

Nr	Obszar	Wymaganie	KOD		Przyznane pkt	Pkt max
1	UI	JEST	<pre> main.py: def main(): # Zmienna lokalna (przykład) data_manager = DataManager("data/work_log.json") analyzer = Analizer() while True: print_header() print_menu() choice = input("\nWybierz opcję: ").strip() if choice == "1": add_entry(data_manager) elif choice == "2": print_entries(data_manager.entries) elif choice == "3": edit_entry(data_manager) elif choice == "4": print("\n--- Wyszukiwanie wpisów po dacie ---") search_date = input("Podaj datę do wyszukania (YYYY-MM-DD): ").strip() found = list(filter(lambda e: e.date == search_date, data_manager.entries)) print_entries(found) elif choice == "5": analyzer.analyze(data_manager.entries) elif choice == "6": try: data_manager.save() print("✅ Dane zapisane.") except Exception as e: print(f"Błąd zapisu: {e}") elif choice == "7": analyzer.plot(data_manager.entries) elif choice == "8": remove_entry(data_manager) elif choice == "9": total = recursive_sum(data_manager.entries) print(f"\n1234 Rekurencyjna suma godzin: {total:.2f}h") elif choice == "10": print("\n--- Filtruj wpisy ---") </pre>	☑		

			<pre> try: min_hours = float(input("Pokaż wpisy z czasem pracy >= (godz): ")) filtered = filter_entries(data_manager.entries, lambda e: e.duration() >= min_hours) print_entries(filtered) except ValueError: print("✗ Podano niepoprawną wartość.") elif choice == "11": string_operations_demo() elif choice == "12": filename = input("Podaj nazwę pliku CSV do eksportu: ").strip() export_to_csv(data_manager.entries, filename) elif choice == "13": filename = input("Podaj nazwę pliku CSV do importu: ").strip() import_from_csv(data_manager, filename) elif choice == "0": print("\nDziękujemy za skorzystanie z systemu. Do zobaczenia!") sys.exit(0) else: print("✗ Niepoprawna opcja. Spróbuj ponownie.") </pre>			
		Wprowadzanie danych	<pre> main.py: def add_entry(data_manager): """Dodaje nowy wpis czasu pracy z walidacją i atrakcyjnym komunikatem.""" print("\n--- Dodawanie nowego wpisu ---") date = input("Podaj datę (YYYY-MM-DD): ").strip() assert len(date) == 10, "Data powinna mieć 10 znaków (YYYY-MM-DD)" if not validate_date(date): print("✗ Niepoprawna data!") return start = input("Godzina rozpoczęcia (HH:MM): ").strip() if not validate_time(start): print("✗ Niepoprawny format godziny rozpoczęcia!") return end = input("Godzina zakończenia (HH:MM): ").strip() if not validate_time(end): print("✗ Niepoprawny format </pre>	☑		2

		<pre>godziny zakończenia!") return entry = WorkEntry(date, start, end) data_manager.entries.append(entry) print("✅ Dodano wpis.")</pre>			
	Wyświetlanie danych	<pre>main.py: def print_entries(entries): """Ładnie wyświetla wszystkie wpisy.""" if not entries: print("Brak wpisów do wyświetlenia.") return print("\n--- Lista wpisów ---") for i, e in enumerate(entries, 1): print(f"{str(i).rjust(2)}. {e}")</pre>	☑		2
	Zmiana danych	<pre>main.py: def edit_entry(data_manager): """Edycja wybranego wpisu czasu pracy.""" if not data_manager.entries: print("Brak wpisów do edycji.") return print("\n--- Edycja wpisu ---") for i, e in enumerate(data_manager.entries): print(f"{i + 1}. {e}") try: index = int(input("Podaj numer wpisu do edycji: ")) - 1 if 0 <= index < len(data_manager.entries): entry = data_manager.entries[index] print(f"Edytujesz: {entry}") new_start = input("Nowa godzina rozpoczęcia (HH:MM): ").strip() if not validate_time(new_start): print("❌ Niepoprawny format godziny!") return new_end = input("Nowa godzina zakończenia (HH:MM): ").strip() if not validate_time(new_end): print("❌ Niepoprawny format godziny!") return entry.start = new_start entry.end = new_end print("✅ Zmieniono wpis.") else: print("❌ Niepoprawny</pre>	☑		2

		<pre>indeks.") except Exception as e: print(f"Błąd edycji: {e}")</pre>			
	Wyszukiwanie danych	<pre>main.py: def filter_entries(entries, predicate): """Zwraca listę wpisów spełniających warunek predicate (funkcja jako argument).""" return [e for e in entries if predicate(e)]</pre>	<input checked="" type="checkbox"/>		2
	Przedstawienie wyników	<pre>analyzer.py: class Analizer: """Analizuje dane o czasie pracy.""" def analyze(self, entries): """Wyświetla statystyki i analizę czasu pracy.""" if not entries: print("Brak danych do analizy.") return # reduce - suma godzin (programowanie funkcyjne) total = reduce(lambda acc, e: acc + e.duration(), entries, 0) avg = total / len(entries) print("\nStatystyki czasu pracy:") print(f" - Łączny czas pracy: {total:.2f}h") print(f" - Średni czas na wpis: {avg:.2f}h") # słownik miesięcy, krotki, zbiór miesięcy (przykład użycia kontenerów) monthly = defaultdict(float) for e in entries: monthly[e.date[:7]] += e.duration() print("\nCzas pracy wg miesięcy:") for month, hours in monthly.items(): tup = (month, hours) # krotka print(f" - {tup[0]}: {tup[1]:.2f}h") def plot(self, entries): """Generuje i zapisuje wykres czasu pracy.""" if not entries: print("Brak danych do wykresu.")</pre>	<input checked="" type="checkbox"/>		2

