(["Meeting Start Time", "Meeting ID",
            "Meeting Password", "Join Meeting", "Edit/Delete Meeting"])

        # Populate table
        self.populate_table_from_db()


class ApplicationStatusBar(tk.Label):
    """ApplicationStatusBar

    Status bar is used for iteration tasks, for example, to find out if
    it is time to join the meeting.

    Args:
        root_element: the MainWindow compatible Tk class.
        autojoiner_handle: The Autojoiner class to use.
    """
    def __init__(self, root_element: tk.Tk,
            autojoiner_handle: Autojoiner = None) -> None:
        super().__init__(root_element, text="Loading…", bd=1,
            relief=tk.SUNKEN, anchor=W)

        # Autojoiner
        if autojoiner_handle:
            self.__autojoiner_handle = autojoiner_handle
```

```
        else:
            # Create one if not supplied
            self.__autojoiner_handle = Autojoiner(PYAG_PICS_DIR)

        self.iterator()

    def check_for_meeting(self) -> None:
        """check_for_meeting

        Check for meetings. If there is one now, join.
        Or else just continue.
        """
        if self.__autojoiner_handle.check_for_meeting():
            logger.info("Status Bar - Meeting now.")
            self["text"] = ("There is a meeting now. Zoom Autojoiner is"
                " initiating the joining process.")
            mtg_id = self.__autojoiner_handle.check_for_meeting()["mtg_id"]
            mtg_pw =
self.__autojoiner_handle.check_for_meeting()["mtg_password"]
            self["text"] = ("There is a meeting now. Zoom Autojoiner has "
                "initiated the joining process.")
            logger.info("Status Bar - Joining Meeting")
            self.__autojoiner_handle.join_zm_mtg(mtg_id, mtg_pw)
            self["text"] = ("Zoom Autojoiner has finished the joining "
                "process. There will be a pause for 60 seconds now.")
            logger.info("Status Bar - Sleeping")

            # This line causes the program to hang. However, it prevents
            # the Autojoiner to rejoin the meeting repeatedly.
            time.sleep(60)

    def iterator(self) -> None:
        """iterator

        The iterator checks for meetings every 10 seconds.
        It calls the `check_for_meeting` method to find if
        a meeting is present.
        """
        logger.info("Status Bar - Iterating")
        self["text"] = "Checking for meeting"
        self.check_for_meeting()
        self["text"] = "Running"
        self.after(10000, self.iterator)
        #          ^ Todo - let this value be set in config file.


class MainWindow(tk.Tk):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
```

```
        # Object instances
        self.__tk_theme = TkinterTheme(THEME_FILE)           # TK Styling
object
        self.__autojoiner_handle = Autojoiner(PYAG_PICS_DIR) # Autojoiner
handler

        # Window Titles
        self.title('Zoom Autojoiner')
        try:
            self.iconbitmap(ICON_FILE)
        except:
            logger.warning("Could not load Window icon", exc_info=True)

        # Menu Bar
        self.__menu_bar = ApplicationMenuBar(self)

        # Title
        self.create_tk_label("Zoom AutoJoiner - My Meeting List",
sticky=N+E+W,
            stickify=False, **self.__tk_theme.get_styling("title"))

        # Window Elements
        # Meetings List
        self.__meeting_list_frame = MeetingListFrame(self, self.__tk_theme,
            autojoiner_handle=self.__autojoiner_handle)
        # Elasticity
        tk.Grid.rowconfigure(self, 1, weight=1)
        tk.Grid.columnconfigure(self, 0, weight=1)
        # Positioning
        self.__meeting_list_frame.grid(row=1, column=0, sticky=N+S+E+W)

        # Give menubar the meeting list farme
        self.__menu_bar.set_meeting_list_frame(self.__meeting_list_frame)

        # Statusbar
        self.__statusbar = ApplicationStatusBar(self, autojoiner_handle=
            self.__autojoiner_handle)

        # Elasticity
        # tk.Grid.rowconfigure(self, 2, weight=1)
        # tk.Grid.columnconfigure(self, 0, weight=1)
        # Positioning
        self.__statusbar.grid(row=2, column=0, sticky=N+S+E+W)

