Architekturdokument

Projektname: ProductivityGarden

Name	Jonas Huber
Matrikelnummer	IU14085128
Modul	Projekt: Software Engineering DLMCSPSE01_D
Datum	15.01.2025

Inhalt

1. Technologieübersicht

Die Auswahl der Technologien und Werkzeuge für die Entwicklung von ProductivityGarden wurde auf der Grundlage der funktionalen und nicht-funktionalen Anforderungen des Projekts getroffen. Die ausgewählten Technologien ermöglichen eine flexible, performante und benutzerfreundliche Umsetzung der Anforderungen von ProductivityGarden. Im Folgenden werden die verwendeten Technologien beschrieben und ihre Auswahl begründet.

Programmiersprache

Python

Python wurde aufgrund seiner vielfältigen Einsatzmöglichkeiten, einer großen Community und einer Vielzahl von Bibliotheken gewählt. Python ermöglicht eine schnelle Entwicklung und bietet hervorragende Unterstützung für plattformübergreifende Anwendungen.

GUI-Framework

PyQt6

Anwendung: Gestaltung der Benutzeroberfläche (GUI)

Begründung: PyQt6 bietet umfassende Funktionen zur Erstellung responsiver Desktop-GUIs. Es unterstützt eine Vielzahl von Widgets, die für die minimalistische und intuitive Gestaltung der Oberfläche von ProductivityGarden notwendig sind.

Zeitmanagement-Funktionalitäten

PyQt6

Anwendung: Implementierung der Timer-, Stoppuhr- und Pomodoro-Funktionen.

Begründung: Die Ereignissteuerung (Event Loop) von PyQt6 ermöglicht eine einfache Implementierung von zeitbasierten Funktionen wie Timer und Stoppuhren. Die Integration in das GUI-Framework gewährleistet eine reibungslose und performante Ausführung dieser Features.

Spiel- und Grafikbibliothek

Pygame

Anwendung: Rendering von grafischen Elementen, Animationen und Interaktionen in den virtuellen Gärten.

Begründung: Pygame ist eine leichtgewichtige Bibliothek, die sich ideal für Spielelemente eignet. Sie bietet Unterstützung für 2D-Grafiken und Benutzerinteraktionen, was für die Gestaltung der virtuellen Gärten essenziell ist.

Datenpersistenz

JSON

Anwendung: Speicherung von Benutzereinstellungen und Benutzerstatistiken.

Begründung: JSON ist ein leichtgewichtiger, plattformunabhängiger Standard für die Datenserialisierung. Es eignet sich ideal zur Speicherung von benutzerdefinierten Einstellungen und ermöglicht einen schnellen Lese- und Schreibzugriff.

SQLite3

Anwendung: Speicherung und Verwaltung komplexerer Datenstrukturen wie Projektinformationen.

Begründung: SQLite3 ist eine eingebettete Datenbank, die keine separate Serverinstallation erfordert. Sie ist robust, effizient und ermöglicht die Speicherung strukturierter Daten direkt auf dem Gerät des Nutzers, wodurch Datenschutzanforderungen (z. B. DSGVO) erfüllt werden.

2. Architekturübersicht

Die Architektur von "ProductivityGarden" basiert auf einer **Schichtenarchitektur** mit einer klaren Trennung zwischen Datenverwaltung, Logik und Benutzeroberfläche und bietet gleichzeitig Flexibilität für die unterschiedlichen Anforderungen der GUI (PyQt6) und des Spiels (Pygame).

Das System ist in zwei getrennte Stränge unterteilt, die jeweils aus mehreren Schichten bestehen. Die Trennung in PyQt6 und Pygame ermöglicht es, die Stärken beider Frameworks zu nutzen. PyQt6 sorgt für eine reaktionsschnelle und moderne GUI, während Pygame auf grafikintensive und interaktive Animationen spezialisiert ist.

Die Schichtenarchitektur bietet folgende Vorteile:

- Flexibilität: Die parallele Strangstruktur in der Präsentationsschicht ermöglicht eine optimale Nutzung von PyQt6 für GUI-Funktionen und Pygame für das grafikintensive Gamification-Element.
- Wartbarkeit und Erweiterbarkeit: Die Schichtenarchitektur mit klaren Abhängigkeiten erleichtert die Entwicklung und Erweiterung einzelner Komponenten.
- Leistung: Beide Stränge sind unabhängig voneinander und können so optimiert werden, um eine reaktionsfreudige und leistungsfähige Nutzung zu gewährleisten.
- Benutzerfreundlichkeit: Die Benutzerinteraktionen sind klar getrennt: die GUI für die Bedienung der Zeitfunktionen und das Projektmanagement und der Pygame-Strang für das Spielerlebnis.

