

Duale Hochschule Baden-Württemberg Mannheim

Praxisarbeit

Konzeption und Implementierung einer Companion App für Arc Raiders.

Studiengang Informatik

Studienrichtung Informationstechnik

Verfasser(in):	Paul Wegfahrt
Matrikelnummer:	2415837
Kurs:	TINF23IT1
Studiengangsleiter:	Prof Dr. Gerhards
Wissenschaftlicher Betreuer:	Jürgen Schultheis
Bearbeitungszeitraum:	14.10.2025 – 14.04.2026
Eingereicht am:	14.04.2026

Ehrenwörtliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit mit dem Titel "*Konzeption und Implementierung einer Companion App für Arc Raiders.*" selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort, Datum

Paul Wegfahrt

Sperrvermerk

Der Inhalt dieser Arbeit darf weder als Ganzes noch in Auszügen Personen außerhalb des Prüfungsprozesses und des Evaluationsverfahrens zugänglich gemacht werden, sofern keine anders lautende Genehmigung der Ausbildungsstätte vorliegt.

Inhaltsverzeichnis

Abbildungsverzeichnis	v
Quelltextverzeichnis	vi
Abkürzungsverzeichnis	vii
Abstract	viii
Zusammenfassung	ix
1 Einleitung	1
1.1 Motivation	1
1.2 Problemstellung	2
2 Aufgabenstellung und Entwicklungsmethodik	3
2.1 Operationalisierung der Problemstellung	3
2.2 Entwicklungsmethodik und Phasenmodell	4
2.2.1 Phase 1: Requirements Engineering	4
2.2.2 Phase 2: Architektur und Design	6
2.2.3 Phase 3: Implementierung in Iterationen	7
2.2.4 Phase 4: Evaluation und Reflexion	7
2.3 Methoden und Verfahren	9
2.3.1 Verwendete Tools und Technologien	9
3 Grundlagen	10
3.1 Arc Raiders & Gaming Tools	10
3.1.1 Companion Applications im digitalen Ökosystem	10
3.1.2 Gaming Companion Apps: Funktionale Kategorisierung	10
3.1.3 Extraction Shooter: Genre-spezifische Anforderungen	11
3.1.4 Arc Raiders: Technische Charakteristika und Systeme	11
3.1.5 Referenzimplementierungen: Tarkov Companion Ecosystem	12
3.2 Moderne Web-Technologien	12
3.2.1 TypeScript	13
3.2.2 React	14
3.2.3 Full Stack React Frameworks	15
3.3 UI/UX Frameworks & Design System	17
3.3.1 Tailwindcss	17
3.3.2 Moderne Komponenten-Bibliotheken	17
3.3.3 React-Flow & Dagre	17
3.4 State Management & Data Fetching	17
3.4.1 State Management/State Stores	17
3.4.2 react-query / tanstack-query als Daten-Fetching Library	18

3.5	Datenbank und Backend Design	18
3.5.1	Supabase	18
3.5.2	Orm - Drizzle	18
3.6	Deployment & DevOps	18
3.6.1	Git basierter Workflow	18
3.6.2	Vercel	18
3.7	Testing Frameworks & Strategien	19
3.7.1	Vitest	19
3.7.2	Cypress	19
4	Durchführung	20
4.1	Anforderungsanalyse	20
4.2	Vorbereitung	20
4.3	Implementierung	20
4.4	Testen	20
5	Ergebnisse und Diskussion	21
5.1	Objektivierung der Ergebnisse	21
5.2	Diskussion?	21
5.3	Marktanalyse, vorhandene Tools/Anwendungen	21
6	Fazit und Ausblick	22
Anhang		
Literatur		23

Abbildungsverzeichnis

Quilltextverzeichnis

Abkürzungsverzeichnis

DHBW	Duale Hochschule Baden-Württemberg
PvPvE	Player versus Player versus Entities
API	Application Programming Interface
TTI	Time to Interactivity
MVP	Minimum Viable Product
ER	Entity-Relationship
MCDA	Multi-Criteria Decision Analysis
DDD	Domain-Driven Design

Abstract

Abstract...

Zusammenfassung

Zusammenfassung....

1 Einleitung

1.1 Motivation

Der globale Gaming-Markt verzeichnet ein beispielloses Wachstum: Mit einer Bewertung von 298,98 Milliarden USD im Jahr 2024 wird prognostiziert, dass der Markt bis 2030 auf 600,74 Milliarden USD anwachsen wird [24]. Diese Entwicklung zeigt deutlich, dass Anwendungen in diesem Bereich ein großes Potenzial haben.

Innerhalb dieses dynamischen Marktes hat sich der Extraction Shooter als besonders anspruchsvolles und komplexes Genre etabliert. Spiele wie Escape from Tarkov und Hunt: Showdown definieren dieses Genre durch ihre charakteristischen Merkmale: hochriskante Player versus Player versus Entities (PvPvE)-Gameplay-Mechaniken, komplexe Progressionssysteme mit zahlreichen Quest-Lines, ressourcenbasierte Upgrade-Systeme und die permanente Gefahr des Verlusts aller mitgeführten Items bei einem Spieltod. Arc Raiders, entwickelt von Embark Studios und im Oktober 2025 veröffentlicht, positioniert sich als ambitionierter Vertreter dieses Genres mit dem Ziel, die Komplexität von Tarkov mit einer zugänglicheren Spielerfahrung zu verbinden.

Die inhärente Komplexität von Extraction Shootern stellt Spieler jedoch vor erhebliche Herausforderungen: Multiple Quest-Lines mit unterschiedlichen Zielen und Abhängigkeiten, begrenzter Inventarplatz, knappe Ressourcen und die Notwendigkeit koordinierter Squad-Basierter Strategien erfordern ein hohes Maß an Planung und Informationsmanagement. Companion Apps haben sich in der Gaming-Industrie als effektive Lösung etabliert, um solche Komplexitäten zu bewältigen. Es bestehen zahlreiche Beispiele aus verschiedenen Genres wie League of Legends, Destiny 2 oder eben Extraction-Shootern wie Escape from Tarkov, welche demonstrieren, wie externe Anwendungen das Spielerlebnis durch Statistik-Tracking, Ressourcenmanagement und Team-Koordination signifikant verbessern können [13] [6].

1.2 Problemstellung

Spieler von Arc Raiders sehen sich mit mehreren miteinander verknüpften Herausforderungen konfrontiert:

Informationsasymmetrie und mangelnde Übersicht: Das Spiel bietet zahlreiche Quest-Lines mit unterschiedlichen Zielen, zeigt jedoch nur die aktiven Quests an. Spieler haben dadurch keinen vollständigen Überblick über verfügbare Quests, deren Abhängigkeiten, den optimalen Pfad zur Erfüllung ihrer Ziele oder benötigter Materialien für die Zukunft. Diese Informationsfragmentierung erschwert strategische Planung und führt zu ineffizienten Entscheidungen.