        # load extensions
        if EXTENSIONS.getboolean("enabled"):
            # if extensions are enabled
```

```
            self.__ext_class = ExtensionHandler(EXTENSIONS)
            if self.__ext_class.load_extensions():
                self.__ext_class.give_extensions_prefs()
                self.__ext_class.give_extensions_objects(self,
self.__menu_bar,
                    self.__meeting_list_frame)
                self.__ext_class.run_extensions()

    def __stickify(self, row = 0, column = 0):
        """Auto resize the TK widget according to window size"""
        tk.Grid.rowconfigure(self, row, weight=1)
        tk.Grid.columnconfigure(self, column, weight=1)

    def create_tk_label(self, text, row = 0, column = 0, sticky=N+S+E+W,
stickify=True, *args, **kwargs):
        """Creates a TK Label and adds resizing capability."""
        # Auto resize
        if stickify: self.__stickify(row, column)

        # Create component
        lbl = tk.Label(self, text=text, *args, **kwargs)
        lbl.grid(row=row, column=column, sticky=sticky)


if __name__ == "__main__":
    try:
        from ctypes import windll
        windll.shcore.SetProcessDpiAwareness(1) # High DPI
    except:
        pass
    window = MainWindow()
    window.mainloop()


# Todo
# -[] After PyAG autojoin, add 1s refresher to a label
```

zoom_autojoiner_gui.__init__

```
import logging
from datetime import datetime
# from multiprocessing import Process

# I am here because the ZAJ module has some logging work to do..
# Check that my level is set to logging.INFO
logging.basicConfig(filename='logs/%s.log' % (datetime.now().strftime("%Y%m"
    "%d-%H%M%S")), filemode='w', format=('[%(asctime)s] [%(name)s:%(levelna'
    'me)s] [pid:%(process)d, tid:%(thread)d] %(message)s'), datefmt='%c',
    level=logging.INFO)
```

```
from zoom_autojoiner_gui.views import MainWindow
from zoom_autojoiner_gui.extensions import ExtensionHandler
from zoom_autojoiner_gui.constants import EXTENSIONS

logger = logging.getLogger(__name__) # This creates logger for this file.

def load_window():
    """load_window

    Load the Main Window

    Load the main Tk window. The reason this is here
    is because a new process will be started to
    launch the Tk window.
    """
    try:
        # logger.info('Attempting to initialise Window')
        window = MainWindow() # Launch the Main Window.
    except:
        logger.error("Failed to initialise window, exiting...", exc_info=True)
        exit(1)
    else:
        logger.info("Window has been initialised.")

    window.mainloop()


def main():
    """
    Main Function

    Inspired by C/C++ main() function, that looks neat
    """
    try:
        # logger.info('Attempting to initialise High DPI awareness') # Don't
clutter the log
        from ctypes import windll
        windll.shcore.SetProcessDpiAwareness(1) # High DPI
    except:
        logger.warning('Windows - Failed to enable High DPI awareness.'
            ' Maybe not on Windows?')
    else:
        logger.info('Windows - High DPI awareness Enabled')

    try:
        # logger.info("Starting child process..")
        # tk_proc = Process(target=load_window, args=())
        # tk_proc.start()
```

```
        # tk_proc.join()
        load_window()
    except:
        pass
        # logger.critical("Failed to start child process!!!", exc_info=True)