Abbildung 1 zeigt die Schichtenarchitektur der Anwendung mit der bereits erwähnten Trennung zwischen der Produktivanwendung und der Spielkomponente. Beide Bereiche haben ihre eigenen Komponenten in jeder Schicht.

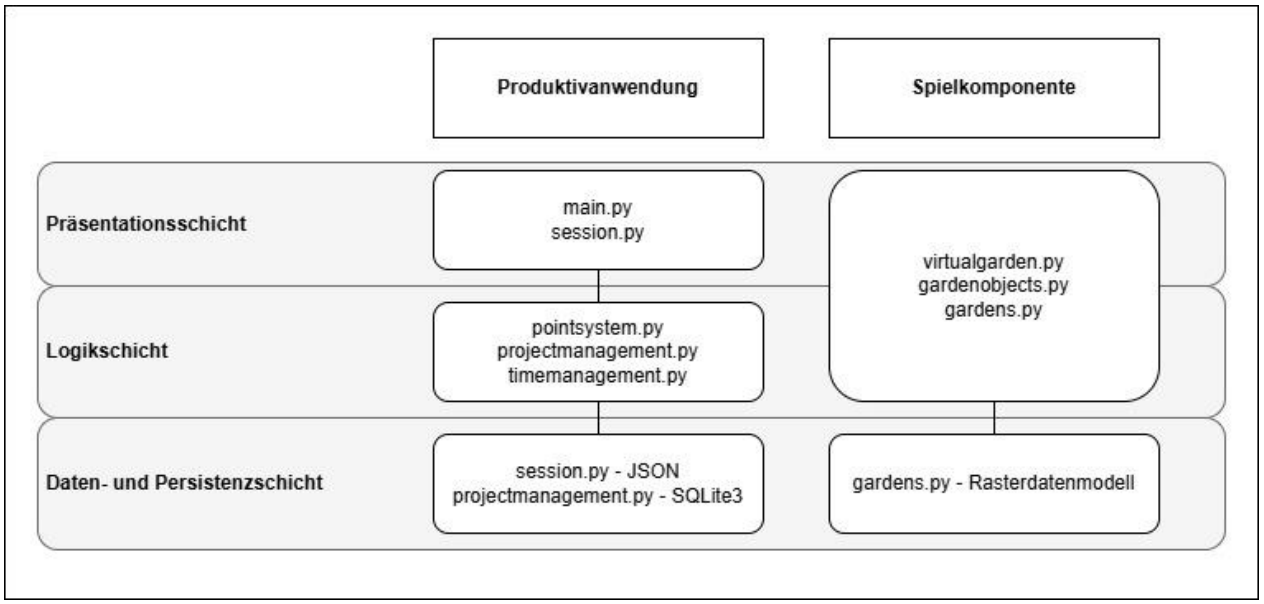

Abbildung 1 – Schichtenmodell der ProductivityGarden-Anwendung

Strang der Produktivanwendung

Präsentationsschicht

Diese Schicht enthält die Benutzeroberfläche, die mit dem PyQt6 Framework entwickelt wurde. Sie stellt das Hauptfenster der Anwendung dar, das für die Bedienung der Timer, die Projektverwaltung und die Konfiguration zuständig ist.

Komponenten:

- *main.py*: Startet die Anwendung und initialisiert die PyQt6-Oberfläche
- *session.py*: Beinhaltet die Logik und Darstellung der GUI

Logikschicht

Die Logikschicht enthält die zentrale Logik der Anwendung. Diese Schicht sorgt für:

- Verwaltung der Timer und Zeitmessungen
- Verwaltung der Projekte und Zuordnung von Zeit zu Projekten
- Berechnung und Verwaltung des Punkte-Systems

Die Logikschicht wurde modular und unabhängig von der Darstellung gestaltet, um Wiederverwendbarkeit und einfache Wartbarkeit sicherzustellen.

Komponenten:

- *pointsystem.py*: Punkteberechnung basierend auf produktiver Zeit
- *projectmanagement.py*: Verwaltung der Projekte und zugehörigen Arbeitszeiten
- *timemanagement.py*: Implementierung von Timer, Stoppuhr und Pomodoro-Funktion

Daten- und Persistenzschicht

Diese Schicht ist für die Speicherung und Verwaltung von Nutzer- und Projektdaten verantwortlich.