			<pre> return # Słownik z sumą godzin dla każdej daty daily = defaultdict(float) for e in entries: daily[e.date] += e.duration() dates = sorted(daily.keys()) hours = [daily[d] for d in dates] # Tworzenie wykresu słupkowego plt.figure(figsize=(10, 5)) plt.bar(dates, hours, color='skyblue') plt.xlabel("Data") plt.ylabel("Godziny pracy") plt.title("Czas pracy dzienny") plt.xticks(rotation=45) plt.tight_layout() plt.savefig("data/work_plot.png") plt.show() print("Wykres zapisano jako data/work_plot.png") </pre>			
2	Podstawy	Zmienne	main.py: GLOBAL_USER = "admin" , zmienne lokalne w funkcjach	☑		2
		typy danych	entry.py: str, float, dict, list, set, tuple	☑		2
		komentarze	main.py, entry.py: # Zmienna globalna (przykład użycia zmiennej globalnej) """Wyświetla nagłówek aplikacji w atrakcyjny sposób.""" """Wyświetla menu główne.""" """Dodaje nowy wpis czasu pracy z walidacją i atrakcyjnym komunikatem.""" """Rekurencyjnie sumuje czas pracy ze wszystkich wpisów.""" """Zwraca listę wpisów spełniających warunek predicate (funkcja jako argument).""" """Usuwa wpis czasu pracy na podstawie numeru.""" """Edycja wybranego wpisu czasu pracy.""" itd.	☑		1
		operatory	main.py: operatory arytmetyczne, logiczne	☑		1,5
		Instrukcje warunkowe (if, elif, else)	main.py: add_entry(), main() if choice == "1": add_entry(data_manager) elif choice == "2": print_entries(data_manager.entries)	☑		3

```

elif choice == "3":
    edit_entry(data_manager)
elif choice == "4":
    print("\n--- Wyszukiwanie wpisów po
dacie ---")
    search_date = input(
        "Podaj datę do wyszukania
(YYYY-MM-DD): ").strip()
    found = list(filter(lambda e:
e.date ==
                        search_date,
data_manager.entries))
    print_entries(found)
elif choice == "5":
    analyzer.analyze(data_manager.entries)
elif choice == "6":
    try:
        data_manager.save()
        print("✅ Dane zapisane.")
    except Exception as e:
        print(f"Błąd zapisu: {e}")
elif choice == "7":
    analyzer.plot(data_manager.entries)
elif choice == "8":
    remove_entry(data_manager)
elif choice == "9":
    total =
recursive_sum(data_manager.entries)
    print(f"\n1234 Rekurencyjna suma
godzin: {total:.2f}h")
elif choice == "10":
    print("\n--- Filtruj wpisy ---")
    try:
        min_hours = float(
            input("Pokaż wpisy z czasem
pracy >= (godz): "))
        filtered = filter_entries(
            data_manager.entries,
lambda e: e.duration() >= min_hours)
        print_entries(filtered)
    except ValueError:
        print("❌ Podano niepoprawną
wartość.")
elif choice == "11":
    string_operations_demo()
elif choice == "12":
    filename = input("Podaj nazwę pliku
CSV do eksportu: ").strip()
    export_to_csv(data_manager.entries,
filename)
elif choice == "13":
    filename = input("Podaj nazwę pliku
CSV do importu: ").strip()
    import_from_csv(data_manager,
filename)
elif choice == "0":
    print("\nDziękujemy za skorzystanie
z systemu. Do zobaczenia!")
    sys.exit(0)
else:

```

			<pre>print("❌ Niepoprawna opcja. Spróbuj ponownie.")</pre>			
	Instrukcje iteracyjne	main.py: for w print_entries(), while w main()				
	for	main.py: for w print_entries() <pre>for i, e in enumerate(entries, 1): print(f"{str(i).rjust(2)}. {e}")</pre>	☑		2	
	while	main.py: while w main() <pre>while True: print_header() print_menu() choice = input("\nWybierz opcję: ").strip() if choice == "1": add_entry(data_manager) elif choice == "2": print_entries(data_manager.entries) elif choice == "3": edit_entry(data_manager) elif choice == "4": print("\n--- Wyszukiwanie wpisów po dacie ---") search_date = input("Podaj datę do wyszukania (YYYY-MM-DD): ").strip() found = list(filter(lambda e: e.date == search_date, data_manager.entries)) print_entries(found) elif choice == "5": analyzer.analyze(data_manager.entries) elif choice == "6": try: data_manager.save() print("✅ Dane zapisane.") except Exception as e: print(f"Błąd zapisu: {e}") elif choice == "7": analyzer.plot(data_manager.entries) elif choice == "8": remove_entry(data_manager) elif choice == "9": total = recursive_sum(data_manager.entries) print(f"\n1234 Rekurencyjna suma godzin: {total:.2f}h") elif choice == "10": print("\n--- Filtruj wpisy ---") try: min_hours = float(input("Pokaż wpisy z czasem pracy >= (godz): ")) filtered = filter_entries(</pre>	☑		2	

			<pre> data_manager.entries, lambda e: e.duration() >= min_hours) print_entries(filtered) except ValueError: print("✗ Podano niepoprawną wartość.") elif choice == "11": string_operations_demo() elif choice == "12": filename = input("Podaj nazwę pliku CSV do eksportu: ").strip() export_to_csv(data_manager.entries, filename) elif choice == "13": filename = input("Podaj nazwę pliku CSV do importu: ").strip() import_from_csv(data_manager, filename) elif choice == "0": print("\nDziękujemy za skorzystanie z systemu. Do zobaczenia!") sys.exit(0) else: print("✗ Niepoprawna opcja. Spróbuj ponownie.") </pre>			
		Operacje wejścia (input)	<pre> main.py: input() w UI print_header() print_menu() choice = input("\nWybierz opcję: ").strip() </pre>	☑		1,5
		Operacje wyjścia (print)	<pre> main.py: print() w UI print_header() print_menu() choice = input("\nWybierz opcję: ").strip() </pre>	☑		1,5
		Funkcje z parametrami i wartościami zwracanymi	<pre> main.py: add_entry(), recursive_sum(), entry.py: duration() def add_entry(data_manager): def recursive_sum(entries, idx=0): def filter_entries(entries, predicate): def remove_entry(data_manager): def edit_entry(data_manager): </pre>	☑		2
		Funkcje rekurencyjne	<pre> main.py: recursive_sum() def recursive_sum(entries, idx=0): """Rekurencyjnie sumuje czas pracy ze wszystkich wpisów.""" if idx >= len(entries): return 0 return entries[idx].duration() + recursive_sum(entries, idx + 1) </pre>	☑		3

