Projektdokumentation

Projektname: ProductivityGarden

Name	Jonas Huber
Matrikelnummer	IU14085128
Modul	Projekt: Software Engineering DLMCSPSE01_D
Datum	25.01.2025

Inhalt

Abbildungsverzeichnis

1. Projektbeschreibung

Im modernen Alltag sehen sich Menschen häufig mit der Herausforderung konfrontiert, sich zu konzentrieren und ihre Zeit effektiv zu managen. Studien belegen, dass Gamification einen positiven Einfluss auf Motivation und Produktivität ausübt (Matallaoui, 2016, S. 4). Gamification, auch Spielifizierung genannt, bezeichnet die Anwendung spielerischer Elemente und Mechanismen in einem nicht-spielerischen Kontext, beispielsweise im Bildungsbereich (Huseynli, 2024, S. 45). Die Integration eines flexiblen Pomodoro-Timers, eines klassischen Timers sowie einer Stoppuhr ermöglicht es den Nutzer:innen, ihre produktiven Phasen präzise zu planen und nachzuverfolgen. Dies erfolgt in einer entspannenden und spielerischen Umgebung, wodurch eine nachhaltige Steigerung der Produktivität gefördert und gleichzeitig eine Reduktion von Stress erzielt werden soll. Zur Optimierung des Zeitmanagements besteht die Möglichkeit, die aufgewendete Zeit einzelnen Projekten zuzuweisen.

Ein wesentlicher Bestandteil der Anwendung ist ein Belohnungssystem, bei dem Nutzer:innen für produktive Zeit Punkte sammeln können. Diese Punkte können dazu verwendet werden, virtuelle Gärten mit Pflanzen und Dekorationen zu gestalten. Die Nutzer:innen haben die Möglichkeit, mehrere Gärten anzulegen und individuell anzupassen. Dies fördert nicht nur die Motivation, sondern auch Entspannung und Kreativität

Die Anwendung ist für das Betriebssystem Windows optimiert und bleibt im Hintergrund aktiv, um eine nahtlose Nutzung zu ermöglichen. Die Speicherung sämtlicher Daten erfolgt lokal, um die Privatsphäre der Nutzer:innen zu schützen Die Oberfläche verzichtet auf ablenkende Elemente, um den Nutzer:innen eine ungestörte Konzentration auf das Spielerlebnis und die Produktivität zu ermöglichen. ProductivityGarden ist werbefrei und bietet keinen Zugang zu Paywalls oder Premium-Inhalten.

Im Gegensatz zu herkömmlichen Apps zur Prokrastinationskontrolle richtet sich ProductivityGarden an eine breite Zielgruppe und ist für alle geeignet, die ihr Zeitmanagement verbessern möchten, eine zugleich entspannende und kreative Herangehensweise.

2. Benutzeranleitung

Installation und notwendige Ressourcen

Um die Anwendung zum Laufen zu bekommen helfen:

1. Repository von GitHub pullen:
   https://github.com/Xecu114/iu_pse
2. "README" lesen und beachten
3. Python Version checken: kompatibel ab 3.9
4. Es werden zwei weitere Python-Pakete benötigt, damit die Anwendung ausgeführt werden kann:
   a. PyQt6
   b. Pygame
   Diese können auch mit Hilfe des „install_py_libs.bat" Skripts automatisch installiert werden.
5. Anwendung starten:
   ```
   python main.py
   ```

Bedienung und Oberfläche

Anwendung starten über „main.py" (z.B. mit: `python main.py`)

Damit öffnet sich das Hauptfenster (in den weiteren Dokumenten auch oft als „Produktivanwendung" bezeichnet):

Hauptfenster (session.py)
Das Hauptfenster ist in drei Spalten aufgeteilt:

Erste Spalte

- Punktestand (zwei Kreise mit „available" und „total" Punkten).
- Button „TO THE GARDENS": Öffnet den virtuellen Garten in einem separaten Fenster/Prozess.
- Darunter ein Bild von einer Blumenwiese

Zweite Spalte

- Zeitmanagement-Bereich mit Timer-Modus und digitaler Uhr
- Steuerknöpfe: Start, Pause/Resume, Stop
- Eingabefelder für Pomodoro-Arbeitszeit, -Pausenzeit, sowie klassischen Timer
- Pomodoro-Mode (Voreinstellung: 25 Min. Arbeit, 5 Min. Pause)
- Ein Button „Switch to Timer", „Switch to Stopwatch" oder „Switch to Pomodoro" zum Wechseln zwischen den verschiedenen Zeitmess-Modi
- Fehlermeldungen bei falscher Zeiteingabe (bspw. Zeit < 1 Min. oder > 24 Std.).
- Textfeld im unteren Teil für Notizen, Aufgaben etc.

Dritte Spalte

- Projektverwaltung: Auswahl eines Projekts (Drop-down-Menü), Hinzufügen und Löschen von Projekten.
- Bearbeiten: Name, Beschreibung, Kategorie, Start- und End-Datum eines Projekts.
- Kreis mit getrackter Zeit (Minuten) für das aktuell ausgewählte Projekt.
- Kreisdiagramm (Pie-Chart) aller Projekte und deren getrackter Gesamtzeit.
- Option zum manuellen Hinzufügen von Zeit in Minuten (z.B. wenn man offline gearbeitet hat).

Funktionsweise:

Timer/Stopwatch/Pomodoro:

- Beim Starten (Start-Button) läuft die gewählte Zeitmessung.
- Bei Pause kann man den Timer anhalten, Resume führt ihn weiter. Stop setzt alles zurück.
- Jede getrackte Produktivzeit addiert sich intern in minute_counter. Alle 10 gesammelten Minuten gibt es automatisch 1 Punkt (erhältlich im „available points“-Kreis).
- Die Zeit wird dem aktuell ausgewählten Projekt in der Projektauswahl gutgeschrieben.

Projekte:

- Über den Add-Button wird ein neues Projekt in der lokalen Datenbank (sqlite) und im Drop-down-Menü angelegt.
- Delete entfernt das aktuell ausgewählte Projekt aus der Datenbank.
- Im Drop-down-Menü kann man zwischen Projekten wechseln.
- Die Felder Name, Description, Category, Start/End Date sind editierbar und werden regelmäßig in die Datenbank geschrieben.
- Das Kreisdiagramm und die Zeit-Anzeige werden automatisch aktualisiert.

Punkte:

- Neue Punkte erscheinen in „available“, gleichzeitig erhöht sich auch „total“.
- In der JSON-Datei (JSON_FILE) wird der Punktestand regelmäßig gespeichert (siehe Abschnitt Datenpersistenz).

Virtueller Garten (virtualgardens.py)
Die Spielkomponente, also die virtueller Garten Anwendung wird über den Button „TO THE GARDENS“ im Hauptfenster gestartet.

Alternativ besteht auch die Möglichkeit die Anwendung manuell zu starten mit bspw. folgendem Befehl: `python virtualgardens.py`

Beim Start wird ein Pygame-Fenster geöffnet und ein Hauptmenü dargestellt mit folgenden Optionen:

- „Create Garden“: Neues Garten-Layout anlegen. Hier wählst du:
  1. Vegetationsart (z.B. City Park, Desert, Rainforest)
  2. Einen Namen für den Garten
  3. Dann wird eine leere Karte erzeugt, die du bepflanzen kannst.
- „Load Garden“: Vorhandenes .map-File aus dem Ordner map_data/ laden. Du kannst so einen bereits angelegten Garten weiterbearbeiten.
- „Back to Productivity Window“: Wechselt zurück zum Hauptfenster

Gärten und Objekte

Die Gärten sind in Kacheln unterteilt. Du hast eine Inventarleiste am unteren Bildschirmrand, in der du verschiedene Objekte (Blumen, Bäume, Wege, etc.) auswählen kannst.

Über Mausklick auf ein Kachel-Feld wird das aktuell gewählte Objekt platziert, sofern genug Punkte vorhanden sind.

Jeder Gegenstand hat Kosten (z.B. 2 Punkte für Blumen, 8 für einen großen Baum). Beim Platzieren eines Objekts werden Punkte von deinem „verfügbaren Punktestand“ abgezogen.

Tipp: Falls du nicht genug Punkte hast, um ein Objekt zu platzieren, kannst du zurück ins Hauptfenster, dir durch weitere Arbeitsminuten Punkte verdienen und erneut in den Garten wechseln.

Anwendung beenden oder wechseln:

In egal welchem Fenster kann über das Windows Fenster x-Symbol die Anwendung komplett beendet werden.

Über die zwei Buttons „TO THE GARDENS“ (Hauptfenster) oder „Back to Productivity Window“ (virtueller Garten) kann beliebig häufig zwischen den beiden Teilanwendungen gewechselt werden

Zusätzlich gelangst du in der virtueller Garten Anwendung jederzeit über die „Escape“ Taste zurück ins Hauptmenü der virtueller Garten Anwendung.

Datenpersistenz

Alle Daten werden lokal gespeichert im Ordner „resources“ und werden an keinen Server übermittelt.

Speicherung der Nutzerdaten:

Die Anwendung speichert fortlaufend folgende Daten:

- Punktestände (total_points, available_points)
- Timer-Einstellungen (z.B. Zeitwerte für Pomodoro-Phasen und normalen Timer)
- Freitext (das große Textfeld).
- Informationen der angelegten Projekte (Name, Beschreibung, Kategorie, Start-& Enddatum, gesammelte Minuten)

Speicherung der Gartendaten:

Beim Schließen eines Gartens oder der Anwendung werden folgende Daten gespeichert:

- Objekt-Positionen (in .map-Dateien)
- Garten-Metadaten (z.B. welche Vegetationsart) in gardens_data.json
- Ebenso wird der verbleibende Punktestand in deiner JSON_FILE (z.B. user_data.json) aktualisiert.

3. Risikomanagement

Potenzielle Risiken

Bei der Risikoanalyse werden potenzielle Risiken identifiziert, deren Eintrittswahrscheinlichkeit und Schadensausmaß bewertet und geeignete Gegenmaßnahmen vorgeschlagen. Die verschiedenen Risiken werden für mehr Übersichtlichkeit in Bereiche kategorisiert.

Technische Risiken

1. Leistungsprobleme bei hoher Last oder Datenmengen
    - Beschreibung: Die Anwendung könnte bei intensiver Nutzung oder Speicherung vieler Daten langsam werden, was die Benutzerfreundlichkeit beeinträchtigt.
    - Eintrittswahrscheinlichkeit: Mittel
    - Schadensausmaß: Hoch (Nutzer:innen könnten das Programm frustriert abbrechen)
    - Gegenmaßnahmen:
        - Nutzung performanter Python-Bibliotheken und Implementierung von Optimierungsmöglichkeiten (Recherche benötigt)
        - Optimierung der Datenbankzugriffe und Minimierung redundanter Prozesse.
        - Vorab Lasttests durchführen.
2. Datenverlust durch unerwartetes Beenden der Anwendung
    - Beschreibung: Fortschritte oder Einstellungen der Nutzer:innen könnten bei einem Absturz verloren gehen.
    - Eintrittswahrscheinlichkeit: Mittel
    - Schadensausmaß: Hoch
    - Gegenmaßnahmen:
        - Regelmäßige Autosave-Funktion implementieren.
        - Backup-Mechanismen für kritische Daten einbauen.

Benutzerbezogene Risiken

3. Mangelnde Akzeptanz der Benutzeroberfläche
    - Beschreibung: Die minimalistische Oberfläche oder die Visualisierung des virtuellen Gartens könnten als unattraktiv empfunden werden.
    - Eintrittswahrscheinlichkeit: Mittel
    - Schadensausmaß: Mittel
    - Gegenmaßnahmen:
        - Feedback einholen und iterativ verbessern.
4. Fehlende Motivation zur Nutzung durch schlechte Umsetzung der Gamification
    - Beschreibung: Nutzer:innen könnten die Anwendung nicht regelmäßig verwenden, da die Gamification nicht motivierend genug ist.
    - Eintrittswahrscheinlichkeit: Mittel
    - Schadensausmaß: Hoch
    - Gegenmaßnahmen:
        - Belohnungssystem abwechslungsreich gestalten (z. B. freischaltbare Inhalte, Fortschrittsanzeige).

- Regelmäßige Erfolgserlebnisse integrieren, die den Nutzer:innen einen Nutzen aufzeigen.

Sicherheitsrisiken

5. Datenschutzverletzungen
   - Beschreibung: Speicherung personenbezogener Daten könnte Sicherheitslücken enthalten oder DSGVO-Vorgaben nicht vollständig erfüllen.
   - Eintrittswahrscheinlichkeit: Niedrig
   - Schadensausmaß: Sehr hoch (rechtliche Konsequenzen, Vertrauensverlust)
   - Gegenmaßnahmen:
     - Speicherung ausschließlich lokal durchführen.
     - Anonyme Datenerfassung sicherstellen.
     - Code- und Sicherheits-Audits vor Veröffentlichung.

Projektbezogene Risiken

6. Zeitüberschreitung oder Nichterfüllung von Anforderungen
   - Beschreibung: Das Projekt könnte mehr Zeit als geplant in Anspruch nehmen oder einige Features könnten nicht rechtzeitig fertiggestellt werden.
   - Eintrittswahrscheinlichkeit: Hoch
   - Schadensausmaß: Hoch
   - Gegenmaßnahmen:
     - Projektplanung in Milestones aufteilen.
     - Priorisierung der wichtigsten Features vornehmen.
7. Unerwartete technische Schwierigkeiten (z. B. mit Python-Bibliotheken)
   - Beschreibung: Verwendete Bibliotheken könnten Bugs enthalten oder nicht die gewünschten Funktionen bieten.
   - Eintrittswahrscheinlichkeit: Mittel
   - Schadensausmaß: Mittel
   - Gegenmaßnahmen:
     - Vorab sorgfältige Auswahl der Bibliotheken treffen.
     - Alternativen evaluieren und Backup-Lösungen bereithalten.

Bewertungstabelle

Tabelle 1 fasst die identifizierten Risiken zusammen, indem sie diese nach Eintrittswahrscheinlichkeit und Schadensausmaß kategorisiert. Auf Basis dieser Bewertung wird die Priorität der Maßnahmen abgeleitet, um die wichtigsten Risiken gezielt zu adressieren.

Risiko	**Eintritts-wahrscheinlichkeit**	**Schadensausmaß**	**Maßnahmenpriorität**
Leistungsprobleme	Mittel	Hoch	Hoch
Datenverlust	Mittel	Hoch	Hoch
Mangelnde Akzeptanz der UI und Visualisierung	Mittel	Mittel	Mittel
Fehlende Nutzungsmotivation	Hoch	Hoch	Hoch
Datenschutzverletzungen	Niedrig	Sehr hoch	Hoch
Nutzer-Manipulation (Cheating)	Mittel	Mittel	Mittel
Zeitüberschreitung	Hoch	Mittel	Hoch
Technische Schwierigkeiten	Mittel	Mittel	Mittel

Tabelle 1 – Risikoanalyse-Bewertungstabelle

4. Zeitplanung

Bei der Zeitplanung sind einige Kriterien zu berücksichtigen. Zum einen ist es schwierig, den Aufwand für die einzelnen Aufgaben im Voraus abzuschätzen, zum anderen ist es schwierig abzuschätzen, wie viel Zeit an welchen Tagen zur Verfügung steht. Daher ist es wichtig, die Zeitplanung so zu gestalten, dass der aktuelle Fortschritt ständig mit den Zielen verglichen werden kann und gegebenenfalls kleine Anpassungen vorgenommen werden können.

Deswegen wird die Projektzeitplanung kontinuierlich mithilfe eines Gantt-Diagramms überwacht, das zugleich zur Fortschrittskontrolle dient. Dieses ist in Abbildung 1 zu sehen.