Ressourcenmanagement: Die Upgrade-Systeme für Workstations erfordern multiple Ressourcentypen über mehrere Stufen hinweg. Bei begrenztem Inventarplatz und knappen Ressourcen fehlen Spielern Werkzeuge zur Kalkulation benötigter Materialien und zur Priorisierung von Upgrades basierend auf ihren individuellen Spielzielen.

Squad-Koordination: Arc Raiders basiert auf Squad-orientiertem Gameplay, doch wenn jedes Squad-Mitglied nur seine eigenen Ziele verfolgt, entstehen Konflikte bei der Routenplanung und Ressourcenverteilung. Die fehlende zentrale Übersicht über Squad-Ziele behindert effektive Koordination und optimale Ressourcennutzung.

Fehlen offizieller Planungstools: Zum aktuellen Stand (November 2025) existieren keine offiziellen Tools oder Application Programming Interface (API) zur Lösung dieser Probleme. Daten werden von Spielern über diverse Plattformen gesammelt und bereitgestellt.

Diese Problemstellung ist nicht singulär für Arc Raiders, sondern repräsentativ für die Herausforderungen moderner Extraction Shooter mit komplexen Meta-Progression-Systemen. Die Lösung dieser Probleme durch eine dedizierte Companion App könnte somit über Arc Raiders hinaus als Referenzimplementierung für ähnliche Spiele dienen.

2 Aufgabenstellung und Entwicklungsmethodik

2.1 Operationalisierung der Problemstellung

Die in Abschnitt 1.2 identifizierten Herausforderungen für Spieler von Arc Raiders – Informationsasymmetrie bei Quest-Lines, ineffizientes Ressourcenmanagement und unzureichende Squad-Koordination – werden durch systematische Operationalisierung in konkrete Entwicklungsaufgaben überführt. Operationalisierung bezeichnet dabei die systematische Ableitung von Services und technischen Constraints aus übergeordneten Zielen [18].

Die identifizierten Probleme lassen sich wie folgt operationalisieren:

- Problem “Informationsasymmetrie und mangelnde Übersicht”
 - Ziel “Vollständiger Überblick über Quest-Lines und Abhängigkeiten”
 - Service “Quest Tracking mit Kanban/Flow-Chart-Visualisierung”
 - Technische Anforderung “Datenmodell für Quest-Abhängigkeiten, Filterung und Statusverwaltung”
- Problem “Ressourcenmanagement bei begrenztem Inventar”
 - Ziel “Optimierte Materialnutzung und Upgrade-Priorisierung”
 - Service “Material Calculator & Workstation Planner”
 - Technische Anforderung “Berechnung für Upgrade-Kosten über multiple Stufen”
- Problem “Fehlende Squad-Koordination”
 - Ziel “Zentrale Übersicht über Squad-Ziele und effiziente Routenplanung”
 - Service “Squad-basierte Routenoptimierung mit interaktiven Karten”
 - Technische Anforderung “Darstellung von Zielen und Karten mit Tools zur Routenoptimierung”

Diese Operationalisierung adressiert das in Abschnitt 1.2 beschriebene Fehlen offizieller Planungstools und nutzt die von der Community bereitgestellten Daten als Grundlage. Sie bildet die methodische Basis für die nachfolgend beschriebene Entwicklung einer Companion App, die als Referenzimplementierung für ähnliche Extraction Shooter dienen kann.

2.2 Entwicklungsmethodik und Phasenmodell

2.2.1 Phase 1: Requirements Engineering

In agilen Entwicklungsumgebungen ist Requirements Engineering integraler Bestandteil zur Sicherstellung, dass sich entwickelnde Bedürfnisse und Erwartungen der Stakeholder während des gesamten Entwicklungsprozesses erfasst werden [1].

1.1 Anforderungserhebung (Requirements Elicitation)

- **1.1.1 Empirische Datenerhebung aus Spieltests:** Die in Abschnitt 1.1 erwähnte Teilnahme an einem Spieltest von Arc Raiders bildet die empirische Grundlage für die Anforderungserhebung. Durch systematische Beobachtung und Protokollierung werden konkrete Pain Points bei Quest-Management und Ressourcenplanung identifiziert.
- **1.1.2 Stakeholder-Interviews:** Durchführung direkter Gespräche mit Stakeholdern zur Extraktion von Bedürfnissen und Ideen [36]. Strukturierte Interviews mit Arc Raiders-Spielern verschiedener Erfahrungsstufen sowie gemeinsame Brainstorming-Sessions zur Feature-Ideenfindung ermöglichen die Erfassung spezifischer Anforderungen für Squad-basiertes Gameplay. Die in Abschnitt 1.2 identifizierten Problemstellungen werden dabei validiert und präzisiert.
- **1.1.3 Comparative Analysis:** Wie in Abschnitt 1.1 dargelegt, haben sich Companion Apps in der Gaming-Industrie als effektive Lösung etabliert. Die systematische Analyse umfasst primär etablierte Companion Apps für Extraction Shooter. Dies identifiziert Standard-Features und Innovationspotenziale, während die heuristische Evaluation von UI/UX-Patterns für komplexe Informationsdarstellung Best Practices aufzeigt.
- **1.1.4 Ableitung von User Stories:** Transformation der identifizierten Nutzerbedürfnisse in User Stories nach dem Format „Als [Rolle] möchte ich [Funktion], um [Nutzen] zu erreichen“. Beispiele bezogen auf die Problemstellung:
 - „Als Spieler möchte ich alle verfügbaren Quests und deren Abhängigkeiten sehen, um meine Ziele zu planen“,
 - „Als Squad-Leader möchte ich die Quest-Ziele meiner Teammitglieder auf einer Karte visualisieren, um eine effiziente Route für das gesamte Team zu planen“
 - „Als Spieler möchte ich kalkulieren, welche Materialien ich für geplante Workstation-Upgrades, Quests sowie Projekte benötige, um mein begrenztes Inventar optimal zu nutzen“.

1.2 Anforderungsanalyse (Requirements Analysis)

- **1.2.1 Kategorisierung und zeitliche Strukturierung:** Gruppierung in funktionale Anforderungen nach Seite (z.B. Dashboard, Quests, Workstations) und Implementierungsphase (z.B. P1-Visualization, P2-Progression) sowie nicht-funktionale Anforderungen (Performance, Usability, Verfügbarkeit). Identifikation von Abhängigkeiten zwischen Anforderungen (z.B. Routenplanung benötigt Maps) zur Planung der Implementierungsreihenfolge.
- **1.2.2 Priorisierung:** Implementierung der höchstpriorisierten Anforderungen zuerst zur Maximierung des Stakeholder-ROI [2]. Anwendung der MoSCoW-Methode (Must, Should, Could, Won't) mit Bewertung nach Business Value (Lösung der Kernprobleme aus Abschnitt 1.2) und technischer Komplexität. Dokumentation in Form eines Kanban Boards sowie Gantt-Diagramms unter Berücksichtigung von Abhängigkeiten zwischen Features.
- **1.2.3 Spezifikation von Akzeptanzkriterien:** Definition messbarer Erfolgskriterien für jede User Story nach SMART-Kriterien (Specific, Measurable, Achievable, Relevant, Time-bound). Beispiele umfassen:
 - „Quest-Suche liefert Ergebnisse in <500ms Time to Interactivity (TTI) für 95% der Anfragen“
 - „Material Calculator berechnet Upgrade-Kosten für beliebige Kombinationen von Workstations korrekt“.