		Funkcje przyjmujące inne funkcje jako argumenty	<pre> main.py: filter_entries() def filter_entries(entries, predicate): """Zwraca listę wpisów spełniających warunek predicate (funkcja jako argument).""" return [e for e in entries if predicate(e)] </pre>	☑		3
		Dekoratory	<pre> validators.py: log_operation # utils/validators.py from datetime import datetime import functools def log_operation(func): """Dekorator logujący operacje na wpisach.""" @functools.wraps(func) def wrapper(*args, **kwargs): print(f"[LOG] Wywołano: {func.__name__}") return func(*args, **kwargs) return wrapper def validate_date(date_str): """Waliduj datę w formacie YYYY-MM-DD.""" try: datetime.strptime(date_str, "%Y-%m-%d") return True except ValueError: return False def validate_time(time_str): """Waliduj czas w formacie HH:MM.""" try: datetime.strptime(time_str, "%H:%M") return True except ValueError: return False </pre>	☑		1,5
3	Kontenery	Użycie listy	<pre> main.py: entries, data_manager.entries def main(): # Zmienna lokalna (przykład) data_manager = DataManager("data/work_log.json") analyzer = Analyzer() while True: print_header() print_menu() choice = input("\nWybierz opcję: ").strip() if choice == "1": add_entry(data_manager) elif choice == "2": </pre>	☑		2

```

print_entries(data_manager.entries)
    elif choice == "3":
        edit_entry(data_manager)
    elif choice == "4":
        print("\n--- Wyszukiwanie
wpisów po dacie ---")
        search_date = input(
            "Podaj datę do
wyszukania (YYYY-MM-DD): ").strip()
        found = list(filter(lambda
e: e.date ==
                                search_date,
data_manager.entries))
        print_entries(found)
    elif choice == "5":

analyzer.analyze(data_manager.entries)
    elif choice == "6":
        try:
            data_manager.save()
            print("✅ Dane
zapisane.")
        except Exception as e:
            print(f"Błąd zapisu:
{e}")
    elif choice == "7":

analyzer.plot(data_manager.entries)
    elif choice == "8":
        remove_entry(data_manager)
    elif choice == "9":
        total =
recursive_sum(data_manager.entries)
        print(f"\n12/34 Rekurencyjna
suma godzin: {total:.2f}h")
    elif choice == "10":
        print("\n--- Filtruj wpisy
---")
        try:
            min_hours = float(
                input("Pokaż wpisy
z czasem pracy >= (godz): "))
            filtered =
filter_entries(
data_manager.entries, lambda e:
e.duration() >= min_hours)
            print_entries(filtered)
        except ValueError:
            print("❌ Podano
niepoprawną wartość.")
    elif choice == "11":
        string_operations_demo()
    elif choice == "12":
        filename = input("Podaj
nazwę pliku CSV do eksportu:
").strip()

export_to_csv(data_manager.entries,

```

		<pre> filename) elif choice == "13": filename = input("Podaj nazwę pliku CSV do importu: ").strip() import_from_csv(data_manager, filename) elif choice == "0": print("\nDziękujemy za skorzystanie z systemu. Do zobaczenia!") sys.exit(0) else: print("✗ Niepoprawna opcja. Spróbuj ponownie.") </pre>			
	Użycie słownika	<pre> analyzer.py: monthly, daily monthly = defaultdict(float) for e in entries: monthly[e.date[:7]] += e.duration() print("\nCzas pracy wg miesięcy:") for month, hours in monthly.items(): tup = (month, hours) # krotka print(f" - {tup[0]}: {tup[1]:.2f}h") # Słownik z sumą godzin dla każdej daty daily = defaultdict(float) for e in entries: daily[e.date] += e.duration() dates = sorted(daily.keys()) hours = [daily[d] for d in dates] </pre>	☑		2
	Użycie zbioru	<pre> analyzer.py: set(monthly.keys()) # services/analyzer.py from collections import defaultdict import matplotlib.pyplot as plt from functools import reduce class Analyzer: """Analizuje dane o czasie pracy.""" def analyze(self, entries): """Wyświetla statystyki i analizę czasu pracy.""" if not entries: print("Brak danych do analizy.") return # reduce - suma godzin (programowanie funkcyjne) total = reduce(lambda acc, e: acc + e.duration(), entries, 0) avg = total / len(entries) print("\nStatystyki czasu </pre>	☑		1,5

```

pracy:")
    print(f" - Łączny czas pracy:
{total:.2f}h")
    print(f" - Średni czas na wpis:
{avg:.2f}h")

    # słownik miesięcy, krotki,
zbiór miesięcy (przykład użycia
kontenerów)
    monthly = defaultdict(float)
    for e in entries:
        monthly[e.date[:7]] +=
e.duration()

    print("\nCzas pracy wg
miesięcy:")
    for month, hours in
monthly.items():
        tup = (month, hours) #
krotka
        print(f" - {tup[0]}:
{tup[1]:.2f}h")

    def plot(self, entries):
        """Generuje i zapisuje wykres
czasu pracy."""
        if not entries:
            print("Brak danych do
wykresu.")
            return

        # Słownik z sumą godzin dla
każdej daty
        daily = defaultdict(float)
        for e in entries:
            daily[e.date] +=
e.duration()

        dates = sorted(daily.keys())
        hours = [daily[d] for d in
dates]

        # Tworzenie wykresu słupkowego
plt.figure(figsize=(10, 5))
plt.bar(dates, hours,
color='skyblue')
plt.xlabel("Data")
plt.ylabel("Godziny pracy")
plt.title("Czas pracy dzienny")
plt.xticks(rotation=45)
plt.tight_layout()

plt.savefig("data/work_plot.png")
plt.show()
print("Wykres zapisano jako
data/work_plot.png")