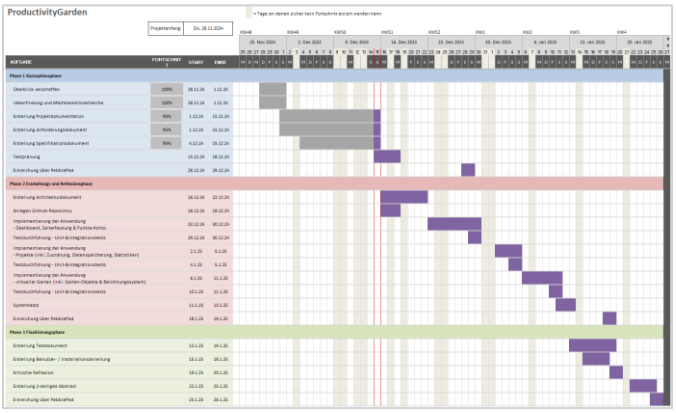

Abbildung 1 - Gantt-Diagramm

Stand 25.01.2025:

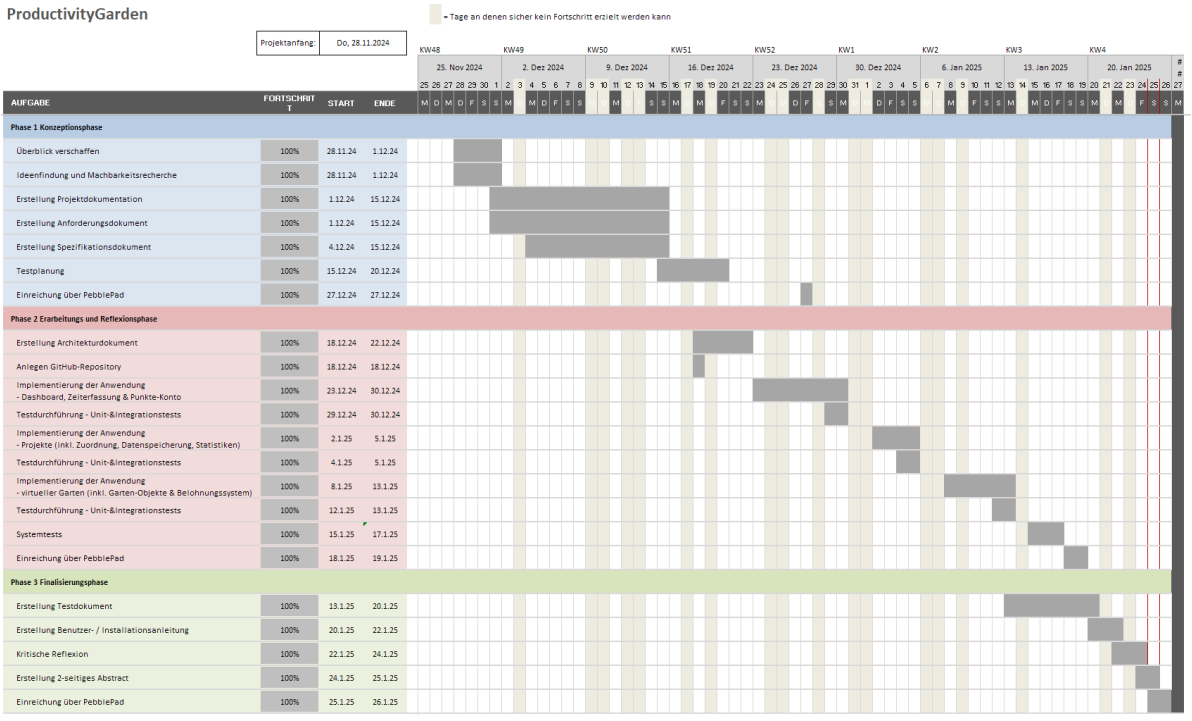

Abbildung 2 -Gantt-Diagramm aktualisiert

5. Literaturverzeichnis

Huseynli, B., & Uslu, A. (2024). A Qualitative Study on the Definition and Concept of Gamification. *Journal of Economic Sciences: Theory & Practice*, *81*(1), (S. 40–50). https://doi.org/10.61640/jestp.2024.81.01.03

Matallaoui, A., Hanner, N., & Zarnekow, R. (2016). Introduction to Gamification: Foundation and Underlying Theories. In S. Stieglitz, C. Lattemann, S. Robra-Bissantz, R. Zarnekow, & T. Brockmann, *Gamification: Using Game Elements in Serious Contexts* (S. 3–18). Springer International Publishing AG. http://ebookcentral.proquest.com/lib/badhonnef/detail.action?docID=4710247

Anforderungsdokument

Projektname: ProductivityGarden

Name	Jonas Huber
Matrikelnummer	IU14085128
Modul	Projekt: Software Engineering DLMCSPSE01_D
Datum	17.01.2025

Inhalt

Abbildungsverzeichnis

1. Zielgruppen

Kundennutzen

- **Verbesserung der Konzentration:** Durch den Pomodoro-Timer, der gezielte Arbeitseinheiten und Pausen fördert.
- **Zeitmanagement:** Flexibel einstellbare Timer und Stoppuhren ermöglichen das Tracking von Aktivitäten.
- **Motivation**: Belohnungen im virtuellen Garten fördern die Motivation
- **Stressabbau und Kreativität:** Der virtuelle Garten sorgt für eine angenehme Atmosphäre.
- **Individuelle Anpassung:** Nutzer:innen können Projekte zuordnen, Zeit aufzeichnen und ihren Garten nach ihren Vorlieben gestalten.

Abgrenzung zu ähnlichen Anwendungen

- **Simpel und effektiv:** Keine Formen von Anwendungen-Bindung und Gewinnerzielung wie Paywalls, Werbung oder tägliche Herausforderungen.
- **Gamification:** Fokus auf Belohnungen und Visualisierung statt auf strikte Prokrastination-Sperren oder Disziplinierung.
- **Breiter Anwendungsbereich:** Geeignet für Arbeit, Lernen, Hobbys oder Entspannung.

Primäre Zielgruppe

Die primäre Zielgruppe umfasst Studierende, Berufstätige und allgemein alle Personen, die ihre Produktivität steigern wollen und gleichzeitig eine spielerische Motivation schätzen mit entspannenden Gamification-Elementen. Junge Erwachsene und Studierende haben oft Herausforderungen im Zeitmanagement und Stressabbau und benötigen Motivation für Studium oder Arbeit, wofür sie gerne einfache und unterhaltsame Tools ohne Paywalls und Werbung nutzen. Berufstätige dagegen (insbesondere Wissensarbeiter) wollen oft ihre Produktivität steigern und Projektzeiten erfassen, ohne viel Aufwand und aufdringliche Anwendungen mit Werbung.

Beispiel-Zielgruppenprofil:

- Sophie, 22 Jahre – Studentin
- Lebensstil: Sophie jongliert Studium, Nebenjob und Freizeitaktivitäten.
- Verwendet die Anwendung, um Lernphasen produktiv zu gestalten und dabei ein wenig Stress wieder abzubauen.
- Sie schätzt die visuelle Belohnung und die Einfachheit.
- Hat wenig Geduld für Anwendungen mit Premium-Elementen und Werbung.

Sekundäre Zielgruppe

Die Anwendung kann auch für Eltern, Teilzeitbeschäftigte oder generell für alle Personen nützlich sein, die mit Hilfe der Zeiterfassung einen besseren Überblick über ihre Zeit gewinnen möchten. Diese können die flexiblen Timer und die Zeiterfassung für Projekte, Hobbys oder

Haushaltstätigkeiten nutzen. Die Motivation wird auch durch die spielerischen Gamification-Elemente gefördert und erfordert weder eine hohe Technikaffinität noch viel Aufwand.

Beispiel-Zielgruppenprofil:

- Julian, 40 Jahre – Hobby-Bäcker und Vater
- Lebensstil: Arbeitet halbtags und widmet seine Nachmittage mit seinem Hobby Backen und der Organisation des Haushalts
- Er verliert schnell den Überblick über die Zeit, die er in verschiedene Aufgaben investiert und verwendet deshalb die Anwendung, um einen Überblick über ihr Zeitmanagement zu bekommen
- Er schätzt den entspannenden Aspekt der Pflanzenvisualisierung und nutzt flexible Timer oder die Stoppuhr für sein Hobby oder Hausarbeiten.
- Schätzt ebenfalls, dass die Anwendung nicht überladen ist und flexibel genutzt werden kann.

2. Funktionale Anforderungen

Im Folgenden erfolgt eine detaillierte Auflistung der gewünschten Funktionen und Features der Anwendung. Die Anforderungen werden dabei zunächst textuell durch User Stories beschrieben und anschließend durch ein UML-Use-Case-Diagramm visualisiert.

User-Storys

Verbesserung der Konzentration und Produktivität

1. Als Nutzer:in möchte ich einen Pomodoro-Timer verwenden, damit ich meine Arbeitsphasen und Pausen strukturiert mit einem festen Rhythmus einteilen kann.
2. Als Nutzer:in möchte ich die Länge von Produktiv- und Pausenzeiten anpassen können, damit der Timer meinen individuellen Bedürfnissen entspricht.
3. Als Nutzer:in möchte ich einen individuell einstellbaren Timer für meine nächste Produktivphase verwenden, ohne feste Pausenzeit und Rhythmus.
4. Als Nutzer:in möchte ich die Anwendung ohne störende Benachrichtigungen nutzen können, damit ich fokussiert bleiben kann.
5. Als Nutzer:in möchte ich eine einfache und intuitive Navigation, damit ich ohne großen Aufwand alle Funktionen nutzen kann.

Zeitmanagement

6. Als Nutzer:in möchte ich eine Stoppuhr verwenden, um flexibel Aktivitäten zu tracken, ohne an feste Zeitintervalle gebunden zu sein.
7. Als Nutzer:in möchte ich Zeiten manuell eintragen können, falls ich vergessen habe, den Timer oder die Stoppuhr zu starten.
8. Als Nutzer:in möchte ich eine Übersicht über meine produktiv genutzte Zeit erhalten, damit ich meinen Fortschritt nachvollziehen kann.
9. Als Nutzer:in möchte ich meine aufgezeichnete Zeit spezifischen Projekten oder Kategorien zuordnen können, um meine Produktivität für unterschiedliche Aufgaben zu verfolgen.

10. Als Nutzer:in möchte ich eine Visualisierung über die aufgewendete Zeit für verschiedene Projekte oder Kategorien sehen, mitunter um diese miteinander vergleichen zu können.

Motivation

11. Als Nutzer:in möchte ich für meine aufgezeichnete produktive Zeit virtuelle Belohnungen erhalten, damit ich motiviert bleibe, kontinuierlich produktiv zu sein.
12. Als Nutzer:in möchte ich sehen, wie mein Garten wächst und sich entwickelt, basierend auf der Zeit, die ich produktiv genutzt habe.
13. Als Nutzer:in möchte ich meinen virtuellen Garten mit verschiedenen Pflanzen und Dekorationen gestalten, um ein visuelles Feedback für meine Fortschritte zu haben.

Stressabbau und Kreativität

14. Als Nutzer:in möchte ich eine beruhigende und minimalistische Benutzeroberfläche haben, damit die Anwendung mich nicht von meiner Arbeit ablenkt.
15. Als Nutzer:in möchte ich, dass mein virtueller Garten eine entspannte Atmosphäre bietet, um Stress während meiner Pausen abzubauen.
16. Als Nutzer:in möchte ich in meiner Pausenzeit nicht tiefer in komplexe Spielmechaniken eintauchen und mich mit verschiedenen Inhalten auseinandersetzen müssen, sondern die Anwendung ohne viel Aufwand verwenden.
17. Als Nutzer:in möchte ich die Ästhetik meines Gartens individuell anpassen können, um meine Kreativität auszuleben.

Abgrenzung zu ähnlichen Anwendungen

18. Als Nutzer:in möchte ich auf Paywalls verzichten können, um die Anwendung ohne Einschränkungen zu nutzen.
19. Als Nutzer:in möchte ich keine täglichen Aufgaben oder Challenges vorgegeben bekommen, um mich nicht unter Druck gesetzt zu fühlen.
20. Als Nutzer:in möchte ich Belohnungen basierend auf meiner echten Produktivität erhalten, anstatt durch kostenpflichtige Optionen bevorzugt zu werden.
21. Als Nutzer:in möchte ich keine visuellen Reize haben und nicht durch Werbung und überladene Inhalte abgelenkt werden.

Use-Case-Diagramm

Das Use-Case-Diagramm (siehe Abbildung 1) veranschaulicht die Hauptinteraktionen der Nutzer:innen mit der Anwendung "ProductivityGarden". Es umfasst die verschiedenen Bereiche mit Zeiterfassung (z. B. Pomodoro-Timer), Zeitmanagement (z. B. Zuordnung von Produktivzeiten zu Projekten) und den virtuellen Gärten (z. B. Pflanzen und Dekorationen kaufen). Die Diagrammstruktur zeigt die einfache und zielgerichtete Benutzerführung der Anwendung.

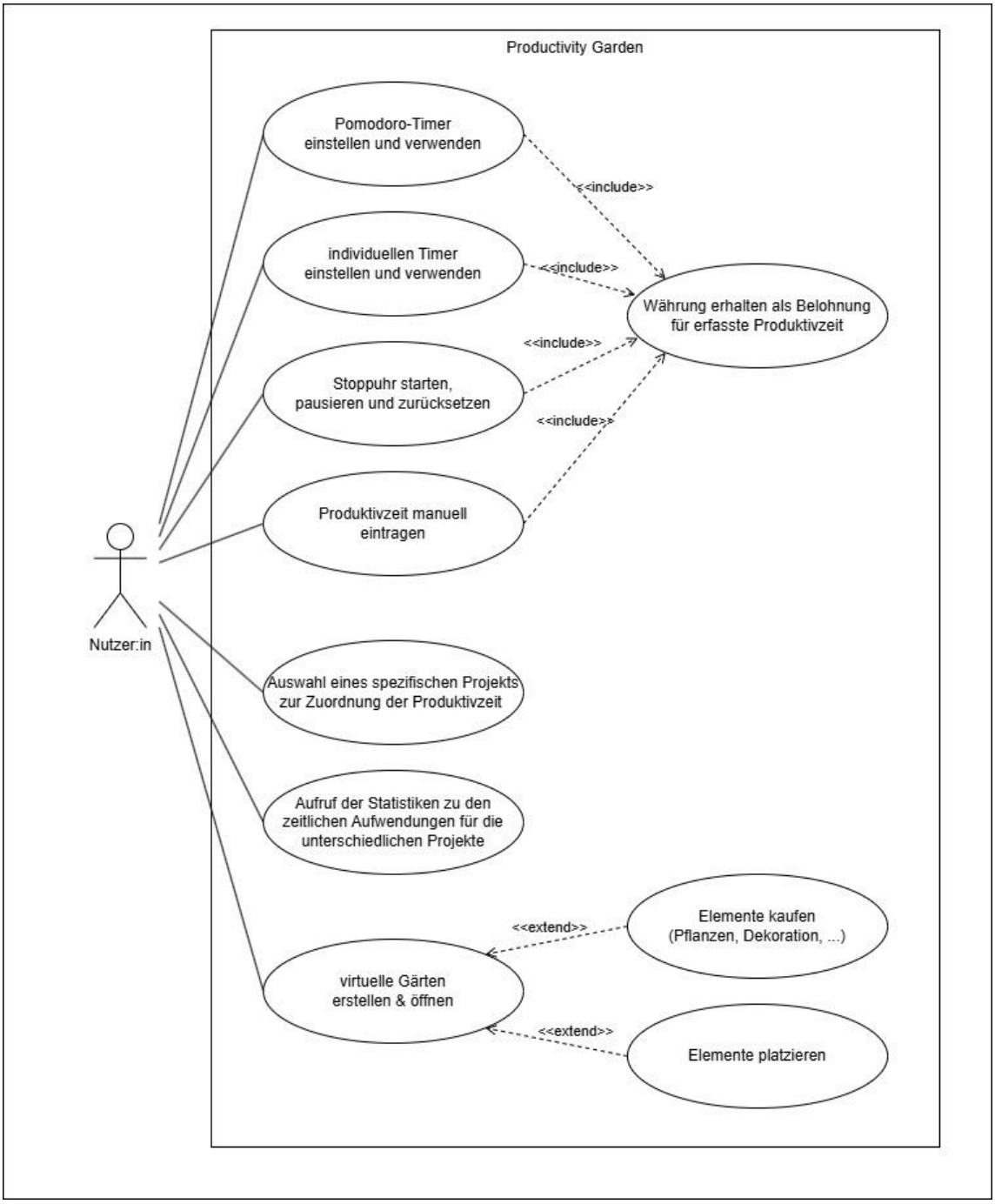

Abbildung 1 - UML-Use-Case-Diagramm

3. Nicht-funktionale Anforderungen

In diesem Kapitel werden die maßgeblichen Qualitätsanforderungen dargelegt, wie beispielsweise Leistung, Sicherheit und Benutzerfreundlichkeit.