if __name__ == "__main__":
    main()
```

Screenshots and Videos

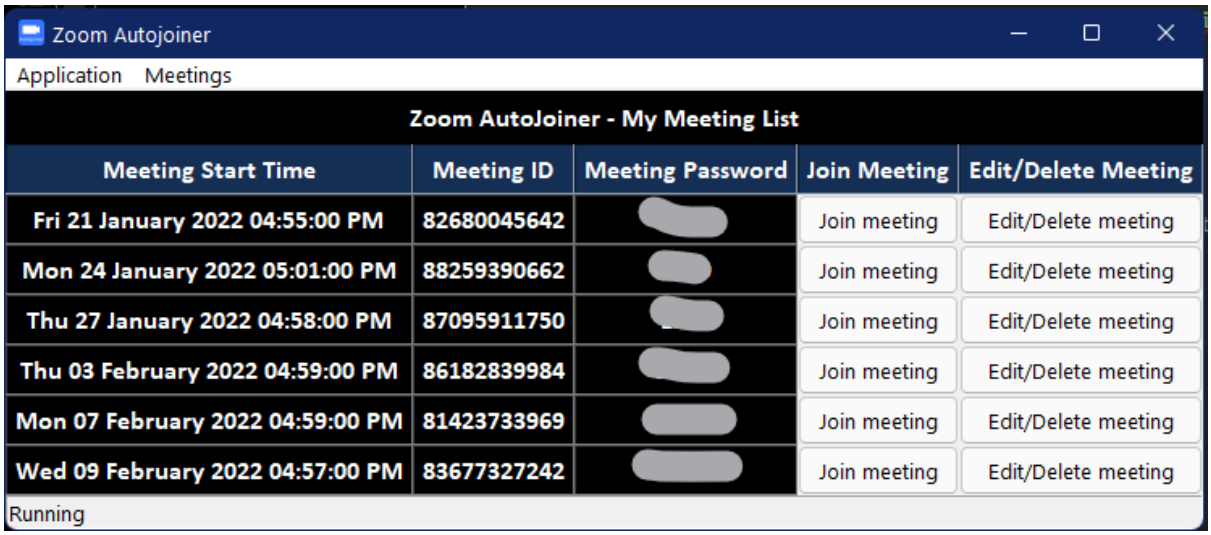
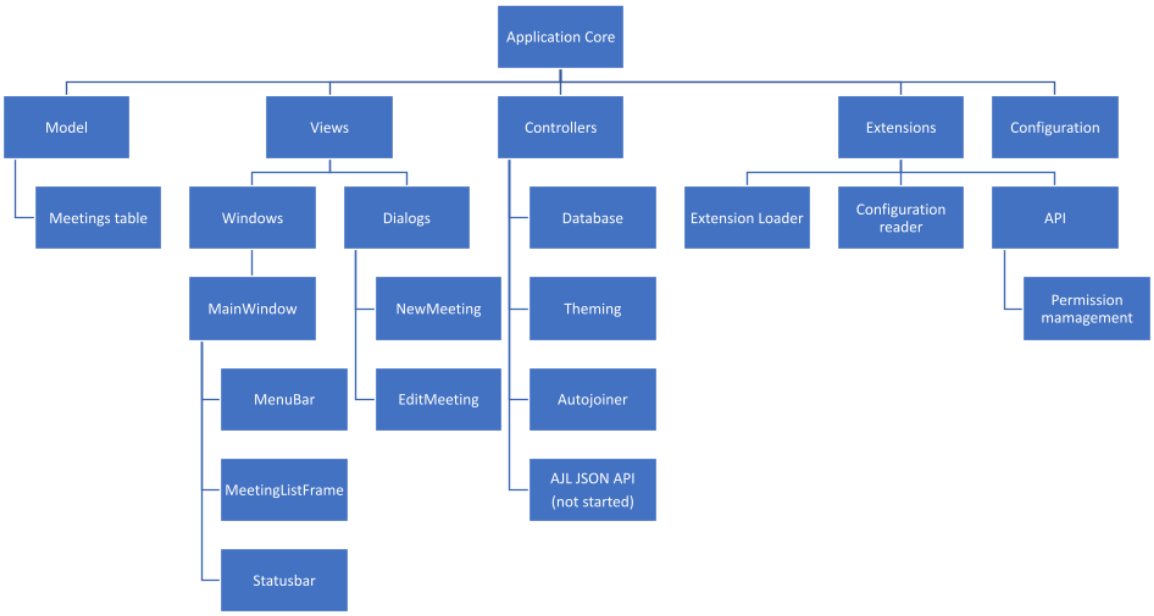


Figure 1 A screenshot of the Autojoiner Home Page

Link to Demo: https://youtu.be/BIxQBrfZCpQ

Appendix

Application Flow Diagram



Network Notice Board

1. Run *pip install pymysql*
2. Change the *SQLALCHEMY_DB_URI* to
   mysql+pymysql://<username>:<password>@<IP>/<database>
   Where,

Field	Value
Username	MySQL username
Password	MySQL password
IP	IP address
Database	Name of DB in MySQL. Please create one.