```

		Użycie krotki	<pre> analyzer.py: tup = (month, hours) tup = (month, hours) # krotka print(f" - {tup[0]}: {tup[1]:.2f}h") </pre>	<input checked="" type="checkbox"/>		1,5
4	Przestrzenie nazw	Zastosowano zmienne lokalne	<pre> main.py: # Zmienna lokalna (przykład) data_manager = DataManager("data/work_log.json") analyzer = Analyzer() </pre>	<input checked="" type="checkbox"/>		1,5
		Zastosowano zmienne globalne	<pre> main.py: GLOBAL USER # Zmienna globalna (przykład użycia zmiennej globalnej) GLOBAL_USER = "admin" </pre>	<input checked="" type="checkbox"/>		1,5
		Zastosowano zakresy funkcji	<pre> main.py: funkcje z parametrami def add_entry(data_manager): """Dodaje nowy wpis czasu pracy z walidacją i atrakcyjnym komunikatem.""" print("\n--- Dodawanie nowego wpisu ---") date = input("Podaj datę (YYYY-MM-DD): ").strip() assert len(date) == 10, "Data powinna mieć 10 znaków (YYYY-MM-DD)" if not validate_date(date): print("✗ Niepoprawna data!") return start = input("Godzina rozpoczęcia (HH:MM): ").strip() if not validate_time(start): print("✗ Niepoprawny format godziny rozpoczęcia!") return end = input("Godzina zakończenia (HH:MM): ").strip() if not validate_time(end): print("✗ Niepoprawny format godziny zakończenia!") return entry = WorkEntry(date, start, end) data_manager.entries.append(entry) print("✓ Dodano wpis.") def main(): </pre>	<input checked="" type="checkbox"/>		1,5

```

# Zmienna lokalna
(przykład)
data_manager =
DataManager("data/work_log.j
son")
analyzer = Analyzer()

while True:
    print_header()
    print_menu()
    choice =
input("\nWybierz opcję:
").strip()
    if choice == "1":
add_entry(data_manager)
    elif choice == "2":

print_entries(data_manager.e
ntries)
    elif choice == "3":

edit_entry(data_manager)
    elif choice == "4":
        print("\n---
Wyszukiwanie wpisów po dacie
---")
        search_date =
input(
            "Podaj datę
do wyszukania (YYYY-MM-DD):
").strip()
        found =
list(filter(lambda e: e.date
==
search_date,
data_manager.entries))

print_entries(found)
    elif choice == "5":

analyzer.analyze(data_manage
r.entries)
    elif choice == "6":
        try:

data_manager.save()
        print("✅
Dane zapisane.")
    except Exception
as e:
        print(f"Błąd
zapisu: {e}")
    elif choice == "7":

analyzer.plot(data_manager.e
ntries)
    elif choice == "8":

remove_entry(data_manager)
    elif choice == "9":

```

			<pre> total = recursive_sum(data_manager.e ntries) print(f"\n12 34 Rekurencyjna suma godzin: {total:.2f}h") elif choice == "10": print("\n--- Filtruj wpisy ---") try: min_hours = float(input("Pokaż wpisy z czasem pracy >= (godz): ")) filtered = filter_entries(data_manager.entries, lambda e: e.duration() >= min_hours) print_entries(filtered) except ValueError: print("X Podano niepoprawną wartość.") elif choice == "11": string_operations_demo() elif choice == "12": filename = input("Podaj nazwę pliku CSV do eksportu: ").strip() export_to_csv(data_manager.e ntries, filename) elif choice == "13": filename = input("Podaj nazwę pliku CSV do importu: ").strip() import_from_csv(data_manager , filename) elif choice == "0": print("\nDziękujemy za skorzystanie z systemu. Do zobaczenia!") sys.exit(0) else: print("X Niepoprawna opcja. Spróbuj ponownie.") </pre>			
		Zastosowano zakresy klas	<pre> entry.py: klasy WorkEntry, ProjectWorkEntry # models/entry.py from datetime import datetime </pre>	<input checked="" type="checkbox"/>		1,5

```

class WorkEntry:
    """Reprezentuje pojedynczy
    wpis czasu pracy."""

    def __init__(self, date,
start, end):
        # Konstruktor klasy
        WorkEntry
        self.date = date
        self.start = start
        self.end = end

    def duration(self):
        """Zwraca liczbę godzin
        między start a end."""
        try:
            fmt = "%H:%M"
            tdelta =
datetime.strptime(
                self.end, fmt) -
datetime.strptime(self.start,
fmt)

            return
tdelta.total_seconds() / 3600
        except Exception as e:
            print(f"Błąd w
obliczaniu czasu: {e}")
            return 0

    def to_dict(self):
        """Konwertuje wpis do
        słownika."""
        return {"date":
self.date, "start": self.start,
"end": self.end}

    @staticmethod
    def from_dict(d):
        """Tworzy wpis na
        podstawie słownika."""
        return
        WorkEntry(d["date"], d["start"],
d["end"])

    def __str__(self):
        # Formatowanie stringa
        (przykład operacji na stringach)
        return (f"{self.date}: "
                f"{self.start} -
"
                f"{self.end} "
f"({self.duration():.2f}h)")

class
ProjectWorkEntry(WorkEntry):
    """Wpis czasu pracy z
    informacją o
    projekcie (dziedziczenie po
    WorkEntry)."""

    def init (self, date,