Leistung und Ausführung der Anwendung

Die Anwendung soll so konzipiert werden, dass sie schnell und reaktionsfreudig arbeitet, was besonders wichtig ist, da es sich im Kern um ein Spiel handelt. Die Anwendung soll auf dem Betriebssystem Windows lauffähig sein und ohne zusätzliche Installationsschritte direkt verwendet werden können. Um den Nutzerkomfort zu erhöhen, bleibt die Anwendung auch im Hintergrund aktiv, sodass der Spielverlauf nicht unterbrochen wird, wenn der Nutzer zu einer anderen App wechselt. Darüber hinaus speichert die Anwendung relevante Daten, sodass der aktuelle Stand beim erneuten Starten nahtlos fortgesetzt werden kann. Diese Eigenschaften gewährleisten eine konsistente und angenehme Benutzererfahrung.

Benutzerschnittstelle und Benutzerfreundlichkeit

Bei der Gestaltung der Benutzerschnittstelle liegt der Fokus auf intuitiver und einfacher Bedienung, sodass Nutzer schnell und problemlos auf alle Funktionen zugreifen können. Die Oberfläche soll bewusst minimalistisch gehalten werden, um Ablenkungen und unnötige visuelle Reize zu vermeiden, sodass sich die Nutzer ganz auf das Spielerlebnis konzentrieren können. Darüber hinaus verzichtet die Anwendung vollständig auf Premium-Elemente, Paywalls, Werbung oder Wartezeiten, um eine uneingeschränkt zugängliche und störungsfreie Nutzung zu ermöglichen.

Sicherheit

Für die Sicherheit und den Schutz der Nutzerdaten werden alle relevanten Datenschutzbestimmungen eingehalten, einschließlich der Richtlinien der Datenschutz-Grundverordnung (DSGVO). Dies gewährleistet, dass personenbezogene Daten sicher verarbeitet und gespeichert werden. Es ist zu gewährleisten, dass keinerlei Daten an einen Server übermittelt werden müssen, sondern ausschließlich auf der ausführenden Instanz in privater Form gespeichert werden.

4. Glossar

Gamification	auch Spielifizierung genannt, bezeichnet die Anwendung spielerischer Elemente und Mechanismen in einem nicht spielerischen Kontext, z.B. im Bildungsbereich (Huseynli, 2024, S. 45).
Paywall	Eine Paywall ist eine digitale Schranke, die Inhalte in Anwendungen nur gegen Bezahlung zugänglich macht, z.B. durch ein Abonnement oder eine einmalige Zahlung (Ryte, 2021).
Pomodoro-Timer	Es handelt sich um eine Methode des Zeitmanagements. Es wird abwechselnd 25 Minuten konzentriert gearbeitet und 5 Minuten Pause gemacht (Pairan, 2024).

5. Literaturverzeichnis

Huseynli, B., & Uslu, A. (2024). A Qualitative Study on the Definition and Concept of Gamification. *Journal of Economic Sciences: Theory & Practice*, *81*(1), (S. 40–50). https://doi.org/10.61640/jestp.2024.81.01.03

Pairan, M. (2024, Oktober 28). Pomodoro-Technik: Mehr schaffen in 25 Minuten? *Gesunde Produktivität - Der Blog*. https://www.marapairan.de/pomodoro-technik/

Ryte. (2021). Was ist eine Paywall? *Ryte Wiki*. https://de.ryte.com/wiki/Paywall/

Spezifikationsdokument

Projektname: ProductivityGarden

Name	Jonas Huber
Matrikelnummer	IU14085128
Modul	Projekt: Software Engineering DLMCSPSE01_D
Datum	24.01.2025

Inhalt

Abbildungsverzeichnis

1. Datenmodell

Dieses Kapitel beinhaltet eine textuelle Beschreibung der zentralen Geschäftsobjekte der Anwendung und deren Beziehungen zueinander sowie einer Visualisierung in Form eines UML-Klassendiagramms.

Textuelle Beschreibung

1. Nutzer:innen
   Nutzer:innen interagieren mit der Anwendung: Sie erstellen Projekte, messen Produktivzeiten, sammeln Punkte, erstellen Gärten und kaufen Elemente (Pflanzen und Dekoration).

2. Projekt
   Ein Projekt steht für eine Aufgabe oder Aktivität, der die Nutzer:innen Zeit widmen.

3. Sitzung
   Eine Sitzung repräsentiert eine Zeitspanne, die die Nutzer:innen für ein bestimmtes Projekt gearbeitet hat und somit dafür eine gewisse Anzahl an Punkten verdient.

4. Punkte-Konto
   Das Punkte-Konto speichert die erforderlichen Daten für das Punktesystem. Es kann nur ein Punkte-Konto geben und dieses ist explizit zu den Nutzer:innen zugewiesen.

5. Virtueller Garten
   Ein virtueller Garten steht für eine Visualisierte Spielwelt, in der Inhalte, sogenannte Garten-Objekte, platziert werden. Nutzer:innen können mehrere virtuelle Gärten erstellen.

6. Garten-Objekt
   Ein Garten-Objekt repräsentiert ein Element, beispielsweise eine Pflanze oder Dekoration, das im Gegenzug zu Punkten in einem virtuellen Garten platziert wird.

Visualisierung durch UML-Klassendiagramm

Die Visualisierung in Abbildung 1 zeigt die zentralen Geschäftsobjekte der Anwendung, ihre Attribute und die Beziehungen untereinander.

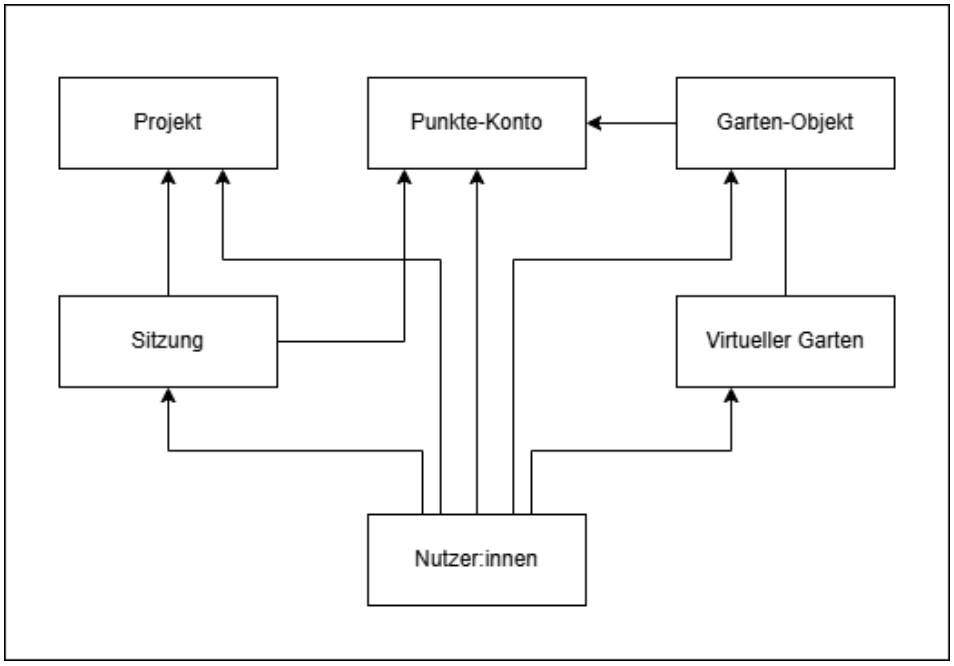

Abbildung 1 – Datenmodell UML-Klassendiagramm

Notiz (#1-01/25):

„Nutzer“ Klasse wurde in der Architekturentwicklung wieder rausgeworfen und für irrelevant empfunden, da nur eine Person die Anwendung verwendet und es keine Profil-/Nutzerverwaltung gibt.

2. Geschäftsprozesse

Im vorliegenden Kapitel erfolgt eine Darlegung der zentralen Funktionen und Prozesse der Anwendung. Zunächst erfolgt eine textuelle Darlegung und anschließend wird ein ausgewählter Kernprozess anhand eines UML-Aktivitätsdiagramms visualisiert.

Textuelle Beschreibung

1. Timer-Funktionalitäten
   Nutzer:innen können zwischen verschiedenen Zeiterfassungsmethoden wählen:
   - Pomodoro-Timer: Die Nutzer:innen arbeiten in Intervallen, z. B. 25 Minuten Arbeit und 5 Minuten Pause.
   - Flexibler Timer: Die Nutzer:innen legen individuelle Zeitspannen fest.
   - Stoppuhr: Startet und stoppt die Zeit flexibel, ohne festen Rahmen.

   Nach Ablauf des Timers oder beenden der Stoppuhr erhalten die Nutzer:innen Punkte
   Notiz (#2-01/25):
   Nutzer:innen bekommen bereits während des Laufens der Timer / Stoppuhr Punkte, hauptsächlich aus Persistenz-Gründen. Die Nutzer:innen könnten früher als ursprünglich geplant fertig sein und falls die Anwendung aus Versehen geschlossen wird (zum Beispiel auch beim Wechsel auf zum „VirtualGarden“), wurden trotzdem die Punkte bereits gutgeschrieben.

2. Punktesystem
   Punkte werden basierend auf produktiv verbrachter Zeit gesammelt (z.B. 1 Punkt pro 10 Minuten). Punkte können verwendet werden, um Objekte für einen virtuellen Garten zu kaufen

3. Projekt-Tracking
   Nutzer:innen können ihre Zeit einem bestimmten Projekt zuweisen. Produktivzeiten werden für die jeweiligen Projekte in einer Statistik erfasst und visualisiert.

4. Virtuelle Gärten
   Nutzer:innen haben die Möglichkeit, virtuelle Gärten zu erstellen. Nach der Erstellung umfasst ein solcher Garten lediglich ein leeres Raster, auf dem Objekte hinzugefügt werden können.

5. Datenspeicherung und Datenschutz
   Alle Daten werden lokal auf dem Gerät der Nutzer:innen gespeichert und es erfolgt keine Übertragung an externe Server. Nutzerdaten wie Punkte, virtuelle Gärten und deren Objekte und Timer-Einstellungen werden bei jedem Start der App wiederhergestellt. Diese Daten werden während der Ausführung der Anwendung automatisch gespeichert.
6. Hintergrundnutzung
   Die Anwendung läuft im Hintergrund weiter, sodass andere Anwendungen parallel genutzt werden können.

Visualisierung durch UML-Aktivitätsdiagramm

Zur Veranschaulichung wird einer der Kernprozesse mittels eines UML-Aktivitätsdiagramms visualisiert. In Abbildung 2 ist ein Teil der Timer-Funktionalitäten dargestellt, wobei es sich um die Pomodoro-Timer-Funktionalität handelt.

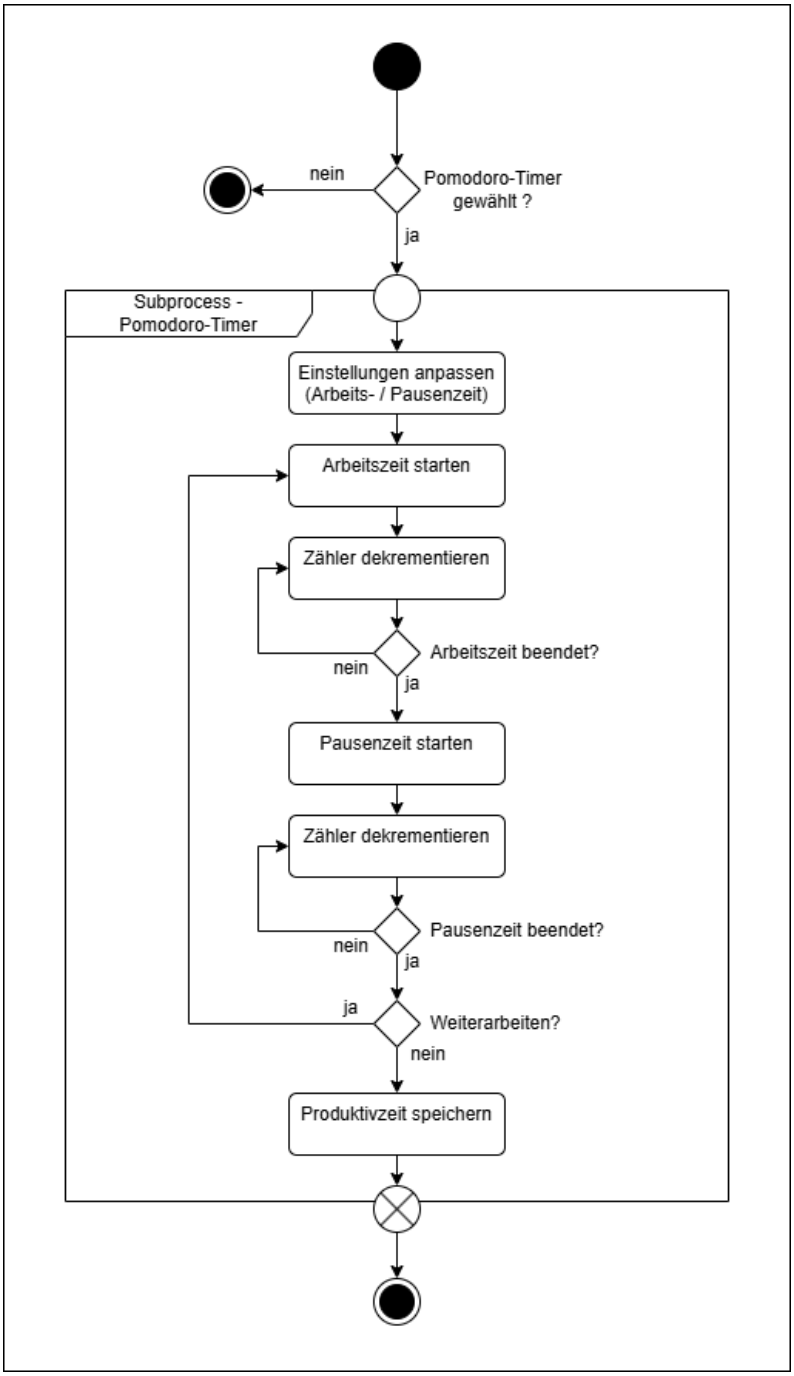

Abbildung 2 – UML-Aktivitätsdiagramm zu Prozess „Pomodoro-Timer“

3. Geschäftsregeln

In folgendem Abschnitt werden die zentralen Geschäftsregeln für die Anwendung ProductivityGarden definiert. Diese Regeln dienen als Grundlage für die Gestaltung der Geschäftsobjekte und -prozesse und stellen sicher, dass die Funktionen der Anwendung sowohl den Nutzeranforderungen als auch den Projektzielen entsprechen. Durch die Festlegung dieser verbindlichen Richtlinien wird ein reibungsloser Ablauf sowie eine intuitive und zuverlässige Nutzung der Anwendung gewährleistet.

Geschäftsregeln für Geschäftsobjekte

1. Zeiterfassung
   - Der Timer muss eine Mindestzeit von 1 Minute und eine Höchstzeit von 24 Stunden unterstützen.
   - Beim Pomodoro-Timer sind standardmäßig 25 Minuten Arbeit und 5 Minuten Pause eingestellt, die Zeiten können jedoch von Nutzer:innen individuell angepasst werden.
   - Die Timer müssen jederzeit pausiert und beendet werden können
   - Die Stoppuhr muss es ermöglichen, dass deren Betrieb jederzeit pausiert, wieder aufgenommen und beendet werden kann.