Komponenten:

- SQLite3: Speicherung persistenter Daten wie Projekte und Punkte
- JSON: Speicherung temporärer Daten und Konfigurationsdateien

Strang der Spielkomponente

Präsentations- und Logikschicht:

Umfasst die Darstellung des Gamification-Elements, inklusive virtuellem Garten mit Interaktionen und Animationen mittels Pygame. Da die Interaktionen, die hinterlegte Logik und die daraus resultierenden Reaktionen wie Animationen in einem Spiel verschmelzen, wurde hier die Präsentationsschicht und die Logikschicht zusammengeführt.

Komponenten:

- *virtualgarden.py*: Verantwortlich für das Rendering und die Interaktionen im virtuellen Garten
- *gardens.py*: Modellierung der virtuellen Gärten
- *gardenobjects.py*: Modellierung der Gartenobjekte

Daten- und Persistenzschicht:

Beinhaltet das Speichern und Verwalten von virtuellen Gärten und deren Spielstand, d.h. welche Objekte sind bereits an welcher Stelle platziert worden.

Komponenten:

- *gardens.py*: ermöglicht das Laden und Speichern eines virtuellen Gartens über ein Rasterdatenmodell

3. Struktur

In diesem Kapitel erfolgt die Beschreibung der Struktur der Anwendung. Der Fokus liegt hierbei auf den Hauptkomponenten und -klassen der Anwendung, deren Beziehungen und Abhängigkeiten zueinander sowie auf den wichtigsten Attributen und Methoden. Die visuelle Darstellung dieser Strukturen erfolgt mittels UML-Klassendiagramm.

Hauptkomponenten der Anwendung

Die Anwendung wird in die folgenden Hauptklassen unterteilt:

- main(): Dient als Startpunkt für die Ausführung der Anwendung
- MainSession: Hauptklasse für die GUI und den Produktivanwendungsteil
- virtualgardens.py: Hauptskript für die Spielkomponente mit den virtuellen Gärten
- TimeManagement: Verwaltung der Timer-Logik (Pomodoro, Stoppuhr, normaler Timer)
- ProjectManagement: Verwaltung der Projekte und produktiven Zeiten, sowie deren Speicherung.
- PointsSystem: Verwaltung der gesammelten und einsetzbaren Punkte.
- Garden: Verwaltung der virtuellen Gärten, inklusive Laden und Speichern der Daten.
- GardenObject: Repräsentiert Objekte, die in den virtuellen Gärten platziert werden können.

UML-Klassendiagramm

Abbildung 2 zeigt das UML-Klassendiagramm mit den Hauptkomponenten der Anwendung. Dieses dient der visuellen Darstellung der Struktur und der Beziehungen zwischen den Hauptkomponenten. Es bietet einen klaren Überblick über die Attribute und Methoden der einzelnen Klassen sowie deren Vererbungs- und Abhängigkeitsbeziehungen.

Hinweise

1. Die Gliederung in 3 Ebenen wurde bewusst gewählt, um eine Top-Down-Struktur widerzuspiegeln. Oben befindet sich der Startpunkt für die Ausführung der Anwendung mit der *main()* Funktion. Darunter befinden sich die beiden Hauptklassen für die beiden verschiedenen Teilkomponenten der Anwendung und darunter die verschiedenen Klassen, die zusätzliche Komponenten der Logik und des Datenmanagements übernehmen.
2. Die Methode *create_gui()* der Klasse *MainSession* enthält alle weiteren Komponenten der Klasse, die zur Erstellung der GUI implementiert sind. Da eine detaillierte Auflistung das Diagramm unübersichtlich machen würde und keinen zusätzlichen Nutzen bringt, wurden diese in einer Methode zusammengefasst.
3. Die beiden Hauptskripte „virtualguardens.py“ und „session.py“ (beinhaltet die MainSession-Klasse) der beiden geteilten Applikationen können sich gegenseitig aufrufen. Somit kann nahtlos von der einen Applikation zur anderen gewechselt werden.

