main

▸ View Source

class MainApp(PySide6.QtWidgets.QMainWindow): ▸ View Source

Główna klasa aplikacji ClariSpace.

Zarządza głównym oknem aplikacji, inicjalizuje zakładki oraz integrację z Google Calendar.

MainApp() ▸ View Source

Inicjalizuje główne okno aplikacji, tworzy zakładki i ładuje arkusz stylów.

tab_widget

google_calendar

focus_timer_tab

todo_calendar_tab

notes_tab

def load_stylesheet(self, filepath): ▸ View Source

Wczytuje arkusz stylów z pliku i stosuje go do aplikacji.

Parameters

- **filepath**: Ścieżka do pliku ze stylami (QSS).

staticMetaObject = PySide6.QtCore.QMetaObject("MainApp" inherits "QMainWindow":)

notes

▸ View Source

class NotesInterface: ▸ View Source

Interfejs definiujący metody do zarządzania notatkami.

def load_notes(self) -> List[Any]: ▸ View Source

Ładowanie notatek.

Returns: List[Any]: Lista notatek.

def save_notes(self) -> **None**: ▸ View Source

Zapisanie notatek do pliku.

def save_note(self) -> **None**: ▸ View Source

Zapisanie pojedynczej notatki.

class NoteSortStrategy: ▸ View Source

Interfejs strategii sortowania notatek.

def sort(self, notes: List[Note]) -> List[Note]: ▸ View Source

Sortowanie notatek.

Args: notes (List[Note]): Lista notatek do posortowania.

Returns: List[Note]: Posortowana lista notatek.

class SortByDate(NoteSortStrategy): ▸ View Source

Strategia sortowania notatek po dacie.

def sort(self, notes: List[Note]) -> List[Note]: ▸ View Source

Sortowanie notatek po dacie.

Args: notes (List[Note]): Lista notatek do posortowania.

Returns: List[Note]: Posortowana lista notatek.

class SortByTitle(NoteSortStrategy): ▸ View Source

Strategia sortowania notatek po tytule.

def sort(self, notes: List[Note]) -> List[Note]: ▸ View Source

Sortowanie notatek po tytule.

Args: notes (List[Note]): Lista notatek do posortowania.

Returns: List[Note]: Posortowana lista notatek.

class SortByCategory(NoteSortStrategy): ▸ View Source

Strategia sortowania notatek po kategorii.

def sort(self, notes: List[Note]) -> List[Note]: ▸ View Source

Sortowanie notatek po kategorii.

Args: notes (List[Note]): Lista notatek do posortowania.

Returns: List[Note]: Posortowana lista notatek.

class SimpleObfuscator: ▸ View Source

Klasa do obfuskacji tekstu przy użyciu XOR.

SimpleObfuscator(key: int) ▸ View Source

Inicjalizuje obfuskator z kluczem.

Args: key (int): Klucz do obfuskacji.

key

def obfuscate(self, text: str) -> str: ▸ View Source

Obfuskacja tekstu.

Args: text (str): Tekst do obfuskacji.

Returns: str: Obfuskowany tekst.

def deobfuscate(self, text: str) -> str: ▸ View Source

Deobfuskacja tekstu.

Args: text (str): Obfuskowany tekst.

Returns: str: Oryginalny tekst.

class Note: ▸ View Source

Klasa reprezentująca pojedynczą notatkę.

Note(title: str, content: str, category: str, date: str) ▸ View Source

Inicjalizuje notatkę z tytułem, treścią, kategorią i datą.

Args: title (str): Tytuł notatki. content (str): Treść notatki. category (str): Kategoria notatki. date (str): Data utworzenia notatki.

title

content

category

date

def save_data(self) -> Dict[str, str]: ▸ View Source

Zwraca dane notatki w formie słownika.

Returns: Dict[str, str]: Słownik z danymi notatki.

class NoteFactory: ▸ View Source

Fabryka do tworzenia notatek.

▸ View Source
@staticmethod

def **create_note**(category: str, title: str, content: str) -> Note:

Tworzy nową notatkę z kategorią, tytułem i treścią.

Args: category (str): Kategoria notatki. title (str): Tytuł notatki. content (str): Treść notatki.

Returns: Note: Nowa instancja notatki.

class **Notes**(PySide6.QtWidgets.QWidget, NotesInterface): ▸ View Source

Klasa reprezentująca główny interfejs użytkownika dla notatek.

Notes() ▸ View Source

Inicjalizuje główny widok notatek.

json_file: str

notes: List[Note]

sort_strategy: NoteSortStrategy

note_title

note_content

note_category

sort_combo

save_button

notes_list

def **change_sort_strategy**(self, strategy_name: str) -> **None**: ▸ View Source

Zmiana strategii sortowania w zależności od wyboru użytkownika.

Args: strategy_name (str): Nazwa wybranej strategii sortowania.

def **apply_sort**(self) -> **None**: ▸ View Source

Zastosowanie strategii sortowania.

def load_notes(self) -> List[Note]: ▸ View Source

Ładowanie notatek z pliku JSON.