2. Punktesystem
   - Nutzer:innen erhalten 1 Punkt für jede 10 Minuten produktiver Zeit
   - Punkte können ausschließlich innerhalb der Anwendung für den Kauf von virtuellen Objekten eingesetzt werden
   - Es ist nicht möglich, Punkte in der Anwendung zu „verlieren“ oder abgezogen zu bekommen

3. Projekt-Tracking
   - Jede produktiv verbrachte Zeit muss einem Projekt oder einer Aufgabe zugeordnet werden
   - Projekte können gelöscht oder archiviert werden, wenn dies von Nutzer:innen gewünscht ist
     Notiz (#3-01/25):
     Projekte können nicht archiviert werden. Feature aus zeitlichen Gründen nicht umgesetzt.
   - Das Enddatum des Projekts darf leer gelassen werden, darf jedoch niemals vor dessen Startdatum liegen

4. Virtueller Garten
   - Nutzer:innen können beliebig viele Gärten erstellen
   - Ein virtueller Garten muss ein Raster enthalten, auf dem Objekte platziert werden können
   - Ein Objekt darf nur in einem Garten platziert werden, wenn der Nutzer:innen über genügend Punkte verfügt

- Ein Objekt kann nur platziert werden, wenn es in das gewählte Raster passt und dieses nicht blockiert ist durch ein bereits platziertes Objekt
  Notiz (#4-01/25):
  Aus zeitlichen Gründen konnte nicht das Bewegen von bereits platzierten Objekten implementiert werden. Wenn nun ein Objekt also platziert wurde, würde es laut dieser Regel für immer da platziert sein. Da aber eventuell ein anderes Objekt platziert werden möchte an genau dieser Stelle, wird das bisherige Objekt aktuell noch überschrieben.

Geschäftsregeln für Geschäftsprozesse

1. Zeiterfassungsprozesse
   - Eine laufende Zeiterfassung darf nicht verändert werden. Änderungen können erst nach der Beendigung oder dem Abbrechen der Zeiterfassung vorgenommen werden.
   - Pomodoro-Timer: Nach Ablauf der eingestellten Zeit müssen die Nutzer:innen eine Aktion wählen (z.B. „neuen Zyklus starten“), bevor ein neuer Zyklus beginnt.

2. Punktevergabeprozess
   - Punkte werden nur bei abgeschlossenen Zeiterfassungen gutgeschrieben. Dazu zählen Timer und Stoppuhren deren Zeit abgelaufen ist oder die explizit „beendet“ wurden.
     Siehe Notiz #2-01/25
   - Beim Kauf eines Objekts, werden die entsprechenden Punkte sofort vom Kontostand abgezogen

3. Datenmanagementprozess
   - Alle Nutzerdaten (z.B. Punkte, Timer-Einstellungen, virtuelle Gärten) müssen lokal und automatisch gespeichert werden.
   - Beim Start der Anwendung müssen die zuletzt gespeicherten Daten geladen werden
   - Es ist sicherzustellen, dass keine personenbezogenen Daten an Dritte weitergegeben werden oder auf externen Servern gespeichert werden

4. Nutzerinteraktionsprozesse
   - Die Benutzeroberfläche muss sicherstellen, dass Benutzeraktionen wie das Löschen von Gärten oder Projekten oder sonstigen Daten eine Bestätigung erfordern, um versehentliche Löschungen zu vermeiden.
   - Jede Aktion, die eine irreversible Auswirkung hat (z. B. Kauf von Objekten), muss den Nutzer:innen vorher klar kommuniziert werden.

4. Systemschnittstellen

In diesem Kapitel werden die technischen Schnittstellen der Anwendung beschrieben. Dazu gehören die genutzten Protokolle, Datenformate und Mechanismen, die für die lokale Speicherung sowie die Integration mit dem Betriebssystem Windows erforderlich sind.

Datenformat und Speicherstruktur

JSON eignet sich ideal für die flexible Speicherung strukturierter Daten wie Timer-Einstellungen und Punktestände. Für eine komplexere Datenverwaltung, beispielsweise die Verknüpfung von Projekten mit Zeitdaten, bietet sich SQLite als effiziente relationale Datenbanklösung an.

Systemintegration

Um Benachrichtigungen und Interaktionen innerhalb des Systems zu implementieren, wie z.B. Toast-Benachrichtigungen bei Ablauf des Timers, können Systemaufrufe an die Windows-API gesendet werden. Python-Bibliotheken wie „win10toast“ oder „plyer“ bieten hierfür praktische Lösungen.

Notiz #5-01/25:
Da das Fehlen dieses Features nicht die „Funktionalität“ beeinträchtig, wurde es aus zeitlichen Gründen nicht umgesetzt.

5. Benutzerschnittstelle

Um eine Benutzeroberfläche für die Anwendung zu definieren, werden die Anforderungen in drei Kategorien gegliedert:

Struktur und Inhalt der wichtigsten Dialoge

Die Anwendung besteht aus verschiedenen Fenstern (Dialogen), deren Struktur und Inhalt definiert werden.

Hauptfenster
Inhalt:

- Timer-Bereich:
  - Pomodoro-Timer mit voreingestellten 25-5 Minuten, anpassbar
  - Timer und Stoppuhr, individuell konfigurierbar
- Punkteanzeige: Anzahl gesammelter Punkte insgesamt und verfügbare Punkte
- Projekte-Bereich:
  - Auswahl eines Projekts über ein Drop-Down-Menü
  - Erstellung eines Projekts
  - Übersicht über relevante Infos des aktuell ausgewählten Projekts wie gesamt verbrachte Produktivzeit, Startdatum und Enddatum (optional)
- Notizen-Bereich:
  - Ein Bereich mit einem Textfeld in den Notizen und Gedanken geschrieben werden können.
- Navigation: Tabs oder Buttons zu „Virtuelle Gärten“, „Einstellungen“ und „Statistiken“

Aufbau:

- Minimalistisch mit klaren, ablenkungsfreien Designs
- Aufteilung in verschiedene Bereiche:
  - Punkteanzeige
  - Timer-Bereich
  - Projekte-Bereich
  - Notizen-Bereich
- Anzeige von Fortschrittbalken (z.B. für die verbleibende Zeit im Timer)

Mockups:

Zur Visualisierung und damit zum besseren Verständnis der oben genannten Punkte wurde ein Mockup erstellt, das in Abbildung 3 dargestellt ist. Dieses ist ein reiner Prototyp, der nur den Ansatz des Aussehens des Hauptfensters zeigen soll. Die beiden KI-generierten Bilder in Abbildung 4 und Abbildung 5 können als weitere Inspiration z.B. für die Gestaltung der Oberfläche oder bestimmter Elemente dienen.

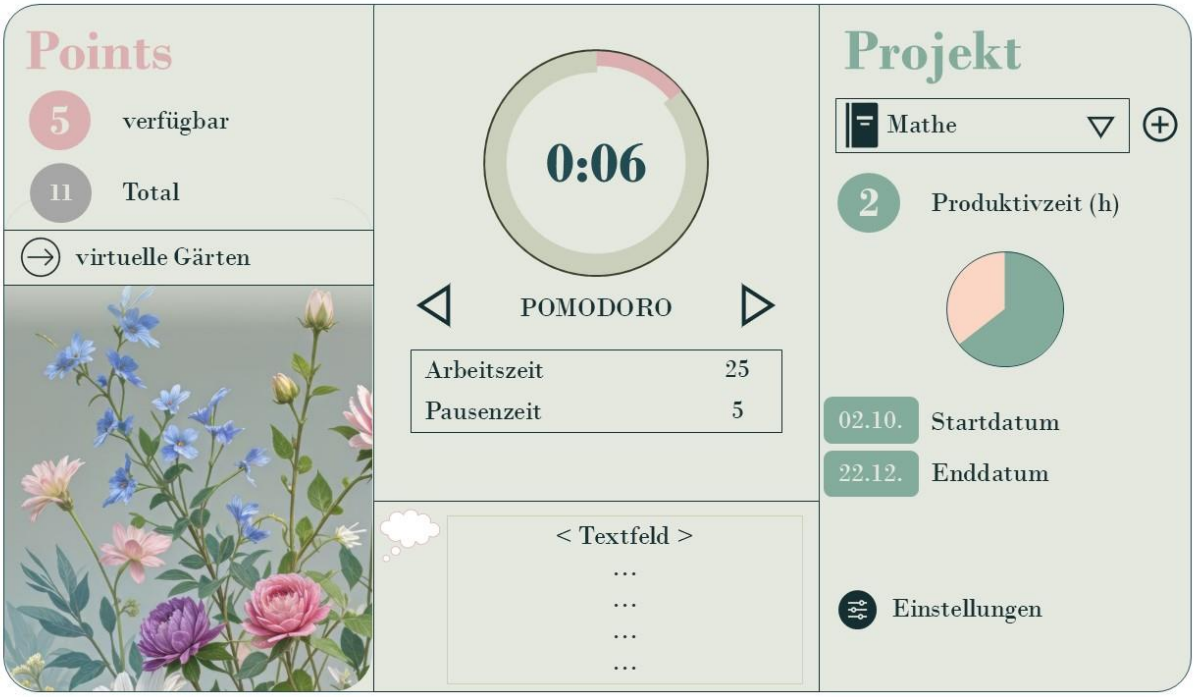

Abbildung 3 – Mockup Hauptfenster

Abbildung 4 – KI-generiertes Mockup Nr.1

Abbildung 5 – KI-generiertes Mockup Nr.2

Virtueller Garten
Inhalt:

- Übersicht der angelegten Gärten
- Ansicht eines expliziten Gartens
    - Raster, auf das Objekte (Pflanzen und Dekorationen) hinzugefügt werden können
    - Menü mit Objektauswahl

Aufbau:

- Übersicht der Gärten zeigt die Vorschau eines Gartens im Vordergrund und eine Schaltfläche zum Durchklicken der Gärten
- Button zum Anlegen eines neuen Gartens
    - Aufforderung zur Bearbeitung der Informationen des neuen Gartens:
        - Name
        - Vegetation
- Nach Auswahl eines Gartens verschwindet die Übersicht der Gärten
    - Fokus auf den ausgewählten Garten
    - Buttons, um wieder zur Übersicht der Gärten oder zum Dashboard zu kommen
- Auswahlbereich mit verschiedenen Objekten zur Platzierung unten (nur wenn Fokus auf einen expliziten Garten)
- Punkteanzeige

Mockups:

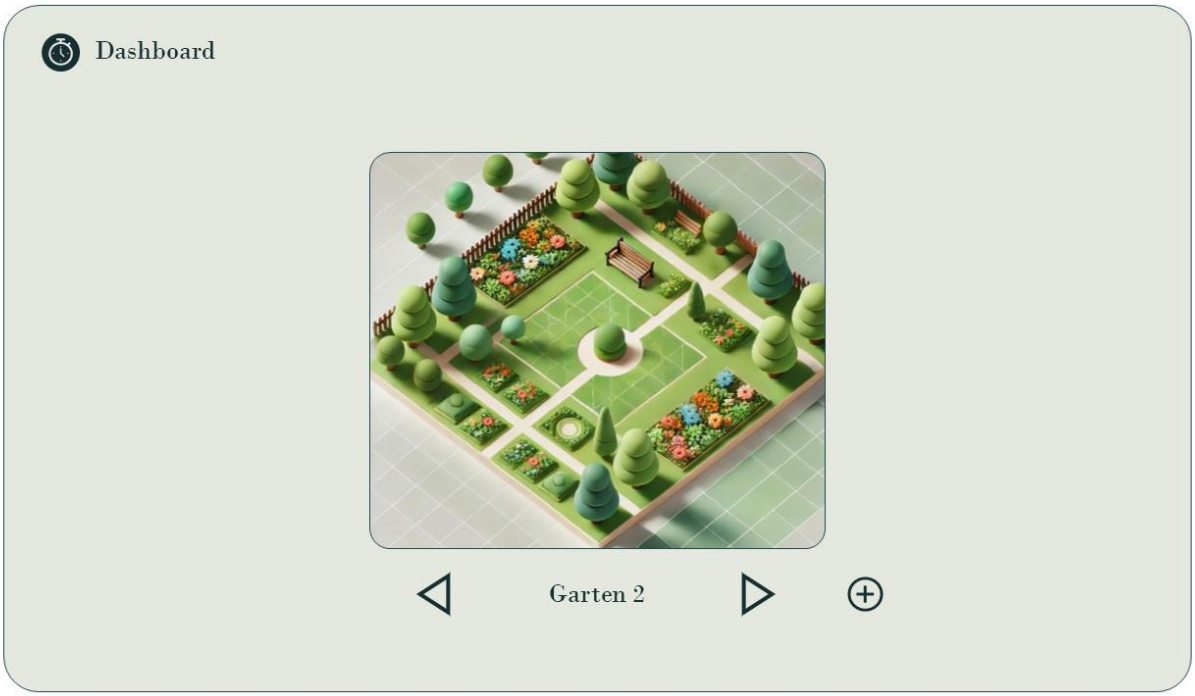

Abbildung 6 – Mockup Übersicht über die virtuellen Gärten

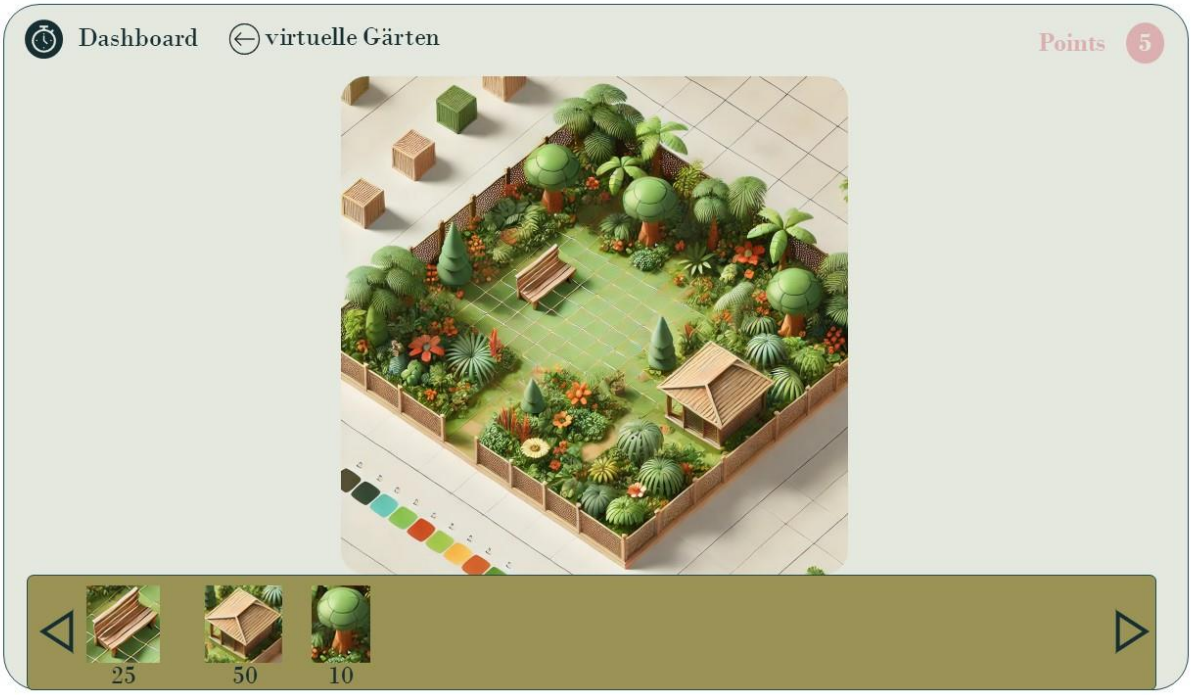

Abbildung 7 – Mockup virtueller Garten

Einstellungen
Inhalt:

- Möglichkeiten zum Löschen von Daten:
  - Alle Daten löschen
  - Explizite Projekte löschen
  - Explizite virtuelle Gärten löschen

Dialogflüsse

Dialogflüsse beschreiben die Abfolge von Interaktionen zwischen den Nutzer:innen und der Anwendung, insbesondere darauf, wie verschiedene Dialoge (z. B. Fenster, Pop-ups oder Bildschirme) miteinander verbunden sind und wie Nutzer:innen durch diese Dialoge navigieren können.