1.3 Anforderungsvalidierung

- **1.3.1 Prototyping:** Erstellen von Minimum Viable Product (MVP)-Prototypen zur Visualisierung und Validierung der Anforderungen mit Stakeholdern. Interaktive Mockups und Wireframes ermöglichen frühes Feedback zu UI/UX-Designs und Funktionalitäten, um sicherzustellen, dass die entwickelten Lösungen den identifizierten Bedürfnissen entsprechen.
- **1.3.3 Abgleich mit Problemstellung:** Systematischer Abgleich der definierten Anforderungen mit der ursprünglichen Problemstellung zur Sicherstellung vollständiger Abdeckung aller drei Hauptherausforderungen: Informationsasymmetrie bei Quest-Lines, ineffizientes Ressourcenmanagement und unzureichende Squad-Koordination.

2.2.2 Phase 2: Architektur und Design

2.1 Systemarchitektur

- **2.1.1 Architekturentwurf:** Definition der Systemgrenzen und Komponenten (Frontend, Backend, Datenbank, externe Datenquellen) sowie Auswahl geeigneter Architekturmuster unter Berücksichtigung der Komplexität. Die Dokumentation erfolgt durch Architekturdiagramme nach dem C4-Modell (Context, Container, Component, Code), um verschiedene Abstraktionsebenen abzubilden und Stakeholdern unterschiedliche Detailgrade zu ermöglichen.
- **2.1.2 Technologie-Assessment:** Wie in Abschnitt 1.1 erwähnt, rechtfertigt das enorme Wachstumspotenzial des Gaming-Marktes die Wahl skalierbarer Technologien. Die systematische Evaluation umfasst Frontend-Frameworks (React, Vue, Angular), Backend-Technologien (Next.js API Routes, separate Backend-Lösung), Datenbank-Systeme (PostgreSQL, MongoDB, Firebase) sowie Hosting-Plattformen (Vercel, Netlify, AWS) unter Berücksichtigung von Kosteneffizienz für Hobby-Projekte. Multi-Criteria Decision Analysis (MCDA) ermöglicht die Technologieauswahl basierend auf Kriterien wie Performance, Entwicklerfreundlichkeit, Skalierbarkeit und Kosten.
- **2.1.3 Risikoanalyse:** Identifikation technischer Risiken, insbesondere die in Abschnitt 1.2 erwähnte Abhängigkeit von Community-bereitgestellten Daten statt offizieller API. Bewertung nach Eintrittswahrscheinlichkeit und Impact sowie Definition von Mitigationsstrategien: Backup-Strategien für Datenquellen (Web-Scraping des offiziellen Fandoms gemäß robots.txt, User-Generated Content mit Qualitätssicherung), Caching-Mechanismen zur Reduzierung der Abhängigkeit von externen Quellen sowie Validierung und Qualitätssicherung von Community-Daten.

2.2 Datenmodellierung

- **2.2.1 Konzeptionelle Modellierung:** Erstellung von Modellierungsdiagrammen wie Entity-Relationship (ER)-Diagrammen zur Abbildung der Hauptentitäten (Quests, Workstations, Materialien, Spielerprofile) und deren Beziehungen oder Context Diagrams aus dem Domain-Driven Design (DDD) zur Identifikation von Bounded Contexts. Berücksichtigung der in Abschnitt 1.2 beschriebenen Anforderungen an Flexibilität und Erweiterbarkeit für zukünftige Features.
- **2.2.2 Datenquellen-Strategie:** Aufgrund des in Abschnitt 1.2 beschriebenen Fehlens offizieller Tools müssen mehrere Community-Datenquellen miteinander verglichen und möglicherweise kombiniert werden. Hierbei wird ähnlich wie bei der Technologie-Assessment-Methode eine MCDA angewendet, um die Zuverlässigkeit, Aktualität und

Vollständigkeit der Datenquellen zu bewerten. Strategien zur Datenintegration und -synchronisation werden definiert, um eine konsistente und aktuelle Datenbasis sicherzustellen.

2.2.3 Phase 3: Implementierung in Iterationen

Agile Entwicklung betont die iterative Natur mit kontinuierlicher Verfeinerung und Validierung [8].

3.1 Entwicklung

- **3.1.1 Iterative Entwicklung:** Umsetzung der priorisierten User Stories in Iterationen unter Berücksichtigung von Clean-Code Prinzipien. Jede Iteration umfasst Planung, Implementierung, Testing und Review. Kontinuierliche Integration von Feedback aus Reviews zur Anpassung des Backlogs und Verbesserung der Implementierung.
- **3.1.2 Continuous Integration:** Automatisierte Builds bei jedem Commit über GitHub Actions oder ähnliche CI-Tools sowie automatisierte Testausführung der gesamten Test-Suite. Code Quality Checks umfassen Linting (ESLint für JavaScript/TypeScript). Automatisches Deployment auf Staging-Umgebung (z.B. Vercel Preview Deployments) für frühzeitiges Testen.

3.2 Review und Retrospektive

- **3.2.1 Review:** Demonstration implementierter Features an Stakeholder zur Einholung von Feedback bezüglich der Lösungen für die Probleme aus Abschnitt 1.2. Backlog-Refinement basierend auf gewonnenen Erkenntnissen sowie Anpassung der Priorisierung bei Bedarf.
- **3.2.2 Retrospektive:** Reflexion des Entwicklungsprozesses mit den Leitfragen: Was lief gut? Was kann verbessert werden? Identifikation von Verbesserungspotenzialen in Prozess, Definition konkreter Aufgaben für den nächsten Zyklus sowie Anpassung der Entwicklungspraktiken basierend auf Lessons Learned.

2.2.4 Phase 4: Evaluation und Reflexion

4.1 Quantitative Evaluation

- **4.1.1 Performance-Metriken:** Messung von Page Load Time durch Metriken wie der TTI für alle Hauptseiten. Messung der API-Antwortzeiten für kritische Endpunk-

te sowie Überwachung der Server- und Datenbank-Performance (CPU-, Speicher- und Datenbank-Latenz) unter Lastbedingungen.