Returns: List[Note]: Lista załadowanych notatek.

def create_note_from_data(self, data: Dict[str, str]) -> Note: ▸ View Source

Tworzenie obiektu notatki z danych z JSON.

Args: data (Dict[str, str]): Słownik z danymi notatki.

Returns: Note: Obiekt notatki.

def save_notes(self) -> None: ▸ View Source

Zapisanie notatek do pliku JSON.

def save_note(self) -> None: ▸ View Source

Zapisanie jednej notatki.

def refresh_notes_list(self) -> None: ▸ View Source

Odświeżenie listy notatek wyświetlanych w GUI.

def load_note(self, item: PySide6.QtWidgets.QListWidgetItem) -> None: ▸ View Source

Ładowanie notatki po kliknięciu w liście.

Args: item (QListWidgetItem): Element listy, który został kliknięty.

staticMetaObject = PySide6.QtCore.QMetaObject("Notes" inherits "QWidget":)

class ObfuscatedNotesDecorator(NotesInterface): ▸ View Source

Dekorator do obfuskacji notatek.

ObfuscatedNotesDecorator(notes_component: Notes, obfuscator: SimpleObfuscator)
▸ View Source

Inicjalizuje dekorator obfuskacji notatek.

Args: notes_component (Notes): Instancja komponentu notatek. obfuscator (SimpleObfuscator): Instancja obfuskatora.

widget

def load_notes(self) -> List[Note]: ▸ View Source

Ładowanie obfuskowanych notatek z pliku.

Returns: List[Note]: Lista załadowanych notatek.

def save_notes(self) -> **None**: ▸ View Source

Zapisanie obfuskowanych notatek do pliku.

def save_note(self) -> **None**: ▸ View Source

Zapisanie notatki z obfuskacją.

focus_timer

▸ View Source

class TimerState: ▸ View Source

Baza dla różnych stanów timera.

def start_timer(self, timer): ▸ View Source

Uruchamia timer.

Args: timer (FocusTimer): Obiekt timera, który ma być uruchomiony.

def stop_timer(self, timer): ▸ View Source

Zatrzymuje timer.

Args: timer (FocusTimer): Obiekt timera, który ma być zatrzymany.

def reset_timer(self, timer): ▸ View Source

Resetuje timer.

Args: timer (FocusTimer): Obiekt timera, który ma być zresetowany.

def update_timer(self, timer): ▸ View Source

Aktualizuje timer na podstawie upływającego czasu.

Args: timer (FocusTimer): Obiekt timera, który ma być aktualizowany.

class IdleState(TimerState): ▸ View Source

Stan kiedy timer jest wstrzymany (bez aktywności).

def start_timer(self, timer): ▸ View Source

Rozpoczyna odliczanie timera.

Args: timer (FocusTimer): Obiekt timera, który ma być uruchomiony.

Inherited Members

TimerState stop_timer(), reset_timer(), update_timer()

class RunningState(TimerState): ▸ View Source

Stan kiedy timer odlicza czas.

def stop_timer(self, timer): ▸ View Source

Zatrzymuje timer i przechodzi do stanu idle.

Args: timer (FocusTimer): Obiekt timera, który ma być zatrzymany.

def reset_timer(self, timer): ▸ View Source

Resetuje timer i przechodzi do stanu idle.

Args: timer (FocusTimer): Obiekt timera, który ma być zresetowany.

def update_timer(self, timer): ▸ View Source

Aktualizuje wyświetlany czas na podstawie upływającego czasu.

Args: timer (FocusTimer): Obiekt timera, który ma być aktualizowany.

Inherited Members

TimerState start_timer()

class CompletedState(TimerState): ▸ View Source

Stan kiedy timer zakończył odliczanie.

def start_timer(self, timer): ▸ View Source

Uruchamia timer ponownie z początkowego stanu.

Args: timer (FocusTimer): Obiekt timera, który ma być uruchomiony.

Inherited Members

TimerState stop_timer(), reset_timer(), update_timer()

class FocusTimer(PySide6.QtWidgets.QWidget): ▸ View Source

Główna klasa odpowiedzialna za zarządzanie timerem i interfejsem użytkownika.

FocusTimer() ▸ View Source

Inicjalizuje aplikację i stan timera.

timer

time_elapsed

is_running

start_time

points

activity

activities

current_state

def initUI(self): ▸ View Source

Tworzy interfejs użytkownika.

def load_stylesheet(self): ▸ View Source

Ładuje arkusz stylów CSS z pliku.

def add_time_button(self, layout, label, minutes): ▸ View Source

Dodaje przycisk ustawiający czas.

Args: layout (QHBoxLayout): Layout, do którego dodawany jest przycisk. label (str): Etykieta przycisku. minutes (int): Liczba minut do ustawienia.

def set_activity(self, activity): ▸ View Source

Ustawia aktywność na podstawie wyboru użytkownika.