Hauptfenster
(Startbildschirm)

Timer-Bereich

- Klick auf Uhr -> Starten eines Timers oder Stoppuhr
- Klick auf Uhr während läuft -> Pausieren des laufenden Timers / der Stoppuhr
- Klick auf Pfeilsymbole unter der Uhr -> Auswahl eines Timers oder Stoppuhr
- Klick auf Zahlen „Arbeitszeit“, „Pausenzeit“, … -> Bearbeitung der Laufzeit der Timer

Projekt-Bereich

- Klick auf Projektmenü -> Auswahl eines Projekts
- Klick auf Plussymbol neben dem Projektmenü -> Erstellung eines Projekts

Textfeld

- Klick auf Textfeld im unteren Bereich -> Möglichkeit zur Bearbeitung des Texts

Navigationsbuttons

- Klick auf „Einstellungen“ - Schriftzug -> Öffnen des Fensters „Einstellungen“
- Klick auf „virtuelle Gärten“ – Schriftzug -> Navigieren zum Fenster „Übersicht virtuelle Gärten“

Übersicht virtuelle Gärten

- Klick auf aktuell angezeigten Garten -> Auswahl des Gartens
- Klick auf die Pfeilsymbole -> Durchstöbern der angelegten Gärten
- Klick auf Plussymbol -> Anlegen eines neuen Gartens
- Klick auf „Dashboard“ – Schriftzug -> Navigieren zum Fenster „Hauptfenster“

Virtueller Garten

- Auswahl eines platzierten Objekts im virtuellen Garten -> Verschieben oder Entfernen des Objekts
- Auswahl eines Objekts aus dem Kauf-Bereich -> Kauf und Platzierung des Objekts im virtuellen Garten
- Klick auf „Dashboard“ – Schriftzug -> Navigieren zum Fenster „Hauptfenster“
- Klick auf „virtuelle Gärten“ – Schriftzug -> Navigieren zum Fenster „Übersicht virtuelle Gärten“

Eingabevalidierung

Die Eingabevalidierungen stellen sicher, dass alle Eingaben korrekt und sinnvoll sind.

Timer-Einstellungen

- Zeitwerte müssen positiv sein
- Zeitwerte dürfen eine festgelegte Grenze nicht überschreiten
    - Beim flexibel anpassbaren Timer maximal 24 Stunden bzw. 1440 Minuten
    - Beim Pomodoro-Timer darf die Arbeitszeit maximal 300 Minuten und die Pausenzeit maximal 120 Minuten betragen

Projekte

- Der Projektname darf nicht leer sein
- Der Projektname darf maximal 40 Zeichen lang sein
- Sonderzeichen sind erlaubt
- Startdatum muss ein Datum sein
    - Format überprüfen -> Eingabe von 1.2. bis zu 01.02.2020 ist erlaubt
    - Regulärer Ausdruck: ^(0?[1-9]|[12][0-9]|3[01])\.(0?[1-9]|1[0-2])\.(\d{4})?$

- Enddatum darf leer gelassen werden
- Enddatum muss ein Datum sein (falls nicht 0 bzw. leer)
- Enddatum muss im Vergleich zum Startdatum größer sein (in der Zukunft liegen)

Textfeld (auf dem Hauptfenster unter dem Timer-Bereich)

- Ausschließlich Text erlaubt
- Maximal 1000 Zeichen

Virtueller Garten

- Der Gartenname darf nicht leer sein
- Der Gartenname darf maximal 40 Zeichen lang sein
- Sicherstellen, dass die Platzierung eines Objekts nur in freien Rasterfeldern erfolgt, siehe Notiz #4-01/25
- Kein Kauf von Objekten, wenn der Punktestand nicht ausreicht

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.

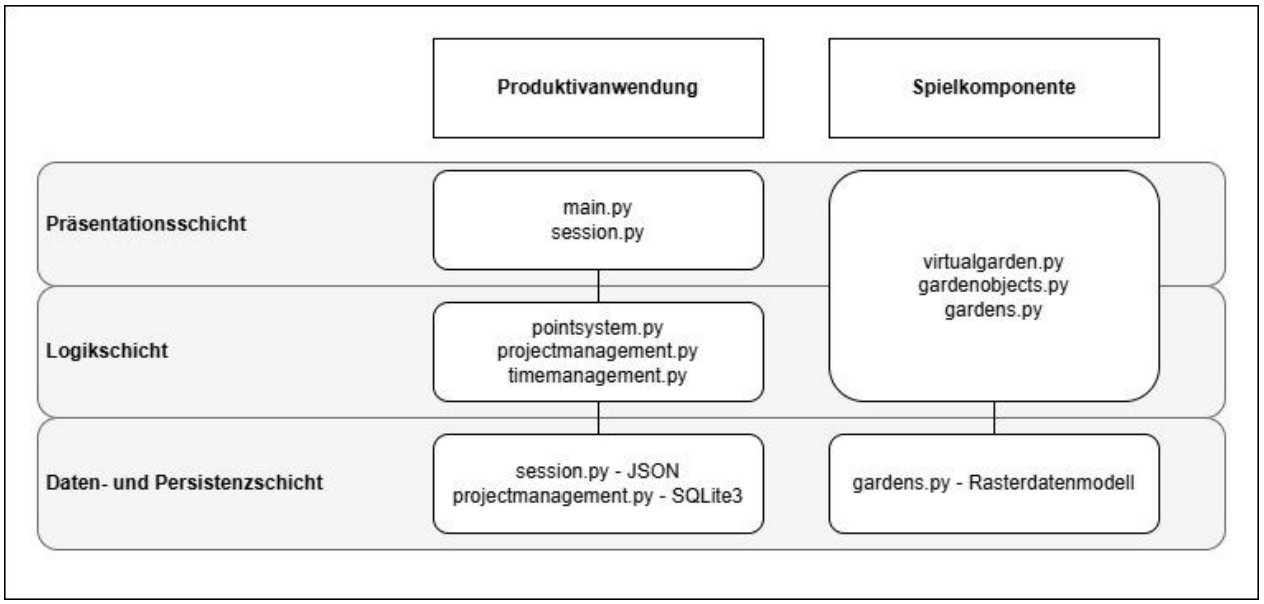

Abbildung 1 – Schichtenmodell der ProductivityGarden-Anwendung

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.

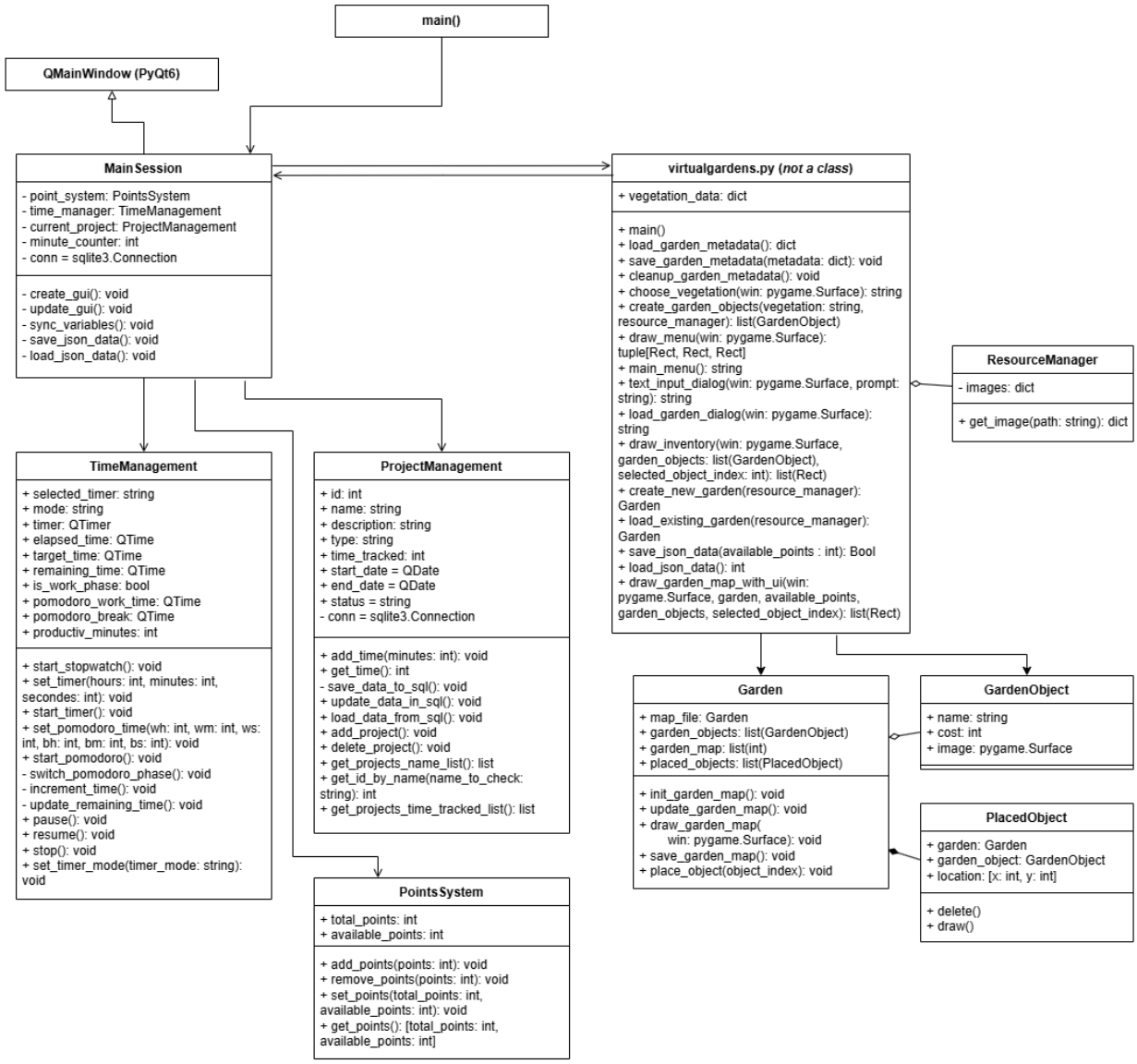

Abbildung 2 – UML-Klassendiagramm

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.

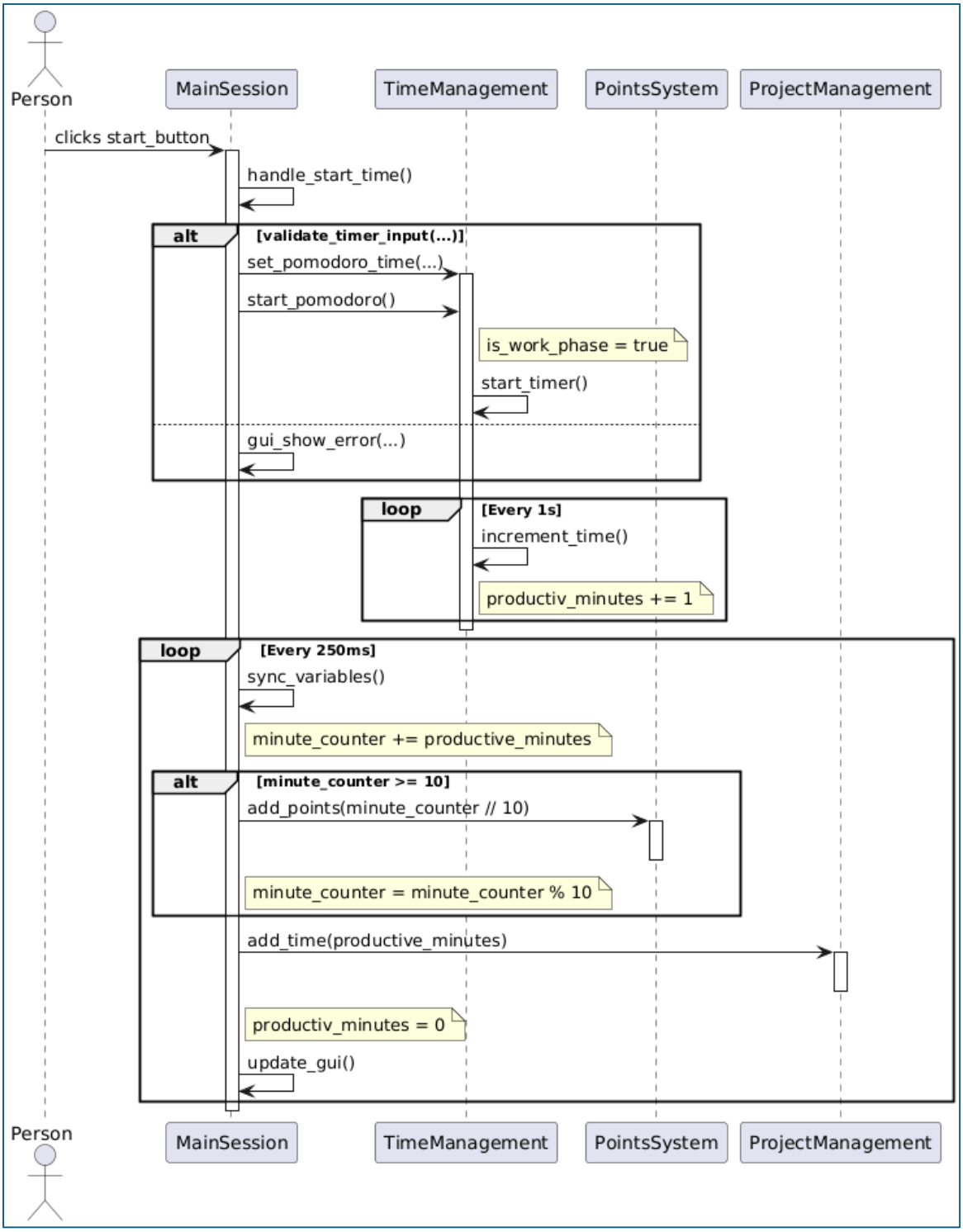

Abbildung 3 – UML-Sequenzdiagramm

Testdokument

Projektname: ProductivityGarden

Name	Jonas Huber
Matrikelnummer	IU14085128
Modul	Projekt: Software Engineering DLMCSPSE01_D
Datum	24.01.2025

Inhalt

Abbildungsverzeichnis

1. Teststrategie

Die Teststrategie beschreibt die Maßnahmen zur Qualitätssicherung der Anwendung ProductivityGarden. Ziel ist es, sicherzustellen, dass die Anwendung fehlerfrei, sicher, performant und benutzerfreundlich ist.

Whitebox-Tests

Bei Whitebox-Tests ist die interne Struktur und Funktionsweise des Codes bekannt und zugänglich, sodass gezielt einzelne Komponenten und deren Interaktionen auf Korrektheit und Effizienz geprüft werden können. Um die Qualität in sämtlichen Entwicklungsphasen gewährleisten zu können, erfolgt eine Unterteilung in drei Teststufen.

Unit-Tests

Unit-Tests konzentrieren sich auf die kleinsten Testeinheiten des Codes, meist Funktionen oder Methoden. Das übergeordnete Ziel besteht in der Sicherstellung der korrekten Funktionsweise einzelner Code-Einheiten, unabhängig von anderen Einheiten.

Diese Art von Tests können somit direkt nach der Fertigstellung einzelner Code-Einheiten durchgeführt werden, ohne dass weitere Abhängigkeiten berücksichtigt werden müssen.

Unit-Tests umfassen die Prüfung der Funktionalität, beispielsweise von Timer-Funktionalitäten wie des Pomodoro-Timers, des anpassbaren Timers und der Stoppuhr. Des Weiteren werden durch Unit-Tests die Korrektheit beim Speichern und Laden von Daten sowie bei Benutzerinteraktionen (Ereignisse wie Button-Klicks) überprüft.

Testmethoden:

- Manuelle Tests
- Automatisierte Unittests:
  Verwendung von Tools wie *unittest* für die Testautomatisierung

Integrationstests

Integrationstests überprüfen, ob die Interaktion zwischen verschiedenen Code-Einheiten reibungslos funktioniert.

Dabei wird beispielsweise überprüft, ob produktive Zeit korrekt in Punkte umgewandelt wird und einem Projekt zugeordnet wird.

Folglich müssen sämtliche Code-Einheiten, die mit den Tests in Zusammenhang stehen, vorab fertiggestellt werden.

Arten von Integrationstests:

- Big-Bang-Ansatz:
  Alle Module werden gleichzeitig integriert und getestet.
- Inkrementeller Ansatz:
  Module werden schrittweise integriert und getestet. (Top-Down-Ansatz oder Bottom-Up-Ansatz)

- Funktionale Integration:
  Testet Funktionen, die mehrere Module betreffen.