```


			<pre> start, end, project): # Konstruktor klasy dziedziczacej super().__init__(date, start, end) self.project = project def to_dict(self): # Rozszerzenie metody bazowej d = super().to_dict() d["project"] = self.project return d @staticmethod def from_dict(d): # Tworzenie obiektu z dodatkowym polem return ProjectWorkEntry(d["date"], d["start"], d["end"], d.get("project", "")) def __str__(self): # Formatowanie stringa z informacją o projekcie return (f"{self.date}: {self.start} - " f"{self.end} ({self.duration():.2f}h) " f"[Projekt: {self.project}]") </pre>			
5	Moduły i pakiety	Projekt podzielony na moduły (import, init)	<pre> main.py, entry.py, analyzer.py, data_manager.py, validators.py from services.data_manager import DataManager from services.analyzer import Analyzer from models.entry import WorkEntry from utils.validators import validate_date, validate_time, log_operation </pre>	<input checked="" type="checkbox"/>		2

Nr	Obszar	Wymaganie	KOD		Przyznane pkt	Pkt max
		Własne pakiety/funkcje pomocnicze w osobnych plikach .py	<pre> utils/validators.py # utils/validators.py from datetime import datetime import functools def log_operation(func): """Dekorator logujący operacje na wpisach.""" </pre>	<input checked="" type="checkbox"/>		2

			<pre> @functools.wraps(func) def wrapper(*args, **kwargs): print(f"[LOG] Wywołano: {func.__name__}") return func(*args, **kwargs) return wrapper def validate_date(date_str): """Waliduj datę w formacie YYYY-MM-DD.""" try: datetime.strptime(date_str, "%Y-%m-%d") return True except ValueError: return False def validate_time(time_str): """Waliduj czas w formacie HH:MM.""" try: datetime.strptime(time_str, "%H:%M") return True except ValueError: return False </pre>			
6	Obsługa błędów	Obsługa wyjątków (try, except, finally)	<p>main.py: try/except w UI, data_manager.py: try/except w load/save</p> <pre> def remove_entry(data_manager): """Usuwa wpis czasu pracy na podstawie numeru.""" if not data_manager.entries: print("Brak wpisów do usunięcia.") return print("\n--- Usuń wpis ---") for i, e in enumerate(data_manager.entries): print(f"{i + 1}. {e}") try: index = int(input("Podaj numer wpisu do usunięcia: ")) - 1 if 0 <= index < len(data_manager.entries): removed = data_manager.entries.pop(index) print(f"✅ Usunięto wpis: {removed}") else: print("❌ Niepoprawny indeks.") except Exception as e: print(f"Błąd usuwania: {e}") def edit_entry(data_manager): """Edycja wybranego wpisu czasu pracy.""" if not data_manager.entries: print("Brak wpisów do edycji.") return print("\n--- Edycja wpisu ---") for i, e in enumerate(data_manager.entries): print(f"{i + 1}. {e}") try: index = int(input("Podaj numer </pre>	☑		2

			<pre> wpisu do edycji: ")) - 1 if 0 <= index < len(data_manager.entries): entry = data_manager.entries[index] print(f"Edytujesz: {entry}") new_start = input("Nowa godzina rozpoczęcia (HH:MM): ").strip() if not validate_time(new_start): print("✗ Niepoprawny format godziny!") return new_end = input("Nowa godzina zakończenia (HH:MM): ").strip() if not validate_time(new_end): print("✗ Niepoprawny format godziny!") return entry.start = new_start entry.end = new_end print("✓ Zmieniono wpis.") else: print("✗ Niepoprawny indeks.") except Exception as e: print(f"Błąd edycji: {e}") </pre>			
		<p>Użycie assert do testów i walidacji</p>	<pre> main.py: assert w add_entry() def add_entry(data_manager): """Dodaje nowy wpis czasu pracy z walidacją i atrakcyjnym komunikatem.""" print("\n--- Dodawanie nowego wpisu ---") date = input("Podaj datę (YYYY-MM-DD): ").strip() assert len(date) == 10, "Data powinna mieć 10 znaków (YYYY-MM-DD)" if not validate_date(date): print("✗ Niepoprawna data!") return start = input("Godzina rozpoczęcia (HH:MM): ").strip() if not validate_time(start): print("✗ Niepoprawny format godziny rozpoczęcia!") return end = input("Godzina zakończenia (HH:MM): ").strip() if not validate_time(end): print("✗ Niepoprawny format godziny zakończenia!") return entry = WorkEntry(date, start, end) data_manager.entries.append(entry) print("✓ Dodano wpis.") </pre>	✓		1,5
7	Łańcuchy znaków	Operacje na stringach (m.in. formatowanie,	<pre> main.py: string operations demo(), entry.py: _str() def string_operations_demo(): """Pokazuje operacje na stringach: dzielenie, wyszukiwanie.""" print("\n--- Operacje na stringach ---") s = input("Podaj tekst do demonstracji </pre>	✓		2

		dzielenie, wyszukiwanie)	<pre> operacji na stringach: ") parts = s.split(" ") print(f" ♦ Podzielony tekst: {parts}") search = input("Podaj fragment do wyszukania: ") found = s.find(search) if found != -1: print(f"🔍 Znalezione '{search}' na pozycji {found}") else: print(f"❌ Nie znaleziono '{search}' w tekście.") entry.py def __str__(self): # Formatowanie stringa (przykład operacji na stringach) return (f"{self.date}: " f"{self.start} - " f"{self.end} " f"({self.duration():.2f}h)") def __str__(self): # Formatowanie stringa z informacją o projekcie return (f"{self.date}: {self.start} - " f"{self.end} ({self.duration():.2f}h) " f"[Projekt: {self.project}]") </pre>			
8	Obsługa plików	Odczyt z plików .txt, .csv, .json, .xml (min. 1)	<pre> data_manager.py: load(), main.py: import_from_csv() # services/data_manager.py import json import os from models.entry import WorkEntry class DataManager: """Zarządza zapisem i odczytem wpisów czasu pracy.""" def __init__(self, filepath): # Konstruktor klasy DataManager self.filepath = filepath self.entries = self.load() def load(self): """Wczytuje dane z pliku.""" if not os.path.exists(self.filepath): return [] try: with open(self.filepath, "r") as f: data = json.load(f) # Tworzenie listy obiektów WorkEntry # na podstawie danych z pliku return [WorkEntry.from_dict(e) for e in data] </pre>	<input checked="" type="checkbox"/>		2