- **4.1.3 User-Engagement-Metriken:** Sofern Veröffentlichung erfolgt: Daily Active Users (DAU), Feature Usage Statistics zur Identifikation der am häufigsten genutzten Funktionalitäten sowie User Retention Rate zur Bewertung des langfristigen Nutzens.

4.2 Qualitative Evaluation

- **4.2.2 User Experience Interviews:** Durchführung von User Experience Interviews mit Spielern zur Bewertung, ob die in Abschnitt 1.2 identifizierten Probleme gelöst wurden. Bewertung der Zielerreichung bezüglich Verbesserung der Quest-Übersicht, Effizienzsteigerung im Ressourcenmanagement sowie Optimierung der Squad-Koordination.
- **4.2.3 Vergleich mit Anforderungen:** Systematischer Vergleich der implementierten Features mit initialen Anforderungen und Akzeptanzkriterien. Identifikation von vollständig erfüllten, teilweise erfüllten und nicht erfüllten Anforderungen mit Begründung der Abweichungen.

4.3 Kritische Reflexion

- **4.3.1 Architektur-Bewertung:** Bewertung der gewählten Architektur hinsichtlich Eignung für die Anforderungen, Entwicklungseffizienz, Skalierbarkeit und Wartbarkeit sowie Diskussion von Trade-offs und alternativen Ansätzen. Reflexion, ob die Architekturentscheidungen im Kontext des in Abschnitt 1.1 beschriebenen Marktwachstums zukunftsfähig sind.
- **4.3.2 Technologie-Entscheidungen:** Diskussion des gewählten Technologie-Stacks (React/Next.js, Supabase/PostgreSQL, Vercel) mit Fokus auf Stärken, Schwächen und Lessons Learned. Reflexion der Datenhaltungsstrategie (Community-Daten, Caching, Synchronisation) und der in Abschnitt 1.2 beschriebenen Herausforderung einer fehlenden offiziellen API.
- **4.3.3 Lessons Learned:** Dokumentation gewonnener Erkenntnisse aus dem Entwicklungsprozess: erfolgreiche Praktiken, aufgetretene Herausforderungen und deren Lösungen sowie Empfehlungen für zukünftige Projekte. Reflexion der agilen Methodik und deren Eignung für die Entwicklung einer Gaming Companion App.
- **4.3.4 Generalisierbarkeit:** Bewertung, inwieweit die entwickelte Lösung als Referenzimplementierung für andere Extraction Shooter dienen kann, wie in Abschnitt 1.2 postuliert. Identifikation von spieldaten spezifischen Aspekten und übertragbaren Konzepten.

2.3 Methoden und Verfahren

"Welche Methoden und Verfahren werden verwendet?"

2.3.1 Verwendete Tools und Technologien

Verwendete Tools und Technologien mit Versionen/Datum (wie z.B. package.json aufbereiten), Ergebnis der Arbeit muss replizierbar sein

3 Grundlagen

3.1 Arc Raiders & Gaming Tools

3.1.1 Companion Applications im digitalen Ökosystem

Companion Applications haben sich als eigenständige Software-Kategorie etabliert, die primäre Anwendungen durch zusätzliche Funktionalität ergänzt, ohne das Hauptprodukt zu ersetzen. Im Gaming-Kontext erfüllen diese Anwendungen mehrere Schlüsselfunktionen: Sie erweitern die Spielerfahrung über die Session hinaus, bieten Planungs- und Analysewerkzeuge und ermöglichen Social Features außerhalb der Spielumgebung.

Companion Apps für Gaming-Konsolen verzeichneten seit 2020 ein signifikantes Wachstum, wobei die PC-Gaming-Plattform Steam mit etwa 33,4 Millionen Downloads im zuletzt gemessenen Quartal führend war [5]. Diese Entwicklung zeigt, dass Spieler bereit sind, zusätzliche Tools zu nutzen, um ihre Gaming-Erfahrung zu optimieren.

Die technische Architektur von Companion Apps variiert je nach Anwendungsfall:

- **Offizielle Plattform-Apps:** Direkte Integration mit Herstellersystemen (PlayStation App, Xbox App, Steam Mobile)
- **Game-spezifische Tools:** Fokus auf ein einzelnes Spiel oder Franchise
- **Community-getriebene Lösungen:** Von Spielern entwickelte Third-Party-Tools

3.1.2 Gaming Companion Apps: Funktionale Kategorisierung

Basierend auf der Analyse existierender Lösungen lassen sich Gaming Companion Apps in folgende funktionale Kategorien einteilen:

Informations- und Datenbank-Tools: Diese Tools bieten Zugriff auf Spieldaten wie Item-Statistiken, Charakterwerte oder Mechaniken. Apps wie Handbook for EFT bieten Spielern offline verfügbare Informationen zu Karten, Munitionsvergleichen, Waffen-Performance, Ausrüstung, Quest-Guides und Key-Informationen [20].

Progress-Tracking-Systeme: Apps wie The Hideout: Tarkov Sidekick ermöglichen Quest-Tracking, Hideout-Modul-Verwaltung und Team-Quest-Tracking, wo Spieler den Status ihrer Freunde sehen können [14].

Markt- und Wirtschafts-Tools: Funktionen wie Flea Market-Preisanzeige, Preishistorien bis zu einem Jahr zurück und Preis-Alerts für Flea Market-Preise helfen Spielern bei wirtschaftlichen Entscheidungen [14].

Karten- und Navigationshilfen: Interactive Maps-Apps bieten detaillierte, community-erstellte Karten mit Loot-Spots, Extraktionspunkten, Key-Locations und Quest-Details [21].

Build-Planer und Kalkulatoren: Weapon Builder und Damage Calculator („Tarkov'd Simulator“) ermöglichen theoretisches Durchspielen von Szenarien vor der Implementierung im Spiel [14].

3.1.3 Extraction Shooter: Genre-spezifische Anforderungen

Extraction Shooter basieren auf Konzepten der Verlustaversion und verzögerten Gratifikation, wobei jeder Raid ein Risiko darstellt und der erfolgreiche Abschluss eines wertvollen Durchgangs intensive dopamingesteuerte Befriedigung freisetzt [16]. Diese psychologischen Mechanismen erzeugen spezifische Anforderungen an unterstützende Tools.

Risiko-Management: Die Permadeath-Mechanik von Extraction Shootern erfordert sorgfältige Planung. Companion Apps können helfen, Risiken zu minimieren durch:

- Vorausschauende Ressourcenplanung
- Optimale Route-Planung zur Minimierung von Begegnungen
- Wert-Kalkulation von Loot vs. Risiko

Komplexitätsreduktion: Das Extraction-Shooter-Genre gilt aufgrund seiner Komplexität als schwer zugänglich für den Massenmarkt [15]. Arc Raiders hat durch seine Betonung von Teamsynergien und intuitiveren Spielerführungssystemen einen zugänglicheren Einstiegspunkt geschaffen [16].