Testmethoden:

- Manuelle Tests
- Optional:
  - Testframeworks wie *pytest* mit Abdeckungsberichten (*pytest-cov*)
  - Integrationstools wie *tox*, um unterschiedliche Konfigurationen zu testen
  - Verwendung von Tools wie *pytest-qt*, *Selenium* oder *PyAutoGUI* für automatisierte GUI-Tests bzw. zur Automatisierung von Benutzerinteraktionen.

Systemtests
Im Rahmen von Systemtests erfolgt eine Prüfung der Anwendung als Ganzes unter realistischen Bedingungen. Dabei ist sicherzustellen, dass die Anwendung wie vorgesehen funktioniert und den Anforderungen entspricht.

Abdeckung:

- Benutzerschnittstelle
- Performance
  - Reaktionsgeschwindigkeit
  - Flüssigkeit der Animationen
  - Auslastung von Hardware-Ressourcen (wie CPU, GPU, RAM, ROM)
- Hintergrundbetrieb
- Kompatibilität

Testmethoden:

- Manuelle Tests:
  - Usability-Tests durch Benutzer, um die Benutzerfreundlichkeit und Performance zu bewerten.
  - Ressourcen-Auslastung mit Hilfe von Windows Task-Manager überwachen
- Optional:
  - Verwendung von Tools wie *pytest-qt*, *Selenium* oder *PyAutoGUI* für automatisierte GUI-Tests bzw. zur Automatisierung von Benutzerinteraktionen.

Weitere Tests
Zur weiteren Abdeckung relevanter Aspekte der Qualitätssicherung werden Statische Tests und bei Möglichkeit auch Blackbox-Tests herangezogen.

Statische Tests
Im Rahmen statischer Tests erfolgt keine Ausführung der zu untersuchenden Software. Im Rahmen eines sogenannten Code-Review wird die Software einer analytischen Überprüfung der einzelnen Code-Zeilen unterzogen, um deren Sinnhaftigkeit zu evaluieren. So kann beispielsweise eine Datenflussanalyse Aufschluss darüber geben, welche Daten herangezogen und weitergegeben werden und ob dies den Anforderungen entspricht.

Sinnvoll ist in diesem Zusammenhang der Einsatz eines Tools, welches den Quellcode mit Hilfe von konfigurierbaren Regeln automatisiert durchdringt und das Ergebnis in einer übersichtlichen Form ausgibt. Beispiele hierfür sind *Pylint*, *Flake8* & *Mypy*.

Blackbox-Tests
Im Gegensatz zu Whitebox-Tests ist dem Tester bei Blackbox-Tests der Code nicht bekannt. Der Fokus liegt hierbei auf der Überprüfung des sichtbaren Ergebnisses. Der Vorteil von Blackbox-Tests liegt in der Betrachtung der Software aus der Perspektive des Endnutzers.

Vorgehen zur Testdurchführung

Testplanung

- Definieren der Testfälle basierend auf den funktionalen und nicht-funktionalen Anforderungen in der Testspezifikation
    - Testfall mit eindeutiger ID
    - Testziel des Testfalls
    - Voraussetzungen die in der Anwendung vor der Ausführung des Tests hergestellt werden müssen
    - Eingabedaten und/oder auszuführende Aktionen der Benutzer:innen zur Durchführung des Tests
    - Erwartetes Ergebnis
- Priorisierung der Tests

Testdurchführung

- Whitebox-Tests
    - Unit-Tests während der Entwicklungsphase durchführen
    - Integrationstests ebenfalls während der Entwicklungsphase, aber jeweils nach Implementierung einzelner Module durchführen
        - Anfänglich wird mit dem inkrementellen Ansatz (Bottom-Up) versucht die Integrationstests abzudecken.
        - In speziellen Fällen können im Nachhinein auch noch weitere Tests mit dem Funktionalen Integrationstest-Ansatz notwendig sein.
    - Systemtests in der finalen Testphase vor der Veröffentlichung.
- Weitere Tests
    - Statische Tests während der Entwicklungsphase durchführen
    - Blackbox-Tests nach Möglichkeit in der finalen Testphase vor der Veröffentlichung (zusammen mit Systemtests)

Testprotokoll

- Erstellung des Testprotokolls im Anschluss an die Durchführung der Tests.
    - Umfasst die Testfälle aus der Testspezifikation sowie die tatsächlichen Ergebnisse

Testpriorisierung

Abbildung 1 zeigt die Testpriorisierung in drei Phasen aufgeteilt, basierend auf der Teststrategie und der Wichtigkeit der jeweiligen Module. Die Tests werden nach ihrer Kritikalität für die Anwendung priorisiert. Diese Priorisierung gewährleistet eine schrittweise Sicherstellung der Funktionalität, von den kleinsten Einheiten bis hin zum gesamten System. Zusätzlich wird sichergestellt, dass Fehler frühzeitig erkannt werden.

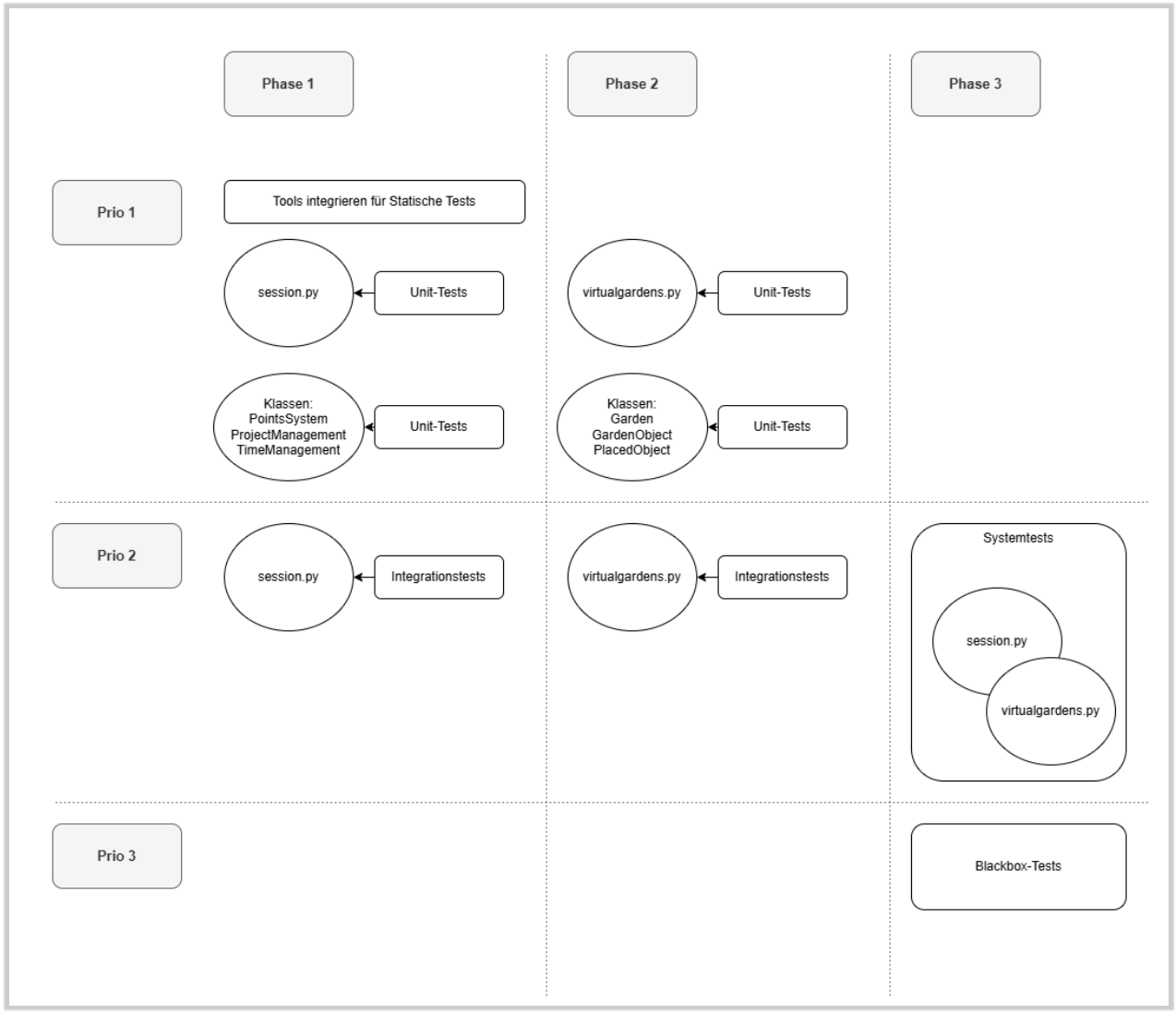

Abbildung 1 – Diagramm Testpriorisierung

2. Testprotokoll

Whitebox-Tests

Unit-Tests

Die Auflistung aller Unit-Tests würde den Rahmen dieses Dokuments sprengen und diese sind in den Dateien „..._test.py“ im Ordner „unittests“ ersichtlich. Aus diesem Grund werden die Tests der Komponenten „session.py“ hier nur zu Demonstrationszwecken aufgeführt:

Testfall 1: Initialisierung der Benutzeroberfläche

Testfall-ID: UTP01

Testziel:
Überprüfung der initialen Konfiguration der Benutzeroberfläche.

Voraussetzungen:
Die Anwendung wird gestartet.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
keine	1. Titel des Fensters: "ProductivityGarden" 2. Fensterbreite entspricht „WIDTH“ 3. Fensterhöhe entspricht „HEIGHT“	1. erfüllt 2. erfüllt 3. erfüllt

Testfall 2: Initialisierung der Punkteübersicht

Testfall-ID: UTP02

Testziel:

Validierung der initialen Punktanzahl in der Benutzeroberfläche.

Voraussetzungen:

- Ein Testdatensatz mit Punktinformationen ist in der Datenbank vorhanden.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
keine	Die Punkte in circle_av und circle_tot entsprechen den Werten der Punktverwaltung	

Testfall 3: Initialisierung des Timer-Modus

Testfall-ID: UTP03

Testziel: Überprüfung des initialen Zustands des Timer-Modus.

Voraussetzungen:

- Die Anwendung ist gestartet.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
keine	1. ausgewählter Timer-Modus ist „pomodoro“ 2. Zeitanzeige: „00:00:00“	

Testfall 4: Validierung der Timer-Eingabe

Testfall-ID: UTP04

Testziel: Überprüfung der Validierung von Benutzereingaben für den Timer.

Voraussetzungen:

- Die Anwendung ist gestartet.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. "01:00:00" 2. "25:00:00" 3. "00:00:59" 4. "invalid"	1. True 2. False 3. False 4. False	1. True 2. False 3. False 4. False

Testfall 5: Start des Timers

Testfall-ID: UTP05

Testziel: Überprüfung der Funktionalität des Timer-Starts.

Voraussetzungen:

- Die Eingaben für Arbeitszeit und Pausenzeit wurden gesetzt.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Setzen von pomodoro_work_input auf "00:25:00". 2. Setzen von pomodoro_break_input auf "00:05:00". 3. Ausführung von handle_start_time().	Timer-Modus wechselt zu: „running“	erfüllt

Testfall 6: Pausieren des Timers

Testfall-ID: UTP06

Testziel: Überprüfung der Funktionalität des Timer-Pausierens.

Voraussetzungen:

- Der Timer läuft.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Ausführung von handle_pause_time() -> 2x	1. Der Modus wechselt zu "paused". 2. Der Modus wechselt zurück zu "running".	1. erfüllt 2. erfüllt

Testfall 7: Stoppen des Timers

Testfall-ID: UTP07

Testziel: Überprüfung der Funktionalität des Timer-Stoppens.

Voraussetzungen:

- Der Timer läuft.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Ausführung von handle_stop_time()	Der Modus wechselt zu „stopped“	erfüllt

Testfall 8: Umschalten des Timer-Modus

Testfall-ID: UTP08

Testziel: Validierung der Funktionalität des Umschaltens zwischen verschiedenen Timer-Modi.

Voraussetzungen:

- Die Anwendung ist gestartet.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Ausführung von handle_toggle_mode(). -> 3x	Modus wechselt zu 1. „timer“ 2. „stopwatch“ 3. „pomodoro“	1. erfüllt 2. erfüllt 3. erfüllt

Testfall 9: Speichern und Laden von Daten

Testfall-ID: UTP09

Testziel: Überprüfung des Speicherns und Ladens von Benutzerdaten.

Voraussetzungen:

- Daten wurden in die Felder eingegeben.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Setzen von Punkten und Texteingaben. 2. Ausführung von save_json_data(). 3. Start einer neuen Sitzung und Ausführung von load_json_data().	Die gespeicherten Werte werden korrekt geladen.	erfüllt

Integrationstests

Dieser Abschnitt beinhaltet die verschiedenen Integrationstests, die manuell durchgeführt wurden. Die in der Teststrategie als optional deklarierten Testmethoden (siehe Integrationstests S.2) wurden aus zeitlichen Gründen nicht implementiert, jedoch würden diese sich für eine umfassendere Testabdeckung zukünftig lohnen. Die Testfälle sind wieder aufgeteilt in die zwei Hauptkomponenten Produktivanwendung und Spielkomponente

Allgemeine Voraussetzungen / Testumgebung:

- Das Python-Projekt ist lokal installiert/ausführbar.
- Standard-Einstellungen in der constants.py sind valide (z.B. Pfade, Farbcodes).
- Benötigte Python-Pakete (PyQt6, PyQt6.QtCharts etc.) sind installiert.

Für Produktivanwendung (session.py):

Testfall 1: Anwendung starten mit leerem/fehlendem JSON

Testfall-ID: ITP01

Testziel: Überprüfen, dass beim ersten Start oder bei fehlender JSON-Datei die Default-Werte korrekt initialisiert werden.

Voraussetzungen:

- Die JSON-Datei (JSON_FILE) existiert nicht oder ist leer.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Sicherstellen, dass die JSON-Datei entweder gelöscht oder geleert wurde. 2. Starte die Anwendung (MainSession).	1. Die Anwendung erzeugt eine neue JSON-Datei mit Default-Werten (z.B. total_points = 0, available_points = 0, voreingestellte Pomodoro-Zeiten etc.). 2. Keine Fehler oder Fehlermeldungen in der GUI. 3. Im GUI werden 0 Punkte (total/available) angezeigt und die voreingestellten Felder für Timer/Pomodoro sind sichtbar (z.B. "00:25:00" / "00:05:00").	1. erfüllt 2. erfüllt 3. erfüllt

Testfall 2: Start/Pause/Stop der Stoppuhr (Stopwatch)

Testfall-ID: ITP02

Testziel: Verifizieren, dass der Stopwatch-Modus korrekt funktioniert und die GUI-Elemente (Zeitlabel, Buttons) richtig reagieren.

Voraussetzungen:

- Anwendung ist gestartet.
- Der Timer-Modus steht auf „Stopwatch“ (ggf. umschalten über den „Switch to ...“-Button).

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Klicke auf den Start-Button. 2. Beobachte das Zeitlabel (z.B. clock_label) für einige Sekunden. 3. Klicke auf Pause. 4. Warte einige Sekunden und überprüfe, ob die Zeit weiterhin eingefroren bleibt. 5. Klicke auf Resume. 6. Beobachte das Zeitlabel wieder einige Sekunden. 7. Klicke auf Stop.	1. Nach Schritt 1 läuft die Stoppuhr los (z.B. 00:00:01, 00:00:02, ...). 2. Nach Schritt 3 wechselt der Button-Text auf „Resume“ und die Uhrzeit friert ein. 3. Nach Schritt 5 zählt die Stoppuhr an der eingefrorenen Stelle weiter (keine Nullstellung). 4. Nach Schritt 7 wird die Zeit auf 00:00:00 zurückgesetzt, und der Button-Text ändert sich zu „Pause“.	1. erfüllt 2. erfüllt 3. erfüllt 4. erfüllt

Testfall 3: Umschalten zwischen Pomodoro, Timer und Stopwatch

Testfall-ID: ITP03

Testziel: Validieren, dass das Umschalten des Timer-Modus per Button die Eingabefelder und Labels korrekt anpasst.