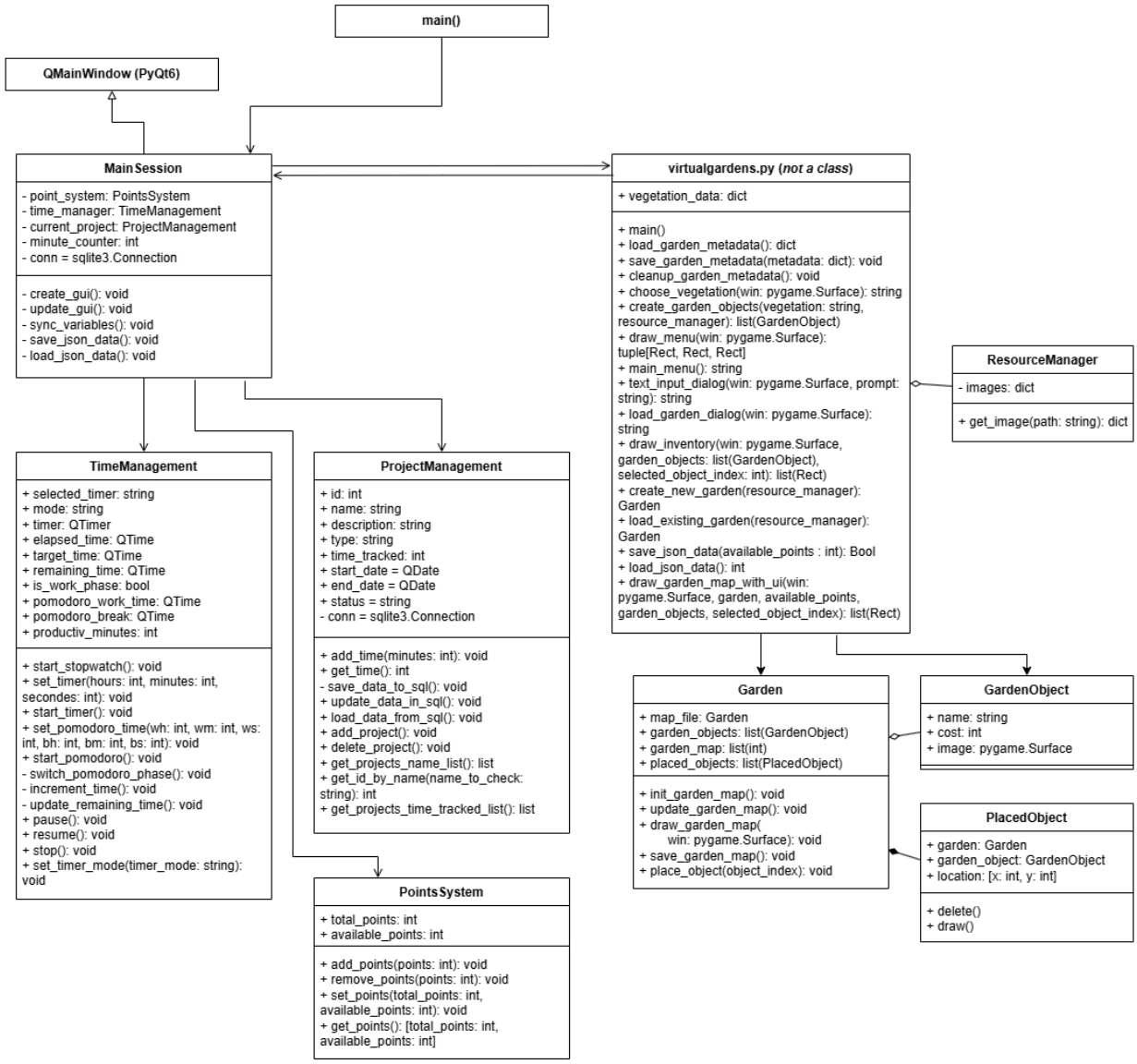

Abbildung 2 – UML-Klassendiagramm

4. Verhalten des zentralen Systemablaufs: Pomodoro-Timer

Im Folgenden wird ein zentraler Systemablauf eines Kernfeatures der Anwendung anhand des Systemablaufs des Pomodoro-Timers beispielhaft beschrieben und visualisiert.

Szenario:

Eine Person startet einen Pomodoro-Timer über die Benutzeroberfläche. Die Eingabe der Timer-Felder wird überprüft und wenn diese gültig sind, startet der Timer. Während der Timer läuft, erhöht sich die Produktivzeit des aktuell ausgewählten Projekts automatisch jede Minute und alle zehn Minuten wird ein Punkt auf dem Punktekonto gutgeschrieben.