		<pre> except Exception as e: print(f"Błąd wczytywania danych: {e}") return [] def save(self): """Zapisuje dane do pliku.""" try: with open(self.filepath, "w") as f: # Zapis listy wpisów jako listy słowników json.dump([e.to_dict() for e in self.entries], f, indent=4) except Exception as e: print(f"Błąd zapisu danych: {e}") def import_from_csv(data_manager, filename): """Importuje wpisy z pliku CSV.""" try: with open(filename, "r", encoding="utf-8") as f: reader = csv.DictReader(f) count = 0 for row in reader: if (validate_date(row["date"]) and validate_time(row["start"]) and validate_time(row["end"])): data_manager.entries.append(WorkEntry(row["date"], row["start"], row["end"])) count += 1 print(f"✅ Zaimportowano {count} wpisów z pliku {filename}") except Exception as e: print(f"Błąd importu CSV: {e}") </pre>			
	<p>Zapis do plików .txt, .csv, .json, .xml (min. 1)</p>	<p>data_manager.py: save(), main.py: export_to_csv()</p> <pre> def export_to_csv(entries, filename): """Eksportuje wpisy do pliku CSV.""" try: with open(filename, "w", newline="", encoding="utf-8") as f: writer = csv.writer(f) writer.writerow(["date", "start", "end"]) for e in entries: writer.writerow([e.date, e.start, e.end]) print(f"✅ Wyeksportowano do pliku {filename}") </pre>	☑		2

			<pre> except Exception as e: print(f"Błąd eksportu CSV: {e}") # services/data_manager.py import json import os from models.entry import WorkEntry class DataManager: """Zarządza zapisem i odczytem wpisów czasu pracy.""" def __init__(self, filepath): # Konstruktor klasy DataManager self.filepath = filepath self.entries = self.load() def load(self): """Wczytuje dane z pliku.""" if not os.path.exists(self.filepath): return [] try: with open(self.filepath, "r") as f: data = json.load(f) # Tworzenie listy obiektów WorkEntry # na podstawie danych z pliku return [WorkEntry.from_dict(e) for e in data] except Exception as e: print(f"Błąd wczytywania danych: {e}") return [] def save(self): """Zapisuje dane do pliku.""" try: with open(self.filepath, "w") as f: # Zapis listy wpisów jako listy słowników json.dump([e.to_dict() for e in self.entries], f, indent=4) except Exception as e: print(f"Błąd zapisu danych: {e}") </pre>			
9	OOP	Klasy	<pre> entry.py: WorkEntry, ProjectWorkEntry # models/entry.py from datetime import datetime class WorkEntry: """Reprezentuje pojedynczy wpis czasu pracy.""" def __init__(self, date, start, end): # Konstruktor klasy WorkEntry self.date = date </pre>	<input checked="" type="checkbox"/>		2

```

        self.start = start
        self.end = end

    def duration(self):
        """Zwraca liczbę godzin między
        start a end."""
        try:
            fmt = "%H:%M"
            tdelta = datetime.strptime(
                self.end, fmt) -
            datetime.strptime(self.start, fmt)
            return tdelta.total_seconds()
            / 3600
        except Exception as e:
            print(f"Błąd w obliczaniu
            czasu: {e}")
            return 0

    def to_dict(self):
        """Konwertuje wpis do słownika."""
        return {"date": self.date,
            "start": self.start, "end": self.end}

    @staticmethod
    def from_dict(d):
        """Tworzy wpis na podstawie
        słownika."""
        return WorkEntry(d["date"],
            d["start"], d["end"])

    def __str__(self):
        # Formatowanie stringa (przykład
        operacji na stringach)
        return (f"{self.date}: "
            f"{self.start} - "
            f"{self.end} "
            f"({self.duration():.2f}h)")

class ProjectWorkEntry(WorkEntry):
    """Wpis czasu pracy z informacją o
    projekcie (dziedziczenie po
    WorkEntry)."""

    def __init__(self, date, start, end,
        project):
        # Konstruktor klasy dziedziczącej
        super().__init__(date, start, end)
        self.project = project

    def to_dict(self):
        # Rozszerzenie metody bazowej
        d = super().to_dict()
        d["project"] = self.project
        return d

    @staticmethod
    def from_dict(d):
        # Tworzenie obiektu z dodatkowym
        polem
        return ProjectWorkEntry(d["date"],
            d["start"],

```

		<pre> d["end"], d.get("project", "") def __str__(self): # Formatowanie stringa z informacją o projekcie return (f"{self.date}: {self.start} - " f"{self.end} ({self.duration():.2f}h) " f"[Projekt: {self.project}]") </pre>			
	Metody	<pre> entry.py: duration(), to_dict(), from_dict(), str () def duration(self): """Zwraca liczbę godzin między start a end.""" try: fmt = "%H:%M" tdelta = datetime.strptime(self.end, fmt) - datetime.strptime(self.start, fmt) return tdelta.total_seconds() / 3600 except Exception as e: print(f"Błąd w obliczaniu czasu: {e}") return 0 def to_dict(self): """Konwertuje wpis do słownika.""" return {"date": self.date, "start": self.start, "end": self.end} @staticmethod def from_dict(d): """Tworzy wpis na podstawie słownika.""" return WorkEntry(d["date"], d["start"], d["end"]) def __str__(self): # Formatowanie stringa (przykład operacji na stringach) return (f"{self.date}: " f"{self.start} - " f"{self.end} " f"({self.duration():.2f}h)") </pre>	<input checked="" type="checkbox"/>		2
	Konstruktory	<pre> entry.py: __init__() def __init__(self, date, start, end): # Konstruktor klasy WorkEntry self.date = date self.start = start self.end = end </pre>	<input checked="" type="checkbox"/>		2
	Dziedziczenie	<pre> entry.py: ProjectWorkEntry(WorkEntry) class ProjectWorkEntry(WorkEntry): """Wpis czasu pracy z informacją o projekcie (dziedziczenie po WorkEntry).""" </pre>	<input checked="" type="checkbox"/>		2