Voraussetzungen:

- Anwendung ist gestartet und befindet sich in irgendeinem Modus.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Stelle sicher, dass der aktuelle Modus z.B. „Pomodoro“ ist. 2. Klicke auf den Button „Switch to Timer“. 3. Beobachte die GUI: Pomodoro-Felder (Work/Break) sollten verschwinden, das Timer-Eingabefeld erscheint. 4. Klicke erneut auf den Button, bis „Stopwatch“ erscheint. 5. Beobachte, dass nun weder Pomodoro- noch Timer-Eingabefelder zu sehen sind.	1. Bei „Pomodoro“ sind zwei Eingabefelder sichtbar (pomodoro_work_input, pomodoro_break_input) und korrekt beschriftet. 2. Bei „Timer“ wird nur das Timer-Eingabefeld (timer_input_field) angezeigt, während Pomodoro-Felder ausgeblendet sind. 3. Bei „Stopwatch“ sind alle Eingabefelder ausgeblendet. 4. Die Label timer_mode_label zeigt die richtige Beschriftung (z.B. „POMODORO“, „TIMER“, „STOPWATCH“).	1. erfüllt 2. erfüllt 3. erfüllt 4. erfüllt

Testfall 4: Pomodoro-Modus starten mit validen und invaliden Zeiten

Testfall-ID: ITP04

Testziel: Überprüfen, dass Pomodoro-Work/Break-Einstellungen korrekt validiert werden und der Ablauf stimmt.

Voraussetzungen:

- Anwendung ist gestartet, Modus ist auf „Pomodoro“ eingestellt.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Setze pomodoro_work_input auf "00:25:00". 2. Setze pomodoro_break_input auf "00:05:00". 3. Klicke auf Start. 4. Prüfe, ob das Label die verbleibende Zeit anzeigt („Work: 00:25:00“, läuft herunter). -- Erneut mit invalide Eingabe -- 1. Trage in pomodoro_work_input einen ungültigen Wert ein, z.B. "25:00" (falsches Format ohne Stunden/Minuten-Trennung) oder "00:00:30" (unter 1 Minute). 2. Klicke auf Start.	1. Keine Fehlermeldung erscheint. 2. Die Pomodoro-Uhr beginnt im Work-Abschnitt zu laufen. 3. Nach Ablauf der Work-Zeit (falls gewartet wird) schaltet es automatisch in den Break-Modus um. -- invalide Eingabe: -- 1. Eine Fehlermeldung in roter Farbe (input_error_label) weist auf ein falsches Format oder auf die Mindestzeit hin. 2. Die Pomodoro-Uhr startet nicht.	1. erfüllt 2. erfüllt 3. erfüllt -- invalide Eingabe: -- 1. erfüllt 2. erfüllt

Testfall 5: Timer-Modus starten, ablaufen lassen und prüfen^

Testfall-ID: ITP05

Testziel: Verifizieren, dass der Timer-Modus die Zeit korrekt herunterzählt und bei Ablauf stoppt.

Voraussetzungen:

- Anwendung ist gestartet, Modus ist ggf. auf „Timer“ umgeschaltet.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Setze das Feld timer_input_field auf "00:00:05" (5 Sekunden zum Testen). 2. Klicke auf Start. 3. Beobachte das clock_label. 4. Warte, bis die 5 Sekunden abgelaufen sind.	1. Die Uhr zählt rückwärts von 00:00:05 auf 00:00:00. 2. Bei 00:00:00 stoppt die Uhr. 3. Modus wechselt in den Zustand „stopped“ (Button „Stop“ hat keinen Effekt mehr bzw. ist wieder auf Default).	1. erfüllt 2. erfüllt 3. erfüllt

Testfall 6: Anlegen eines neuen Projekts und Aktualisierung der Übersicht

Testfall-ID: ITP06

Testziel: Sicherstellen, dass beim Hinzufügen eines Projekts (handle_add_new_project) die Datenbankeinträge und GUI-Elemente (Dropdown, Infofelder) aktualisiert werden.

Voraussetzungen:

- Anwendung ist gestartet.
- Es existiert mindestens 1 Projekt in der Datenbank.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Klicke im Project-Bereich auf Add. 2. Beobachte das Dropdown projects_dropdown. 3. Ein neues Projekt mit Standardwerten (z.B. "New Project" o. Ä.) sollte ausgewählt sein. 4. Ändere den Namen im Feld pr_name_input auf z.B. "Testprojekt" und beobachte, ob sich der Eintrag im Dropdown direkt ändert. 5. Klicke z.B. in ein anderes Feld oder warte, bis der Datensatz gespeichert wird.	1. Dropdown enthält einen neuen Eintrag. 2. Die Anzeige in pr_name_input und der Dropdown-Eintrag stimmen überein. 3. Kein Fehler im Log / keine Exception. 4. Bei Aufruf von ProjectManagement.get_projects_name_list(...) taucht das neue Projekt in der Rückgabe auf.	1. erfüllt 2. erfüllt 3. erfüllt 4. erfüllt

Testfall 7: Löschen eines Projekts

Testfall-ID: ITP07

Testziel: Prüfen, dass das Löschen des aktuell gewählten Projekts korrekt die Datenbank und GUI anpasst.

Voraussetzungen:

- Es sind mindestens 2 Projekte in der Datenbank (damit das Löschen bemerkbar ist).
- Ein Projekt ist im Dropdown angewählt.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Stelle sicher, dass im Dropdown das zu löschende Projekt ausgewählt ist. 2. Klicke auf Delete. 3. Beobachte das Dropdown.	1. Das gelöschte Projekt verschwindet aus dem Dropdown. 2. Die Anwendung schaltet (automatisch) auf das nächste verfügbare Projekt um (z.B. Index 0). 3. Kein Fehler/Absturz. 4. ProjectManagement.get_id_by_name(...) sollte für das gelöschte Projekt nun None oder Fehler liefern, weil es nicht mehr existiert.	1. erfüllt 2. erfüllt 3. erfüllt 4. erfüllt

Testfall 8: Manuelles Hinzufügen von Zeit zu einem Projekt

Testfall-ID: ITP08

Testziel: Überprüfen, dass die Eingabe im Feld pr_add_time korrekt validiert wird und die Projektzeit entsprechend hochgezählt wird.

Voraussetzungen:

- Ein bestehendes Projekt ist ausgewählt.
- Zeitmanagement kann gestoppt sein (zur Vereinfachung).

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Trage im Feld pr_add_time einen gültigen Integer ein, z.B. "15". 2. Klicke auf den Button Add time. 3. Beobachte die Anzeige (Kreis circle_project_time). 4. Wiederhole den Vorgang mit einem ungültigen Wert, z.B. "abc" oder "1000".	1. Bei "15" wird die Projektzeit (in Minuten) um 15 erhöht. circle_project_time zeigt entsprechend einen um 15 erhöhten Wert. 2. Bei Eingaben außerhalb des erlaubten Bereichs (1–999) oder nicht numerischen Werten erscheint eine Fehlermeldung im GUI (gui_show_error). 3. Keine Exceptions im Log	1. erfüllt 2. erfüllt 3. erfüllt

Testfall 9: Punkte-Update alle 10 produktiven Minuten

Testfall-ID: ITp09

Testziel: Sicherstellen, dass alle 10 gesammelten Produktiv-Minuten (z.B. im Stopwatch-Modus) automatisch Punkte gutgeschrieben werden.

Voraussetzungen:

- Es existiert mindestens 1 Projekt. (Wird automatisch angelegt beim erstmaligen Starten der Anwendung)
- Punktesystem steht zu Beginn auf einem bekannten Wert (z.B. total=0, available=0).

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Starte die Stoppuhr (oder füge manuell Zeit hinzu, sodass self.time_manager.productiv_minutes ansteigt). 2. Erzeuge mindestens 10 produktive Minuten. 3. Achte darauf, dass der interne Zähler (in minute_counter) alle 10 Minuten 1 Punkt vergibt.	1. Nach (akkumulierten) 10 produktiven Minuten erhöht sich circle_av und circle_tot beide um 1. 2. GUI zeigt diesen neuen Punktestand an. 3. Bei z.B. 20 Minuten steigt der Wert erneut um 1 Punkt.	1. erfüllt 2. erfüllt 3. erfüllt

Testfall 10: Aktualisierung der Projekt-Pie-Chart

Testfall-ID: ITP10

Testziel: Überprüfen, dass das Pie-Chart (ProjectsOverviewPieChart) nach Änderungen an den Projektdaten (Zeit) korrekt aktualisiert wird.

Voraussetzungen:

- Mindestens 2 Projekte existieren.
- Die Projekt-Zeitdaten im Dropdown sind verschieden (z.B. Projekt A = 10 min, Projekt B = 30 min).

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Wähle „Projekt A“ aus und füge manuell 10 min hinzu. 2. Lasse einige Sekunden verstreichen, damit der update_low_frequency()-Timer auslöst. 3. Beobachte das Pie-Chart. 4. Schalte auf „Projekt B“ und füge dort ebenfalls Zeit hinzu. 5. Beobachte erneut das Pie-Chart nach einigen Sekunden.	1. Die Tortenstücke in der Chart spiegeln sofort oder nach dem 2-Sekunden-Intervall die neuen Zeitwerte wider. 2. Kein Darstellungsfehler oder GUI-Absturz.	1. erfüllt 2. erfüllt

Testfall 11: Ungültige Eingabe im Projekt-Namen (Sonderzeichen, Leerzeichen)

Testfall-ID: ITP11

Testziel: Sicherstellen, dass die Eingabefelder für den Projektnamen und andere Felder nur erwartete Zeichen zulassen und ggf. Validierungen/Begrenzungen greifen.

Voraussetzungen:

- Anwendung ist gestartet.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Klicke auf Add, um ein neues Projekt hinzuzufügen. 2. Gib im Feld pr_name_input eine Zeichenkette mit Sonderzeichen ein, z. B. "@@@###!!!" oder eine sehr lange Zeichenkette mit über 40 Zeichen. 3. Beobachte, ob und wie das GUI reagiert (z. B. rote Rahmen, Fehlermeldung, automatisches Kürzen).	1. Das Feld darf maximal 40 Zeichen übernehmen und sollte automatisch kürzen oder eine Fehlermeldung ausgeben. 2. Sonderzeichen könnten erlaubt sein oder vom System abgewiesen werden – je nach Designvorgabe. 3. Keine Abstürze oder Exceptions.	1. erfüllt 2. erfüllt 3. erfüllt

Testfall 12: Datenpersistenz beim Neustart (JSON)

Testfall-ID: ITP12

Testziel: Verifizieren, dass User-Eingaben (Punktestand, Timer-Einstellungen, Text aus text_box) korrekt in der JSON-Datei gespeichert und beim Neustart wiederhergestellt werden.

Voraussetzungen:

- Anwendung ist gestartet.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Setze in der Anwendung einige Werte: total_points = 5, available_points = 3 (durch Zeitmessung oder manuelles Hinzufügen). 2. Pomodoro-Felder auf z. B. 00:20:00 und 00:10:00. 3. Tippe einen Text in text_box. 4. Schließe die Anwendung sauber (Fenster schließen). 5. Starte die Anwendung erneut.	1. Die zuvor gesetzten Werte werden aus der JSON-Datei geladen und in der GUI angezeigt. 2. Punktestand, Timer-Eingaben und Text bleiben erhalten. 3. Keine Fehlermeldung aufgrund ungültiger oder fehlender JSON-Daten.	1. erfüllt 2. erfüllt 3. erfüllt

Testfall 13: Projektwechsel während laufender Zeitmessung

Testfall-ID: ITP13

Testziel: Überprüfen, dass ein Wechsel des ausgewählten Projekts keine Probleme verursacht, wenn die Stoppuhr oder Timer noch läuft.

Voraussetzungen:

- Zwei verschiedene Projekte sind angelegt.
- Zeitmessung (z. B. Stopwatch) ist gestartet.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Während die Stoppuhr läuft, wähle über das Dropdown ein anderes Projekt aus. 2. Beobachte, ob die gestoppte Zeit später korrekt auf das nun selektierte Projekt gebucht wird. 3. Stoppe die Zeitmessung. 4. Prüfe den Zeitwert in beiden Projekten (via GUI-Kreise oder manuelle SELECT-Abfrage in der DB), ob eine doppelte oder falsche Zeitbuchung stattfand.	1. Die bis zum Projektwechsel gesammelten Minuten werden dem ursprünglichen Projekt zugerechnet. 2. Die gesammelten Minuten nach dem Projektwechsel, werden dem neu selektierten Projekt zugerechnet. 3. Kein Absturz, keine doppelte oder falsche Zeitbuchung.	1. erfüllt 2. erfüllt 3. erfüllt

Für Spielkomponente (virtualgardens.py):

Testfall 14: Start des Spiels & Metadaten-Cleanup

Testfall-ID: ITVG01

Testziel: Prüfen, ob das Spiel in das Hauptmenü gelangt, sowie sicherstellen, dass vorhandene .map-Dateien in gardens_data.json eingetragen sind und nicht mehr existierende Dateien entfernt werden.

Voraussetzungen:

- gardens_data.json existiert (ggf. mit Einträgen, die nicht mehr zu .map-Dateien passen).
- Mindestens eine .map-Datei im gardens Ordner die nicht in der gardens_data.json aufgeführt ist
- Mindestens eine .map-Datei die in der gardens_data.json existiert löschen

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Starte das Python-Skript (virtualgardens.py). 2. Beobachte die Konsolenausgabe zur „Cleanup“-Funktion. 3. Warte, bis das Hauptmenü erscheint.	1. Nicht mehr existierende .map-Dateien werden aus gardens_data.json entfernt (Konsolenausgabe: "[Cleanup] Removed metadata entry ..."). 2. Neue .map-Dateien, die noch nicht im JSON vorhanden sind, werden hinzugefügt (Konsolenausgabe: "[Cleanup] Added metadata entry ...").	1. erfüllt 2. erfüllt

Testfall 15: Neues Gartenprojekt anlegen (Create Garden)

Testfall-ID: ITVG02

Testziel: Integrationstest für das Anlegen eines neuen Gartens über das Hauptmenü:

- Menüauswahl („Create Garden“).
- Vegetationsauswahl über choose_vegetation().
- Namenseingabe über text_input_dialog().
- Anlage der entsprechenden GardenObjects über create_garden_objects() und Speichern in .map-Datei.

Voraussetzungen:

- Spiel befindet sich im Hauptmenü
- gardens_data.json ist vorhanden/synchronisiert.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Klicke im Hauptmenü auf „Create Garden“. 2. Wähle im darauf folgenden Vegetationsdialog „City Park“ (oder eine andere Option) per Mausklick. 3. Im Namenseingabedialog: Gib z. B. „TestGarden1“ ein und bestätige mit ENTER. 4. Beobachte nach Bestätigung, ob ein neues Spielinterface geladen wird. 5. Prüfe im gardens Ordner, ob eine neue .map Datei erzeugt wurde. 6. Prüfe in der gardens_data.json, ob ein neuer Eintrag erzeugt wurde.	1. Das Spiel fragt erfolgreich die Vegetation ab (Fenster mit „Choose vegetation: …“). 2. Nach Eingabe des Namens wechselt das Spiel in den „Garten-Editor“-Bildschirm. 3. Im gardens Ordner liegt nun eine neue .map Datei. 4. In gardens_data.json existiert ein entsprechender Eintrag.	1. erfüllt 2. erfüllt 3. erfüllt 4. erfüllt

Testfall 16: Garten laden (Load Garden)

Testfall-ID: ITVG03

Testziel: Prüfung der Funktionalität zum Laden eines existierenden Gartens:

- Auswahl im Hauptmenü („Load Garden“).
- Dateiauswahl-Dialog via load_garden_dialog().
- Korrekte Wiederherstellung der Vegetation und GardenObjects über create_garden_objects().