Der aktuelle Markt konzentriert sich auf Escape from Tarkov (ca. 60.000 gleichzeitige Nutzer), Hunt Showdown (ca. 20.000) und Dark and Darker (ca. 10.000), zusammen etwa 100.000 gleichzeitige Nutzer [15]. Die Herausforderung für neue Titel liegt in der Etablierung eigener Tool-Ökosysteme.

3.1.4 Arc Raiders: Technische Charakteristika und Systeme

ARC Raiders ist ein 2025 veröffentlichter Third-Person-Extraction-Shooter, entwickelt mit der Unreal Engine 5 für PlayStation 5, Windows und Xbox Series X/S [37]. Bis zum 11. November 2025 hatte das Spiel weltweit über vier Millionen Exemplare verkauft [37], was eine signifikante Nutzerbasis für Companion-Tools darstellt.

Das Spiel implementiert mehrere Systeme, die durch externe Tools unterstützt werden können:

Quest-System: Spieler erfüllen Quests für Händler mit unterschiedlichen Motiven und Agen- den, was sich in komplexen Quest-Chains mit Abhängigkeiten manifestiert [9]. Das Ingame- Interface zeigt nur aktive Quests, was einen Gesamtüberblick erschwert.

Crafting und Ressourcen: Spieler müssen Workshop-Stationen upgraden und Blueprints lernen, um fortgeschrittenere Items zu craften [9]. Bei begrenztem Inventarplatz ist strategisches Ressourcen-Management essentiell.

Skill-Progression: Der ARC Raiders Skill-Tree verzweigt sich in drei Pfade: Survival, Mobility und Conditioning [9], was Langzeitplanung erfordert.

Multiplayer-Koordination: Das Spiel unterstützt nahtloses Cross-Platform-Spiel zwischen PlayStation, Xbox und PC, wobei Spieler in Squads bis zu drei Personen oder solo spielen können [9].

3.1.5 Referenzimplementierungen: Tarkov Companion Ecosystem

Das Escape from Tarkov Ökosystem bietet wertvolle Erkenntnisse für die Entwicklung von Companion Apps.

Tarkov Companion als Overwolf-App bietet Quest-Management sortiert nach Händler, Location oder Status, Browse-Funktionen für Karten mit Extraktionen, Loot-Hotspots und Quest-Items sowie Key-Suche und Hideout-Upgrade-Tracking [30]. Diese Features repräsentieren den aktuellen Standard für Extraction-Shooter-Companion-Apps.

Tarkov.dev bietet ein freies, community-erstelltes und Open-Source-Ökosystem von Escape from Tarkov-Tools inklusive Informationen zu Items, Crafts, Barters, Maps, Loot-Tiers, Hideout-Profiten und einer freien API [28]. Die Verfügbarkeit einer API ermöglicht die Entwicklung vielfältiger Third-Party-Tools.

3.2 Moderne Web-Technologien

Die Wahl geeigneter Web-Technologien ist entscheidend für den langfristigen Erfolg komplexer Webanwendungen. Für die Entwicklung einer Arc Raiders Companion App kommen moderne, produktionsreife Technologien zum Einsatz, die sowohl Entwicklerproduktivität als auch Anwendungsperformance optimieren.

3.2.1 TypeScript

TypeScript hat sich als De-facto-Standard für moderne JavaScript-Entwicklung etabliert. Als Superset von JavaScript fügt TypeScript statische Typisierung und erweiterte Sprachfeatures hinzu, die besonders für große Projekte wertvoll sind [17].

Type Safety in großen Projekten: Type Safety ist eines der herausragenden Merkmale von TypeScript und adressiert eine kritische Limitierung des dynamischen Typsystems von JavaScript [17]. In größeren Anwendungen, wo mehrere Entwickler am selben Codebase arbeiten, können unterschiedliche Annahmen über Datentypen zu unerwartetem Verhalten und schwer auffindbaren Bugs führen [17]. TypeScript ermöglicht es Entwicklern, Typen explizit für Variablen, Funktionsparameter und Rückgabewerte zu definieren, wodurch potenzielle Fehler zur Compile-Zeit statt zur Laufzeit erkannt werden [17].

Organisationen übernehmen TypeScript zunehmend für Large-Scale-Anwendungen aufgrund seiner Fähigkeit, Fehler zur Compile-Zeit zu erkennen, wodurch Laufzeitfehler reduziert und die Code-Qualität verbessert wird [10]. Der strukturierte Ansatz von TypeScript hilft Teams, komplexe Codebases effizienter zu verwalten [10]. In größeren Teams verbessert Type Safety die Zusammenarbeit durch Reduzierung der kognitiven Last und Förderung klarerer Kommunikation [17].

Developer Experience: TypeScript verbessert nicht nur die Code-Qualität, sondern steigert auch signifikant die Developer Experience durch überlegenes Tooling und Error Reporting [17]. Die Integration mit modernen IDEs bietet erweiterte Features wie intelligente Code-Vervollständigung, automatisches Refactoring und Code-Navigation [7].

Tooling und IDE-Unterstützung für TypeScript erfuhr 2024 signifikante Verbesserungen, wobei Entwickler von besserem IntelliSense, Auto-Completion und Refactoring-Tools profitieren, was den Entwicklungsprozess reibungsloser und effizienter gestaltet [29]. TypeScript wird zunehmend mit modernen Frameworks wie React, Angular und Vue.js integriert, was Entwicklern ermöglicht, die Vorteile von TypeScripts Type-Checking zu nutzen und gleichzeitig die leistungsstarken Features dieser Frameworks für den Aufbau von Benutzeroberflächen zu verwenden [10].

Durch die Nutzung des statischen Typsystems von TypeScript können Entwickler sichereren und wartbareren Code schreiben und die Gesamtqualität und Zuverlässigkeit ihrer Anwendungen verbessern [7]. Mit dem Fokus auf statische Typisierung und Developer-Ergonomie ist TypeScript gut positioniert, um den sich entwickelnden Anforderungen der Webentwicklung auch zukünftig gerecht zu werden [4].

3.2.2 React

React hat sich als führende JavaScript-Bibliothek für den Aufbau von Benutzeroberflächen etabliert, primär für Single-Page-Applications (SPAs) [12]. Eines der wichtigsten Features von React ist seine komponentenbasierte Architektur, die Entwicklern ermöglicht, skalierbare und wartbare Anwendungen effizient zu erstellen.

Komponentenbasiertes UI: In React ist eine Komponente eine wiederverwendbare, eigenständige Einheit einer Benutzeroberfläche [12]. Komponenten ermöglichen es, eine Anwendung in kleinere, unabhängige Teile zu zerlegen, die effizient verwaltet und wiederverwendet werden können [12]. Diese modulare Struktur macht die Anwendung einfacher zu entwickeln, zu warten und zu skalieren, da Komponenten über verschiedene Teile der App oder sogar in unterschiedlichen Projekten wiederverwendet werden können [12].