			<pre> def __init__(self, date, start, end, project): # Konstruktor klasy dziedziczącej super().__init__(date, start, end) self.project = project def to_dict(self): # Rozszerzenie metody bazowej d = super().to_dict() d["project"] = self.project return d @staticmethod def from_dict(d): # Tworzenie obiektu z dodatkowym polem return ProjectWorkEntry(d["date"], d["start"], d["end"], d.get("project", "")) def __str__(self): # Formatowanie stringa z informacja o projekcie return (f"{self.date}: {self.start} - " f"{self.end} ({self.duration():.2f}h) " f"[Projekt: {self.project}]") </pre>			
10	Programowanie funkcyjne	map	<p>main.py: map() w print_entries()</p> <pre> def print_entries(entries): """Zadanie wyświetla wszystkie wpisy.""" if not entries: print("Brak wpisów do wyświetlenia.") return print("\n--- Lista wpisów ---") for i, e in enumerate(entries, 1): print(f"{str(i).rjust(2)}. {e}") </pre>	<input checked="" type="checkbox"/>		1,5
		filter	<p>main.py: filter_entries(), filter() w wyszukiwaniu</p> <pre> def filter_entries(entries, predicate): """Zwraca listę wpisów spełniających warunek predicate (funkcja jako argument).""" return [e for e in entries if predicate(e)] </pre>	<input checked="" type="checkbox"/>		1,5
		lambda	<p>main.py: lambda w filter_entries()</p> <pre> elif choice == "4": print("\n--- Wyszukiwanie wpisów po dacie ---") search_date = input("Podać datę do wyszukania (YYYY-MM-DD): ").strip() found = list(filter(lambda e: e.date </pre>	<input checked="" type="checkbox"/>		1,5








			<pre>== search_date, data_manager.entries)) print_entries(found)</pre>			
--	--	--	--	--	--	--

		reduce	<pre> analyzer.py: reduce() w analyzer() def analyze(self, entries): """Wyświetla statystyki i analizę czasu pracy.""" if not entries: print("Brak danych do analizy.") return # reduce - suma godzin (programowanie funkcyjne) total = reduce(lambda acc, e: acc + e.duration(), entries, 0) avg = total / len(entries) print("\nStatystyki czasu pracy:") print(f" - Łączny czas pracy: {total:.2f}h") print(f" - Średni czas na wpis: {avg:.2f}h") # słownik miesięcy, krotki, zbiór miesięcy (przykład użycia kontenerów) monthly = defaultdict(float) for e in entries: monthly[e.date[:7]] += e.duration() print("\nCzas pracy wg miesięcy:") for month, hours in monthly.items(): tup = (month, hours) # krotka print(f" - {tup[0]}: {tup[1]:.2f}h") </pre>	☑		1,5
11	Wizualizacja danych	Wygenerowano wykres (np. matplotlib, seaborn)	<pre> analyzer.py: plot() def plot(self, entries): """Generuje i zapisuje wykres czasu pracy.""" if not entries: print("Brak danych do wykresu.") return # Słownik z sumą godzin dla każdej daty daily = defaultdict(float) for e in entries: daily[e.date] += e.duration() dates = sorted(daily.keys()) hours = [daily[d] for d in dates] # Tworzenie wykresu słupkowego plt.figure(figsize=(10, 5)) plt.bar(dates, hours, color='skyblue') plt.xlabel("Data") plt.ylabel("Godziny pracy") plt.title("Czas pracy dzienny") plt.xticks(rotation=45) plt.tight_layout() plt.savefig("data/work_plot.png") plt.show() print("Wykres zapisano jako </pre>	☑		2

			data/work_plot.png")			
		Zapisano wykres do pliku graficznego (.png lub .jpg)	<pre> analyzer.py: plot() -> plt.savefig() plt.savefig("data/work_plot.png") </pre>	☑		1,5
T12	Testowanie	Testy jednostkowe (assert, unittest, pytest)	<pre> test_entry.py: TestValidators, TestWorkEntry # Testy walidatorów i dekoratora class TestValidators(unittest.TestCase): def test_validate_date_correct(self): self.assertTrue(validate_date("2024-06-01")) def test_validate_date_incorrect(self): self.assertFalse(validate_date("01-06-2024")) def test_validate_time_correct(self): self.assertTrue(validate_time("08:00")) def test_validate_time_incorrect(self): self.assertFalse(validate_time("800")) def test_log_operation_decorator(self): calls = [] @log_operation def foo(): calls.append(1) return 42 result = foo() self.assertEqual(result, 42) self.assertEqual(len(calls), 1) </pre>	☑		1,5
		Testy funkcjonalne	<pre> test_entry.py: TestFunctional # Test funkcjonalny (dodanie i usunięcie wpisu) class TestFunctional(unittest.TestCase): def test_add_and_remove_entry(self): manager = DataManager(":memory:") entry = WorkEntry("2024-01-01", "08:00", "16:00") manager.entries.append(entry) self.assertEqual(len(manager.entries), 1) manager.entries.pop(0) self.assertEqual(len(manager.entries), </pre>	☑		1,5