Akteure:

Person: Interagiert mit der Benutzeroberfläche und startet den Timer

MainSession: Zuständig für die Benutzeroberfläche und Logik

TimeManagement: Verwalten des Timers

PointsSystem: Hinzufügen von Punkten

ProjectManagement: Speichern der Produktivzeit zum zugehörigen Projekt

UML-Sequenzdiagramm

Das in Abbildung 3 dargestellt UML-Sequenzdiagramm veranschaulicht den zentralen Systemablauf dar und visualisiert die Interaktionen zwischen den einzelnen Komponenten während eines typischen Ablaufes.

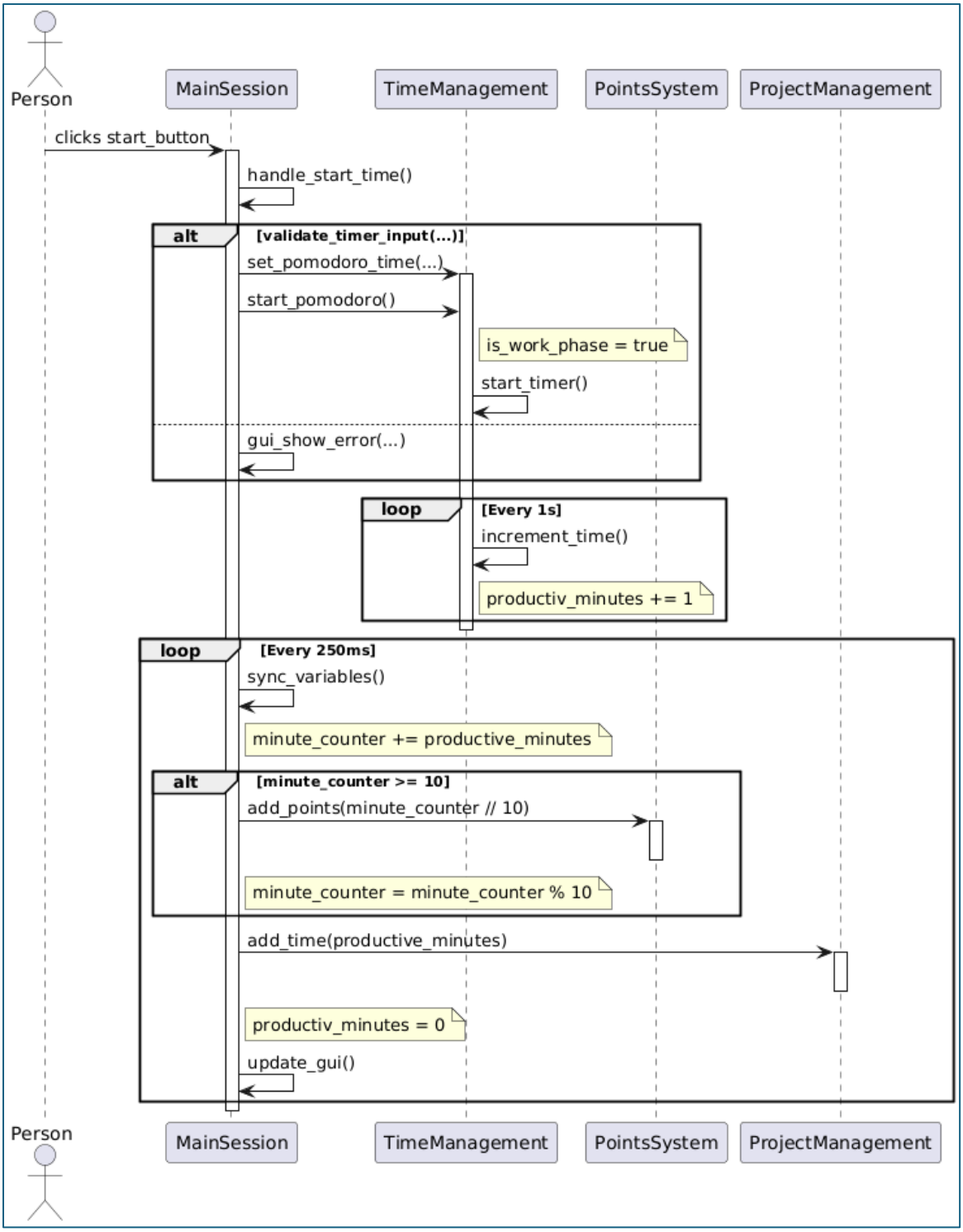

Abbildung 3 – UML-Sequenzdiagramm