Jede React-Anwendung besteht aus einem Baum von Komponenten, wobei jede Komponente ihre eigene Logik, ihren State und ihre UI-Repräsentation hat [12]. Die Vorteile dieser Architektur umfassen:

- **Wiederverwendbarkeit:** Komponenten können mehrfach in verschiedenen Teilen einer Anwendung verwendet werden
- **Modularität:** Jede Komponente handelt ein spezifisches Stück Funktionalität, was die Anwendung strukturierter macht
- **Skalierbarkeit:** Große Anwendungen können durch Zusammensetzen kleinerer, wieder verwendbarer Komponenten entwickelt werden
- **Wartbarkeit:** Updates und Bugfixes sind einfacher, da Änderungen auf spezifische Komponenten lokalisiert sind [12]

Komposition over Inheritance: React verfügt über ein leistungsstarkes Kompositionsmo dell, und die Verwendung von Komposition anstelle von Vererbung wird empfohlen, um Code zwischen Komponenten wiederzuverwenden [11]. Bei Facebook verwenden die Entwickler React in tausenden von Komponenten, und es wurden keine Anwendungsfälle gefunden, bei denen die Erstellung von Komponenten-Vererbungshierarchien empfohlen würde [11]. Props und Komposition bieten die gesamte Flexibilität, die benötigt wird, um das Aussehen und Verhalten einer Komponente auf explizite und sichere Weise anzupassen [11].

Komposition ist eine Technik, bei der eine Komponente durch Zusammensetzen anderer Komponenten aufgebaut wird, ähnlich wie man ein Lied aus verschiedenen musikalischen Noten komponieren würde [27]. React fördert die Verwendung von Komposition anstelle von Vererbung aus mehreren Gründen:

- **Flexibilität:** Komposition gibt mehr Flexibilität bei der gemeinsamen Nutzung von Funktionalität zwischen Komponenten [27]

- **Einfachheit:** Sie fördert einfache Hierarchien mit Komponenten, die isoliert verstanden werden können, ohne eine Vererbungskette kennen zu müssen [27]
- **Wiederverwendbarkeit:** Für Komposition entworfene Komponenten sind oft einfacher wiederzuverwenden, da sie keine Annahmen über den Kontext treffen, in dem sie verwendet werden [27]
- **Vermeidung von Tight Coupling:** Vererbung kann zu enger Kopplung zwischen Komponenten führen, was die Codebasis fragil und schwer zu refaktorieren macht [27]

React fördert Komposition gegenüber Vererbung, weil sie größere Flexibilität und Trennung von Belangen ermöglicht [19]. Vererbung kann manchmal zu enger Kopplung zwischen Komponenten führen, was es schwieriger macht, Anwendungen zu modifizieren oder zu skalieren [19]. Durch die Übernahme von Komposition anstelle von Vererbung können Entwickler die Wiederverwendbarkeit von Komponenten vereinfachen und ihren Code flexibler und wartbarer gestalten [19].

3.2.3 Full Stack React Frameworks

Während React als UI-Bibliothek exzelliert, benötigen produktionsreife Anwendungen zusätzliche Funktionalitäten wie Routing, Server-Side Rendering und Daten-Fetching. Full-Stack React Frameworks wie Next.js adressieren diese Anforderungen durch Bereitstellung einer kompletten Lösung für moderne Webanwendungen.

Next.js als React-Backend-Framework: Next.js hat sich zu einem Kraftpaket für den Aufbau performanter und SEO-freundlicher React-Anwendungen entwickelt [26]. Der App Router ist ein dateibasierter Router, der Reacts neueste Features wie Server Components, Suspense und Server Functions nutzt [33]. Next.js verwendet ein dateisystembasiertes Routing, bei dem Ordner zur Definition von Routen verwendet werden [35]. Jeder Ordner repräsentiert ein Routen-Segment, das einem URL-Segment zugeordnet wird [35].

File-Based Routing: Mit dem App Router ermutigt Next.js zu einem dateibasierten Routing-System, bei dem die Verzeichnisstruktur die URL-Struktur widerspiegelt [22]. In Next.js nutzt der App Router das dateibasierte Routing, was bedeutet, dass die Position der `page.tsx`- oder `route.ts`-Datei innerhalb des app-Verzeichnisses definiert, wie sie auf eine gegebene URL abgebildet wird [23].

Next.js folgt weiterhin dem dateibasierten Routing, aber mit der Einführung des App Routers haben Dateien und Ordner nun strikt definierte Rollen [3]:

- **Ordner:** Ordner definieren die Routen der Anwendung. Ein Routen-Segment ist ein Pfad vom Root-Ordner bis zu einem Blatt-Ordner, der eine `page.ts`-Datei enthält [3]
- **Dateien:** Dateien erstellen die UI für ein Routen-Segment [3]

Eine spezielle `page.ts`-Datei wird verwendet, um Routen-Segmente öffentlich zugänglich zu machen [35]. Dynamic Route Segments können durch Umschließen des Ordnernamens mit eckigen Klammern erstellt werden, wie `[productId]` [3].

Rendering-Strategien: Next.js ist zu einem führenden Framework geworden, indem es verschiedene Rendering-Strategien anbietet: Static Site Generation (SSG), Server-Side Rendering (SSR), Client-Side Rendering (CSR), Incremental Static Regeneration (ISR) und das experimentelle Partial Prerendering (PPR) [32]. Diese wurden entwickelt, um Performance, SEO und User Experience in verschiedenen Situationen zu optimieren [32].

SSG (Static Site Generation): Bei SSG wird die initiale HTML zur Build-Zeit generiert, was zu ultra-schnellen Ladezeiten führt [26]. SSG ist ideal für Inhalte, die sich nicht häufig ändern, wie Blog-Posts, Dokumentation oder Marketing-Seiten [26]. Im App Router-Modell rendert SSG automatisch jede Komponente statisch, die keine server-spezifischen Funktionen nutzt [31].

SSR (Server-Side Rendering): SSR generiert HTML auf dem Server für jede Anfrage [26]. Diese Strategie ist optimal für echtzeitbezogene, nutzerspezifische Inhalte wie personalisierte Dashboards, Account-Profile oder News-Feeds [31]. SSR garantiert, dass jeder Besucher bei jedem Laden der Seite die neuesten Daten sieht [31].

ISR (Incremental Static Regeneration): ISR baut auf SSG auf und fügt die Fähigkeit hinzu, Inhalte zu aktualisieren, ohne einen vollständigen Rebuild der Site zu erfordern [26]. Bei ISR wird weiterhin die initiale HTML zur Build-Zeit generiert, aber Next.js wird auch mitgeteilt, wie oft nach Updates geprüft und die HTML-Dateien revalidiert werden sollen [26]. ISR ist geeignet für Inhalte, die sich periodisch ändern, wie Blog-Posts oder Produkt-Listings [31].