			0)			
		Testy Integracyjne	<pre> test_entry.py: TestIntegration # Test integracyjny (zapis/odczyt ProjectWorkEntry) def test_save_and_load_project_entry(self): base_dir = os.path.dirname(os.path.dirname(__file__)) # katalog główny projektu data_dir = os.path.join(base_dir, "data") # upewniamy się, że katalog istnieje os.makedirs(data_dir, exist_ok=True) test_file = os.path.join(data_dir, "test_project_entry.json") entry = ProjectWorkEntry("2024-01-01", "08:00", "16:00", "ProjX") manager = DataManager(test_file) manager.entries = [entry] manager.save() manager2 = DataManager(test_file) self.assertEqual(manager2.entries[0].da te, "2024-01-01") # Clean up if os.path.exists(test_file): os.remove(test_file) </pre>	<input checked="" type="checkbox"/>		1,5
		Testy graniczne / błędne dane	<pre> test_entry.py: test_duration_invalid_format() , test duration invalid time() class TestWorkEntry(unittest.TestCase): def test_duration_valid(self): entry = WorkEntry("2024-01-01", "08:00", "16:00") self.assertEqual(entry.duration(), 8.0) def test_duration_invalid_format(self): entry = WorkEntry("2024-01-01", "invalid", "16:00") self.assertEqual(entry.duration(), 0) def test_to_dict(self): entry = WorkEntry("2024-01-01", "08:00", "16:00") expected = {"date": "2024-01-01", "start": "08:00", "end": "16:00"} </pre>	<input checked="" type="checkbox"/>		1,5

			<pre> self.assertEqual(entry.to_dict(), expected) def test_from_dict(self): data = {"date": "2024-01-01", "start": "08:00", "end": "16:00"} entry = WorkEntry.from_dict(data) self.assertEqual(entry.date, "2024-01-01") self.assertEqual(entry.start, "08:00") self.assertEqual(entry.end, "16:00") def test_duration_negative(self): entry = WorkEntry("2024-01-01", "16:00", "08:00") self.assertEqual(entry.duration(), -8) def test_duration_invalid_time(self): entry = WorkEntry("2024-01-01", "xx:yy", "16:00") self.assertEqual(entry.duration(), 0) </pre>			
		Testy wydajności (np. czas wykonania, timeit)	<pre> test_entry.py: TestPerformance class TestPerformance(unittest.TestCase): def test_duration_performance(self): entry = WorkEntry("2024-01-01", "08:00", "16:00") duration = timeit.timeit(lambda: entry.duration(), number=10000) self.assertLess(duration, 1) # powinno być szybkie </pre>	<input checked="" type="checkbox"/>		1,5
		Testy pamięci memory_profiler	<pre> test_entry.py: test_memory_usage() # Test pamięci i jakości kodu - tylko szkielet (do uruchomienia z zewnątrz) def test_memory_usage(self): from memory_profiler import memory_usage entry = WorkEntry("2024-01-01", "08:00", "16:00") mem = memory_usage((entry.duration,)) assert max(mem) - min(mem) < 10 # MB </pre>	<input checked="" type="checkbox"/>		1,5
		Test jakości kodu (flake8, pylint)	<pre> test_entry.py: test_code_quality() def test_code_quality(self): import subprocess </pre>	<input checked="" type="checkbox"/>		1,5

			<pre> result = subprocess.run(["flake8", "--exclude=.venv"], capture_output=True, text=True) if result.returncode != 0: print("\nFlake8 errors:\n" + result.stdout) assert result.returncode == 0, "Flake8 reported style violations" </pre>			
13	Wersjonowa nie	Repozytorium GIT	work-time-tracker	<input checked="" type="checkbox"/>		1
		Historia commitów	<p>Commits on Jun 26, 2025</p> <p>Rozwiązanie problemów zgłaszanych przez Flake8  Błażej-Knap committed 36 minutes ago 1c5a469</p> <p>Commits on Jun 24, 2025</p> <p>Dodanie dokumentacji projektu: RAPORT.md  alkolodziej committed 2 days ago 7d76f98</p> <p>Commits on Jun 21, 2025</p> <p>Dodanie dokumentacji, przykładowych danych i pliku eksportu  alkolodziej committed 5 days ago 9b43b96</p> <p>Dodanie testów jednostkowych dla modeli i serwisów  alkolodziej committed 5 days ago f5917d8</p> <p>Dodanie głównego pliku aplikacji: main.py  alkolodziej committed 5 days ago 6883b1e</p> <p>Dodanie warstwy serwisów: analiza i zarządzanie danymi  alkolodziej committed 5 days ago a925d3f</p> <p>Dodanie narzędzi walidujących i funkcji pomocniczych  alkolodziej committed 5 days ago</p>	<input checked="" type="checkbox"/>		1

			559765c Dodanie modeli danych: struktura Entry i powiązane klasy  alkolodziej committed 5 days ago			
--	--	--	--	--	--	--

Nr	Obszar	Wymaganie	KOD		Przyznane pkt	Pkt max
		Link do GitHub	https://github.com/alkolodziej/work-time-tracker	<input checked="" type="checkbox"/>		1
		Opis commitów	<code>Poprawa kodu do testów jakości kodu</code>	<input checked="" type="checkbox"/>		1
14	Dokumentacja	Plik README.md (cel, autorzy, uruchamianie)	Cel Aplikacja do ewidencji i analizy czasu pracy z obsługą plików, wyjątków, OOP, programowania funkcyjnego i testów. Autorzy Grupa 12B: Alan Kołodziej Błażej Knap Uruchamianie python main.py	<input checked="" type="checkbox"/>		1,5
		Przykładowe dane wejściowe i wyjściowe	README.md: Przykładowe dane wejściowe 2024-06-01, 08:00, 16:00 2024-06-02, 09:00, 17:00 Przykładowe dane wyjściowe 2024-06-01: 08:00 - 16:00 (8.00h) 2024-06-02: 09:00 - 17:00 (8.00h)	<input checked="" type="checkbox"/>		2
		Diagram klas lub struktura modułów	README.md: Diagram klas (tekstowy) WorkEntry — date — start — end — duration() — to_dict() — from_dict() — __str__() ProjectWorkEntry (dziedziczy po WorkEntry) — project — to_dict() — from_dict() — __str__()	<input checked="" type="checkbox"/>		2

			<div>Struktura modułów</div> <div>main.py models/ entry.py services/ data_manager.py analyzer.py utils/ validators.py tests/ test_entry.py</div>			
			SUMA			