Voraussetzungen:

- Spiel befindet sich im Hauptmenü.
- Mindestens eine .map-Datei liegt in MAP_FOLDER_PATH vor (z. B. „MyDesert.map“).
- gardens_data.json enthält einen entsprechenden Eintrag

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Klicke im Hauptmenü auf „Load Garden“. 2. Wähle in der angezeigten Liste eine .map aus. 3. Beobachte, ob der Gartenbildschirm geöffnet wird. 4. Stelle sicher, dass die geladenen Objekte (z. B. Kakteen, Büsche etc.) zum Vegetationstyp passen (z.B. „Desert“).	1. Nach der Auswahl von „MyDesert.map“ zeigt das Spiel den entsprechenden Garten an. 2. Die Vegetation stimmt mit „Desert“ überein (Sandhintergrund, Objekte der Wüste im Inventar). 3. Keine Fehlermeldungen in der Konsole.	1. erfüllt 2. erfüllt 3. erfüllt

Testfall 17: Platzieren eines Objekts mit ausreichenden und nicht ausreichenden Punkten

Testfall-ID: ITVG04

Testziel: Verifizieren, dass das Platzieren eines Objekts funktioniert, wenn genügend „available_points“ vorhanden sind. Integration zwischen Garden-Objekt, Punktelogik aus JSON-Datei und Inventar-Darstellung.

Voraussetzungen:

- Ein geladener oder neu erstellter Garten ist geöffnet.
- available_points ist im JSON_FILE auf einen Wert ≥ 10 gesetzt (zum Testen ausreichend hoch).
- Der Nutzer ist im Garten-Editor, ein Objekt (z. B. tree mit Kosten=8) steht im Inventar zur Verfügung.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Stelle sicher, dass available_points ≥ 2 2. Klicke in der Inventarleiste auf ein Objekt das 2 Punkte kostet. 3. Klicke anschließend irgendwo auf die freie Gartenfläche. 4. Beobachte, ob das Objekt platziert wird. 5. Prüfe die neue Punktzahl (z. B. „Points: 8“, wenn man 10 hatte und 2 ausgibt). 6. Klicke in der Inventarleiste auf ein Objekt das mehr Punkte kostet, als im Punktekonto verfügbar sind. 7. Klicke anschließend irgendwo auf die freie Gartenfläche. 8. Beobachte, ob das Objekt platziert wird, ob eine Fehlermeldung erscheint und prüfe die Punktzahl	-- ausreichende Punkte -- 1. Das ausgewählte Objekt erscheint auf der angeklickten Position des Gartens. 2. Punktestand sinkt korrekt um die Objektkosten (z. B. von 10 auf 8). 3. Keine Fehlermeldung 4. Die .map-Datei wird gespeichert (logisch oder im Dateisystem überprüfbar). -- nicht ausreichenden Punkte -- 5. Das ausgewählte Objekt erscheint nicht ! auf der angeklickten Position des Gartens. 6. Punktestand bleibt bestehen. 7. Fehlermeldung „Not enough points“ wird in der Konsole angezeigt	1. erfüllt 2. erfüllt 3. erfüllt 4. erfüllt 5. erfüllt 6. erfüllt 7. erfüllt

Systemtests

Testfall 1: Gesamte Funktionsabfolge als Endanwender

Testfall-ID: ST01

Testziel: Prüfen, ob die Kernabläufe (Projekte anlegen, Zeit tracken, Punkte erhalten, Daten speichern/laden, Anwendungen wechseln, Garten laden und Objekte platzieren) in einer durchgängigen Endnutzer-Situation fehlerfrei funktionieren.

Voraussetzungen:

- Anwendung ist installiert/ausführbar.
- Eine leere oder Standard-Datenbank ist vorhanden.
- Vorhandensein von Standard-Dateien (Images, JSON).

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Start der Anwendung • Stelle sicher, dass das Hauptfenster erscheint. 2. Projekt anlegen • Lege ein neues Projekt an (z. B. „Mein Systemtest-Projekt“). • Fülle Name, Beschreibung, Kategorie und Start/Enddatum. 3. Stopwatch starten • Stelle den Modus auf „Stopwatch“. • Starte, pausiere, setze fort, und stoppe schließlich die Uhr. 4. Punktesystem prüfen • Beobachte, ob nach einigen gesammelten Minuten die Punkte hochgezählt werden. 5. Kompletten Pomodoro-Zyklus durchlaufen • Schalte den Modus auf Pomodoro. • Eingabe gültiger Zeiten (z. B. Work 00:25:00, Break 00:05:00). • Starte, warte den Work-Abschnitt ab, prüfe automatischen Wechsel zur Break-Phase. 6. Projekt-Zeit prüfen • Wechsle zum Projekt, sieh dir den Zeitfortschritt im Kreis (z. B. 15 min) an.	1. Anwendung bleibt stabil, ohne Fehlermeldungen / Abstürze. 2. Alle Eingaben werden korrekt übernommen und angezeigt. 3. Punkte, Projektzeit und Projektinfos, sowie Gartendaten sind konsistent. 4. Keine spürbaren Verzögerungen bei normalen Interaktionen.	1. erfüllt 2. erfüllt 3. erfüllt 4. erfüllt

• Füge manuell weitere Minuten hinzu und verifiziere Diagramm-Update. 7. Daten persistieren • Schließe die Anwendung. • Starte sie erneut. Prüfe, ob die Projekte, Punktestände, Timer-Einstellungen und Texte in der Oberfläche geladen sind. 8. Über Button „TO THE GARDENS" in die Spielkomponente wechseln 9. Neuen Garten anlegen • Prüfen ob Punktestand korrekt • Objekte platzieren 12. mit Escape-Taste ins Menü navigieren 13. Zurück zur Produktivanwendung wechseln und wieder zu den Gärten zurückwechseln. 14. Erstellten Garten laden • Prüfen ob Objekte platziert • Prüfen ob Punktestand korrekt		

Testfall 2: Performance- und Ressourcen-Test (manuell)

Testfall-ID: ST02

Testziel: Untersuchen der Reaktionsgeschwindigkeit und Hardwareauslastung (CPU, RAM) unter typischer und erhöhter Last.

Voraussetzungen:

- Anwendung ist installiert/ausführbar.
- Windows Task-Manager (oder ein ähnliches Monitoring-Tool) ist offen, um CPU-/RAM-Auslastung zu beobachten.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Normalbetrieb • Starte die Anwendung; beobachte RAM-Verbrauch im Task-Manager. • Führe typische Aktionen aus (Stopwatch, Timer starten/stoppen, Projekt anlegen/löschen) und notiere grob CPU-/RAM-Peaks. 2. Stressphase • Wechsle schnell zwischen Pomodoro, Timer und Stoppuhr. • Erstelle mehrere Projekte in schneller Abfolge. • Öffne und schließe ggf. das Fenster (sofern Fenster-/Minimieren-Funktionen existieren). 3. Beobachtung • Achte auf Ruckler oder Verzögerungen in der GUI (z. B. Eingabeverzögerungen). • Prüfe, ob CPU-Last dauerhaft unverhältnismäßig hoch wird. 4. Über Button „TO THE GARDENS" in die Spielkomponente wechseln 5. Garten laden 6. Objekte platzieren 7. Schnell mehrmals Gärten schließen und wieder laden	1. Das Programm bleibt reaktionsschnell, kein Einfrieren oder starker Ressourcenanstieg. -- für beide Anwendungen jeweils: -- 2. CPU-Auslastung steigt eventuell kurzzeitig bei Diagramm-Updates, normalisiert sich aber zügig. 3. RAM-Verbrauch bleibt im erwarteten Rahmen (< einige hundert MB, je nach Projekt).	1. erfüllt -- Produktivanwendung -- 2. keine merkbare Mehrbelastung der CPU 3. etwa 50-80 MB RAM-Verbrauch -- Spielkomponente – 2. beim Laden eines Gartens leicht spürbare Mehrbelastung der CPU 3. etwa 80-100MB RAM-Verbrauch

Testfall 3: Hintergrundbetrieb und Minimieren

Testfall-ID: ST03

Testziel: Validieren, dass die Funktionen der Produktivanwendung wie der Timer oder die Stoppuhr weiterlaufen, wenn das Fenster minimiert wird oder in den Hintergrund wechselt.

Voraussetzungen:

- Produktivanwendung läuft normal im Vordergrund

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Start die Zeitmessung in einem beliebigen Modus. 2. Starte parallel eine Stoppuhr auf dem Smartphone oder Ähnlichem. 2. Minimiere die Anwendung oder wechsle zu einer anderen Anwendung. 3. Warte ca. 11–12 Minuten. 4. Maximiere die Anwendung wieder. 5. Über Button „TO THE GARDENS" in die Spielkomponente wechseln 6. Einen beliebigen Garten laden oder erstellen 7. Neues Objekte platzieren 8. Minimiere die Anwendung 9. Warte 1-2 Minuten 10. Maximiere die Anwendung	1. Die im Hintergrund weitergelaufene Zeit erscheint im Zeit-Label (z. B. 00:01:30). 2. Keine Fehlermeldung oder Stopp der Zeitmessung durch Minimieren. 3. Punkte-Berechnung (alle 10 Minuten) funktioniert weiterhin, wenn es im Hintergrund erreicht wird. 4. Garten bleibt geöffnet und platziertes Objekt bleibt bestehen	1. erfüllt 2. erfüllt 3. erfüllt 4. erfüllt

Statische Tests

Während der Entwicklung der verschiedenen Python Skripte und Komponenten wurden die Code-Analyse-Tools Pylance und Flake8 verwendet. Pylance wurde hauptsächlich eingesetzt als Echtzeitunterstützung und Flake8 zur Überprüfung der Codequalität und Einhaltung von PEP 8.

Blackbox-Tests

Aus Zeitgründen wurden keine Blackbox-Tests durchgeführt. Dies ist eine wichtige Methode, um Fehler in der Funktionalität und im Verhalten der Software zu identifizieren und einen besseren Einblick in die Gestaltung der Benutzeroberfläche und der Interaktionen zu bekommen. Daher wäre dies eine gute Möglichkeit, um weitere Verbesserungsmöglichkeiten der Anwendung zu identifizieren. Als Hilfe kann folgender Testfall dienen:

Testfall: Usability-Test durch Benutzer

Testziel: Überprüfen, wie „externe“ Testpersonen den Workflow wahrnehmen.

Voraussetzungen:

- Ein Test-User (nicht Entwickler) ist verfügbar.
- Kurze Einführung in die Hauptfunktionen (Projekte, Timer, Punkte).

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Der Test-User bedient die Anwendung auf eigene Faust. 2. Der Test-User dokumentiert: Verständnisfragen („Wo erstelle ich ein neues Projekt?“). Subjektive Wahrnehmungen wie bspw. Reaktionszeiten und Optik der Oberfläche. 3. Gefundene Bugs oder unklare Fehlermeldungen.	1. Nutzer findet sich intuitiv zurecht, kann bspw. Zeit erfassen, Projekte anlegen und zu den virtuellen Gärten wechseln. 2. Es gibt keine großen Hindernisse (z. B. unklare Eingabefelder oder Buttons). 3. Feedback zur GUI-Leistung und -Optik ist überwiegend positiv.	…

Abstract

Projektname: ProductivityGarden

Name	Jonas Huber
Matrikelnummer	IU14085128
Modul	Projekt: Software Engineering DLMCSPSE01_D
Datum	25.01.2025

1. Making-of

Projektidee
Die Grundidee von ProductivityGarden basiert auf der bekannten Pomodoro-Technik, erweitert um ein spielerisches Belohnungssystem. Die Idee ist entstanden, weil ich selbst ein großer Fan von Produktivitätssteigerungs-Anwendungen und Projekt- bzw. Zeittracking bin. Aus diesem Grund habe ich in der Vergangenheit bereits mehrfach ähnliche Anwendungen genutzt.

Planung und Entwurf
Zu Beginn wurden User-Stories und erste UML-Diagramme (z.B. Klassendiagramm) entwickelt. Aufgrund mangelnder Erfahrung gestaltete sich dieser Entwurfsprozess jedoch herausfordernd. Zudem erwies sich die Definition einer klaren Architektur als anspruchsvoll, was später zu mehrfachen Anpassungen im Code führte.

Aufteilung der Implementierung
Die Implementierungsphase wurde in drei weitere Phasen aufgeteilt. Zunächst wurde an der Implementierung des Dashboards, der Zeiterfassung & des Punktekontos gearbeitet. Diese haben sich als die wichtigsten Komponenten herausgestellt und sollten daher als erstes implementiert und getestet werden. Danach wurden die Features rund um die Projekte implementiert und getestet. Nach Fertigstellung der produktiven Anwendung, konnte mit der Implementierung der Spielkomponente begonnen werden.

Spielkomponente und Assets
Für die Gestaltung der Garten-Objekte wurde zunächst mit *Pillow* experimentiert. Schließlich wurden die Bilder mit *DALL·E* generiert und der Hintergrund mit *GIMP* entfernt, um die Pflanzen und Dekorationen optisch besser in die Gärten zu integrieren.

2. Kritische Reflexion

Was lief gut?

Motivierende Projektidee:
Da ich selbst zur Zielgruppe der Anwendung gehöre, war das Projekt insgesamt sehr praxisnah und interessant.

Implementierung aller grundlegenden Funktionalitäten:
Die in der Konzeptionsphase (hauptsächlich im Anforderungs- und Spezifikationsdokument) definierten Features und Geschäftsregeln konnten fast ausnahmslos erfolgreich umgesetzt werden.

Technische Umsetzung:
Die Verwendung von PyQt6 und Pygame bot eine solide Basis, um sowohl GUI als auch spielerische Elemente ohne Vorkenntnisse zu realisieren.

Herausforderungen und Verbesserungsmöglichkeiten

Entwurf und Architektur
Durch fehlende Vorerfahrungen gestaltete sich die Konzeptionsphase schwieriger als erwartet. In der Folge wurde bspw. der Code sowohl in der session.py als auch in der virtualgarden.py stark gebündelt. Dadurch besteht keine klare Trennung zwischen den verschiedenen Schichten in der

Schichtenarchitektur. Künftig sollte eine bessere Trennung der Schichten erfolgen. Beispielsweise könnte man sich stärker am MVC Design Pattern orientieren.

Zeitplan und Umsetzung
Die Projektphasen waren zu wenig klar auf Basis- und Erweiterungsfunktionen abgestimmt. Es empfiehlt sich, Implementierungsphasen deutlicher in die Kategorien „Essentials“ und „Optimizations/Optionals“ zu trennen. Dies hätte es ermöglicht, die Kernfunktionen stabil und vollständig fertig zu stellen, bevor optionale Funktionen integriert oder Optimierungen z.B. am GUI-Design vorgenommen werden.

Da die Architekturphase zu kurz kam, mussten spätere Anpassungen vorgenommen werden. Das führte zu verringertem Fokus auf Tests. Eine engmaschigere Projektüberwachung und ein flexibleres Task-Management (z.B. Scrum-Sprints) hätten hier helfen können.

Testing
Auch wenn das Projekt ein funktionsfähiges Produkt liefert, wurde das Testkonzept aus Zeitgründen vernachlässigt. Besonders Blackbox-Tests und automatisierte GUI-Tests (z.B. mit *pytest-qt*, *Selenium* oder *PyAutoGUI*) sollten bei zukünftigen Projekten fest eingeplant werden, um bessere Stabilität und Nutzerfreundlichkeit zu gewährleisten.

Zeitaufwändige Asset-Erstellung
Die Suche nach geeigneten Bildern für die Gärten nahm mehr Zeit in Anspruch als geplant. Eine klarere Ressourcenplanung oder die frühzeitige Einbindung einer Bilddatenbank hätte hier Abhilfe geschaffen.

Wichtige Erkenntnisse und erworbene Skills

Planungskompetenz:
Das Projekt zeigte deutlich, wie wichtig eine solide Vorplanung ist.

Technische Fähigkeiten:
Neue Frameworks (PyQt6, Pygame) und Tools (DALL·E, GIMP) wurden erlernt und erfolgreich eingesetzt.

Projekt- und Zeitmanagement:
Es wurde anerkannt, dass klare Meilensteine und eine bessere Prioritätensetzung erforderlich sind.

Ausblick:

Zukünftige Weiterentwicklungen könnten beinhalten:

- Integration von Windows-Benachrichtigungen, bspw. wenn ein Timer abgelaufen ist
- Möglichkeit Projekte zu archivieren, statt wie aktuell nur zu löschen
- Umfangreichere Statistiken zu den Projekten erstellen
- Scroll-Gärten ermöglichen, um so die Dimensionen eines Gartens zu erweitern
- Design bzw. visuelle Elemente sowohl in der GUI als auch in den Gärten überarbeiten