PPR (Partial Prerendering): PPR ist eine der neuesten Ergänzungen zu Next.js und befindet sich derzeit im experimentellen Status [31]. PPR ermöglicht es, dass eine Seite teilweise mit statischen und dynamischen Segmenten kombiniert pre-gerendert wird [31]. Dies ist besonders nützlich für Seiten mit Sektionen, die progressiv laden können, während kritischer Content sofort erscheint [31].

Partial Prerendering kombiniert ultra-schnelle statische Edge-Delivery mit vollständig dynamischen Fähigkeiten und hat das Potenzial, das Standard-Rendering-Modell für Webanwendungen zu werden [34]. PPR bietet ein vereinheitlichtes Modell, das die Zuverlässigkeit und Geschwindigkeit von Incremental Static Regeneration (ISR) mit den dynamischen Fähigkeiten von Server-Side Rendering (SSR) verbindet [34].

PPR basiert auf React Suspense Boundaries: zur Build-Zeit wird der gesamte Content bis zur Suspense-Boundary zusammen mit den Fallbacks statisch generiert und suspendiert das Rendering zur Build-Zeit [25]. Zur Request-Zeit wird die pre-gerenderte statische Shell sofort an den Client geliefert, während Next.js das Rendering dort fortsetzt, wo es zur Build-Zeit

suspendiert wurde [25]. Sobald die suspendierten Children auflösen, wird die UI zum Client in derselben Response gestreamt [25].

Bei der Entscheidung für eine Rendering-Strategie sollten folgende Faktoren berücksichtigt werden: Wie oft ändert sich dieser Content? SSG ist gut für statische Inhalte, ISR ist hervorragend für periodisch wechselnde Inhalte, und SSR oder CSR ist am besten für Echtzeit-Daten [32]. Next.js ermöglicht Entwicklern, verschiedene Rendering-Methoden innerhalb einer einzelnen Anwendung zu nutzen, je nach Bedarf, auf Seiten-Basis [32].

3.3 UI/UX Frameworks & Design System

3.3.1 Tailwindcss

Utility First CSS Framework, inline

v4 mit "Just in Time" Compiler und global.css

3.3.2 Moderne Komponenten-Bibliotheken

Shadcn/using

unterschied zu "klassischen" Bibliotheken wie MUI bootstrap

3.3.3 React-Flow & Dagre

Visualisierung von Graphen/Netzwerken

Layout Algorithmen (Dagre)

3.4 State Management & Data Fetching

3.4.1 State Management/State Stores

Bibliotheken wie zustand

client side

3.4.2 react-query / tanstack-query als Daten-Fetching Library

Server State Management

Caching Strategien

Optimistic Updates

3.5 Datenbank und Backend Design

3.5.1 Supabase

PostgreSQL-basiert

Real-time Capabilities

Row Level Security

edge functions (Serverless Functions)

3.5.2 Orm - Drizzle

ORM's

Type-Safety

code-first approach

3.6 Deployment & DevOps

3.6.1 Git basierter Workflow

3.6.2 Vercel

Deployment von Nextjs Applikationen

weitere features

CI/CD Pipelines

3.7 Testing Frameworks & Strategien

3.7.1 Vitest

Unit und Integration Tests

fast, modern, built for TS

3.7.2 Cypress

End-to-End Testing

real browser testing

4 Durchführung

an T2000 orientieren? SZeige, wie du von der Problemstellung zu Requirements kommst
Traceability Matrix: Verknüpfen Anforderungen → Issues → Tests → Code"

4.1 Anforderungsanalyse

- Zielgruppenanalyse
- Funktionale Anforderungen
- Nicht-funktionale Anforderungen

4.2 Vorbereitung

- Modellierung
- Technologieentscheidungen (kurz)

4.3 Implementierung

- Architektur
- Beispiel Datenbankmodell, edge function
- Beispiel API Endpunkte
- Beispiel Frontend Komponenten

4.4 Testen

- Teststrategie/-umgebung
- Beispiel Test
- Ergebnisse

5 Ergebnisse und Diskussion

Metriken definieren: Wie misst du Erfolg? (Performance, Usability, Code-Qualität)

5.1 Objektivierung der Ergebnisse

(Tabelle, Aufgabe, erledigt?, verifiziert?)

5.2 Diskussion?

Stellungnahme zu den Ergebnissen, aufzeigen von Alternativen

5.3 Marktanalyse, vorhandene Tools/Anwendungen

(Parallelen zu Tools aus anderen Spielen wie Tarkov?)

6 Fazit und Ausblick

Was kann man damit anfangen

Wie kann man es erweitern

Literatur

- [1] A. Abukhalaf et al. „Automated requirements engineering framework for agile model-driven development“. In: *Frontiers in Computer Science* 7 (2025). DOI: 10.3389/fcomp.2025.1537100.
- [2] Scott W. Ambler. *Agile Requirements Modeling: Strategies for Agile Teams*. Agile Modeling. 2023. URL: <http://agilemodeling.com/essays/agileRequirements.htm> (besucht am 27.01.2025).
- [3] Kulsum Ansari. „Next.js App Router: Routing“. In: (2024). Published: April 19, 2024. URL: <https://medium.com/@kulsumansari4/next-js-app-router-routing-8d795dbe324c> (besucht am 18.11.2024).
- [4] Rohit Bhatu. „Why Choose TypeScript in 2024: The Evolution of JavaScript Development“. In: (2024). Published: March 8, 2024. URL: <https://medium.com/@bhaturohit/why-choose-typescript-in-2024-the-evolution-of-javascript-development-fdf77dd56a62> (besucht am 18.11.2024).
- [5] J. Clement. *Number of gaming console and storefront companion app downloads worldwide from 2020 to 2024 YTD (in millions)*. Published: August 19, 2024. Statista. 2024. URL: <https://www.statista.com/statistics/1266736/game-console-companion-app-downloads/> (besucht am 18.11.2024).
- [6] Matthew Cochran. „Best Companion Apps In Video Games“. In: (2023).
- [7] Hassan Djirdeh. „Mastering TypeScript: Benefits and Best Practices“. In: *Telerik Blog* (2024). Published: November 01, 2024. URL: <https://www.telerik.com/blogs/mastering-typescript-benefits-best-practices> (besucht am 18.11.2024).
- [8] G. Ebirim et al. „Advancements and innovations in requirements elicitation“. In: *World Journal of Advanced Research and Reviews* 22.01 (2024), S. 1209–1220.
- [9] Embark Studios. *ARC Raiders on Steam*. Valve Corporation. 2025. URL: https://store.steampowered.com/app/1808500/ARC_Raiders/ (besucht am 18.11.2024).
- [10] Expedite Informatics. *TypeScript in 2024: Trends, Standards, Benefits, Challenges, and Commitments*. Expedite Informatics. 2024. URL: <https://expediteinformatics.com/typescript-in-2024-trends-standards-benefits-challenges-and-commitments/> (besucht am 18.11.2024).
- [11] Facebook Open Source. *Composition vs Inheritance*. React Documentation. 2024. URL: <https://legacy.reactjs.org/docs/composition-vs-inheritance.html> (besucht am 18.11.2024).
- [12] GeeksforGeeks. *React Component Based Architecture*. Published: July 23, 2025. GeeksforGeeks. 2025. URL: <https://www.geeksforgeeks.org/react-component-based-architecture/> (besucht am 18.11.2024).
- [13] Olga Giersza. „Companion Apps are Taking Video Games to the Next Level“. In: (2024).
- [14] Austin Hodak. *The Hideout: Tarkov Sidekick - Mobile Companion App for Escape from Tarkov*. Google Play Store. 2024. URL: <https://play.google.com/store/apps/details?id=com.austinhodak.thehideout> (besucht am 18.11.2024).