Args: activity (str): Nazwa aktywności wybranej przez użytkownika.

def set_timer(self, minutes): ▸ View Source

Ustawia czas na podstawie wybranego minutowego okresu.

Args: minutes (int): Liczba minut do ustawienia timera.

def start_timer(self): ▸ View Source

Uruchamia timer w odpowiednim stanie.

def stop_timer(self): ▸ View Source

Zatrzymuje timer w odpowiednim stanie.

def reset_timer(self): ▸ View Source

Resetuje timer w odpowiednim stanie.

def update_timer(self): ▸ View Source

Aktualizuje stan timera w zależności od czasu.

def confirm_stop_timer(self): ▸ View Source

Potwierdzenie zatrzymania timera.

def confirm_reset_timer(self): ▸ View Source

Potwierdzenie resetowania timera.

def add_points(self): ▸ View Source

Dodaje punkty po zakończeniu timerem.

def save_points(self): ▸ View Source

Zapisuje punkty do pliku JSON.

def load_points(self): ▸ View Source

Ładuje punkty z pliku JSON.

def update_stats(self): ▸ View Source

Aktualizuje dane statystyczne o czasie spędzonym na aktywności.

def show_stats(self): ▸ View Source

Pokazuje czas spędzony na aktywności.

def show_completion_message(self): ▸ View Source

Pokazuje komunikat o zakończeniu timera.

def update_activities(self): ▸ View Source

Aktualizuje czas spędzony na danej aktywności.

def save_activities(self): ▸ View Source

Zapisuje dane o aktywnościach do pliku JSON.

def load_activities(self): ▸ View Source

Ładuje dane o aktywnościach z pliku JSON.

def update_pie_chart(self): ▸ View Source

Aktualizuje wykres kołowy na podstawie danych o aktywnościach.

staticMetaObject = PySide6.QtCore.QMetaObject("FocusTimer" inherits "QWidget":)

todocalendar

▸ View Source

class SimpleObfuscator: ▸ View Source

Klasa do prostej obfuskacji tekstu przy użyciu operacji XOR.

Parameters

- **key**: Klucz do obfuskacji/deobfuskacji.

SimpleObfuscator(key: int) ▸ View Source

key

def obfuscate(self, text: str) -> str: ▸ View Source

Obfuskacja tekstu.

Parameters

- **text**: Tekst do obfuskacji.

Returns

Obfuskowany tekst.

def deobfuscate(self, text: str) -> str: ▸ View Source

Deobfuskacja tekstu.

Parameters

- **text**: Obfuskowany tekst.

Returns

Oryginalny tekst.

class ToDoCalendar(PySide6.QtWidgets.QWidget): ▸ View Source

Główne okno aplikacji do zarządzania zadaniami z kalendarzem.

ToDoCalendar() ▸ View Source

Inicjalizacja interfejsu użytkownika i załadowanie zapisanych zadań.

google_calendar

obfuscator

json_file

tasks_by_date

current_date

date_label

calendar

task_input

add_task_button

delete_task_button

task_list

def load_tasks(self): ▸ View Source

Wczytuje zadania z pliku JSON i deobfuskuje je.

Returns

Słownik z zadaniami.

def save_tasks(self): ▸ View Source

Zapisuje zadania do pliku JSON po ich obfuskacji.

def display_tasks_for_date(self, date): ▸ View Source

Wyświetla zadania dla wybranej daty.

Parameters

- **date**: Wybrana data w formacie "dd-MM-yyyy".

def add_task_to_list_widget(self, task_text): ▸ View Source

Dodaje zadanie do listy zadań w GUI.

Parameters

- **task_text**: Tekst zadania.

def add_task(self): ▸ View Source

Dodaje nowe zadanie i zapisuje je do pliku oraz Google Calendar.

def delete_task(self): ▸ View Source

Usuwa zaznaczone zadania.

def change_date(self, date): ▸ View Source

Zmienia aktualnie wybraną datę i wyświetla przypisane do niej zadania.

staticMetaObject = PySide6.QtCore.QMetaObject("ToDoCalendar" inherits "QWidget":)

google_calendar_adapter

▸ View Source

SCOPES = ['https://www.googleapis.com/auth/calendar']

class GoogleCalendarAdapter: ▸ View Source

GoogleCalendarAdapter() ▸ View Source

Inicjalizuje adapter do Google Calendar API, uzyskując poświadczenia i tworzy usługę API do interakcji z kalendarzem.

creds

service

def get_credentials(self): ▸ View Source

Uzyskuje poświadczenia użytkownika dla Google Calendar API. Jeśli poświadczenia istnieją, ale są nieważne, są odświeżane. W przeciwnym razie użytkownik jest proszony o autoryzację.

Zwraca: creds (Credentials): Poświadczenia użytkownika do API

def add_event(self, title, date): ▸ View Source

Dodaje wydarzenie do Google Calendar.

Parametry: title (str): Tytuł wydarzenia date (str): Data wydarzenia w formacie YYYY-MM-DD

Zwraca: None