- [15] IconEra Community. *What will the Extraction Shooter market look like in two years? (market size)*. Published: November 20, 2024. IconEra Forums. 2024. URL: <https://icon-era.com/threads/what-will-the-extraction-shooter-market-look-like-in-two-years-market-size.14825/> (besucht am 18.11.2024).
- [16] Insider Gaming. „Love Them or Hate Them, Extraction Shooters Are Running Wild Right Now“. In: (2025). Published: 1 week ago from retrieval date. URL: <https://insider-gaming.com/love-or-hate-extraction-shooters-wild/> (besucht am 18.11.2024).
- [17] Invictarasolutions. *Why TypeScript is Becoming the Go-To for Large-Scale JavaScript Projects*. Published: October 27, 2024. Medium. 2024. URL: <https://medium.com/@invictarasolutions/why-typescript-is-becoming-the-go-to-for-large-scale-javascript-projects-5439aabf4bb7> (besucht am 18.11.2024).
- [18] Axel van Lamsweerde. „Requirements Engineering in the Year 00: A Research Perspective“. In: *Proceedings of the 22nd International Conference on Software Engineering (ICSE'00)*. IEEE Press, 2000, S. 5–19. DOI: 10.1145/337180.337184.
- [19] Mohi Mishra. „Composition vs Inheritance in React: Simplifying Component Reusability“. In: (2024). Published: April 2, 2024. URL: <https://medium.com/@mohimishra016/composition-vs-inheritance-in-react-simplifying-component-reusability-f978363bacf6> (besucht am 18.11.2024).
- [20] NehalemX. *Handbook for EFT - Escape from Tarkov Companion App*. Google Play Store. 2024. URL: <https://play.google.com/store/apps/details?id=com.nehalemx.tarkovwiki2> (besucht am 18.11.2024).
- [21] Oakcode. *Game Maps: Tarkov & More - Interactive Gaming Maps Desktop App*. Overwolf. 2024. URL: https://www.overwolf.com/app/W4rGo-Game_Maps_Escape_from_Tarkov (besucht am 18.11.2024).
- [22] Thiraphat Phutson. „Mastering Next.js App Router: Best Practices for Structuring Your Application“. In: (2024). Published: September 21, 2024. URL: <https://thiraphat-ps-dev.medium.com/mastering-next-js-app-router-best-practices-for-structuring-your-application-3f8cf0c76580> (besucht am 18.11.2024).
- [23] ProNextJS. *File-Based Routing with App Router*. ProNextJS Workshops. 2024. URL: <https://www.pronextjs.dev/workshops/next-js-foundations-for-professional-web-development~1xb18/file-based-routing-with-app-router~dtnrx> (besucht am 18.11.2024).
- [24] Grand View Research. *Video Game Market Size, Share And Growth Report*. 2024.
- [25] Nikhil Snayak. „Dissecting Partial Pre Rendering“. In: (2024). Published: July 6, 2024. URL: <https://www.nikhilsnayak.dev/blogs/dissecting-partial-pre-rendering> (besucht am 18.11.2024).
- [26] Ritesh Srivastava. „Next JS: Rendering Strategies — SSG, SSR, CSR and PPR“. In: (2025). Published: February 10, 2025. URL: <https://medium.com/@ritsrivastava/next-js-rendering-strategies-ssg-ssr-csr-and-ppr-d6243ec0ce72> (besucht am 18.11.2024).
- [27] StudyRaid. *Composition vs Inheritance - React - The Complete Guide*. StudyRaid. 2024. URL: <https://app.studyraid.com/en/read/1665/22475/composition-vs-inheritance> (besucht am 18.11.2024).

- [28] Tarkov.dev Community. *Tarkov.dev - A free, community made, and open source ecosystem of Escape from Tarkov tools and guides*. 2024. URL: <https://tarkov.dev/> (besucht am 18.11.2024).
- [29] Toxigon. *TypeScript Trends 2024: The Rise of Static Typing*. Published: March 21, 2025. Toxigon Blog. 2025. URL: <https://toxigon.com/typescript-trend-2024> (besucht am 18.11.2024).
- [30] TreeStn. *Tarkov Companion - Desktop App on Overwolf*. Overwolf. 2024. URL: https://www.overwolf.com/app/treestn-tarkov_companion (besucht am 18.11.2024).
- [31] Emmanuel Udeji. „Mastering Next.js 15: A Guide to App Router Rendering Strategies for Modern Web Applications“. In: (2024). Published: November 4, 2024. URL: <https://medium.com/@emmaudeji/mastering-next-js-15-a-guide-to-app-router-rendering-strategies-for-modern-web-applications-8a6683d8c348> (besucht am 18.11.2024).
- [32] Vercel. *How to choose the best rendering strategy for your app*. Vercel Blog. 2024. URL: <https://vercel.com/blog/how-to-choose-the-best-rendering-strategy-for-your-app> (besucht am 18.11.2024).
- [33] Vercel. *Next.js Docs: App Router*. Next.js Documentation. 2024. URL: <https://nextjs.org/docs/app> (besucht am 18.11.2024).
- [34] Vercel. *Partial prerendering: Building towards a new default rendering model for web applications*. Vercel Blog. 2024. URL: <https://vercel.com/blog/partial-prerendering-with-next-js-creating-a-new-default-rendering-model> (besucht am 18.11.2024).
- [35] Vercel. *Routing: Defining Routes*. Next.js Documentation. 2024. URL: <https://nextjs.org/docs/14/app/building-your-application/routing/defining-routes> (besucht am 18.11.2024).
- [36] Visure Solutions. *Requirements Gathering Techniques in Agile Software Engineering*. 2025. URL: <https://visuresolutions.com/alm-guide/requirements-gathering-techniques-for-agile> (besucht am 27.01.2025).
- [37] Wikipedia contributors. *ARC Raiders*. Wikipedia, The Free Encyclopedia. Last edited: 3 hours ago from retrieval date. 2025. URL: https://en.wikipedia.org/wiki/ARC_Raiders (besucht am 18.11.2024).