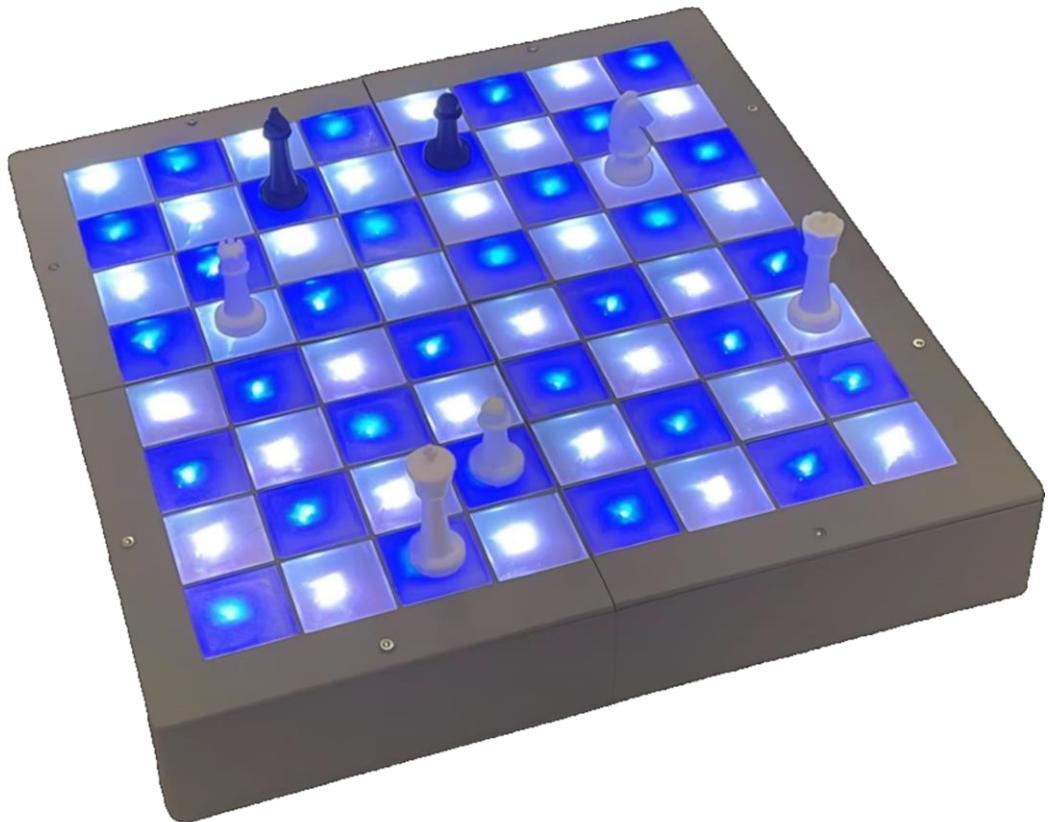


Hausmesse 2025

An der Walter-Rathenau Gewerbeschule Freiburg

Smart Chessboard



Luca Schreiner (33%)

Kevin Chmolenko (33%)

Bilal Karali (33%)

Inhalt

1 Projektauftrag	3
2 Grundkonzept.....	4
3 Projektorganisation.....	4
3.1 Ablauforganisation – Gantt Diagramm.....	4
3.2 Teamsitzungsprotokoll	5
4 Das Smart Chessboard.....	6
4.1 Umsetzung.....	6
4.2 Hardware	6
4.2.1 Reed-Schalter Matrix – Figuren-Registrierung	7
4.2.2 LED-Aufbau und Interface.....	8
4.3 Software	9
4.3.1 Backend-Software Architektur	9
4.3.2 Erkennung der Figuren und Auswertung der Rohdaten.....	10
4.3.3 Client-Server Schnittstelle	11
4.4 Website	12
4.4.1 Aufbau und Inhalt.....	12
4.4.2 Technische Umsetzung.....	13
5 Abschlussbericht, Eindrücke und Reflektion.....	15
6 Literaturverzeichnis	15
 Abbildung 1: Gantt-Diagramm	4
Abbildung 2: Schaltplan Reed-Schalter Matrix.....	7
Abbildung 3: Schaltplan der LED-Matrix	8
Abbildung 4: Funktion zum Initialisieren eines Schachbrettmusters.....	8
Abbildung 5: Backend-Softwarearchitektur.....	9
Abbildung 6: Multiplexing mit Python	10
Abbildung 7: Website-Startmenü	12

1 Projektauftrag

Projekttitle:	Smart Chessboard	
Projektteam:	Luca Schreiner, Kevin Chmolenko, Bilal Karali	
Projektdauer:	Geplanter Beginn: 21.07.2025 Geplantes Ende: Datum der Hausmesse	
Projektgesamtziel:	Erstellen eines physischen Schachbretts mit visuellen Zugvorschlägen und digitaler Spielansicht, mit Benutzeroberfläche.	
Projektteilziele und -ergebnisse:	Teilziele: Hardwareaufbau Schachbrettprototyp Backendlogik Schnittstelle Backend und Benutzeroberfläche Benutzeroberfläche	Ergebnisse: <ul style="list-style-type: none">• Ansteuerbare LEDs, Funktionierende Reed-Switch Matrix• Figuren und Stellungserkennung, Schachlogik, Zugauswertung und Validierung• Übermittlung Relevante Stellungs- und Zuginformationen an Benutzeroberfläche• Visualisierung des Analogen Schachbretts und Live-Update, Spielerauswahl
Meilensteine:	Meilensteine: Proof of Work Prototypaufbau Vollständiger Aufbau Benutzeroberfläche Umfangreiches Testen	Datum: 23.07.2025 23.07.2025 15.08.2025 15.09.2025 15.09.2025
Projektressourcen:	Ressourcen: PTH-LED Reed-Switches RaspberryPi 4 Leiterplatte Magneten Schachfiguren	Menge: 64 64 1 1 32 32
Projektbudget:	150,00€	
Projektrisiken und -unsicherheiten:	Zeitgemäße Materialbeschaffung, physische Unsicherheiten bei Figurenerkennung	
Sonstige relevante Informationen:	-	
Anlagen:	-	

Walther-Rathenau Gewerbeschule in Freiburg, 28.05.2025,
Luca Schreiner, Kevin Chmolenko, Bilal Karali

2 Grundkonzept

Das Grundkonzept ist das Erstellen eines physischen Schachbretts mit visuellen Zugvorschlägen und digitaler Spielansicht, mit Benutzeroberfläche als Website. Die Spieler sollen eine bestimmte Spielfigur aufheben können und die möglichen Spielzüge für diese Figur durch RGB-LEDs und einer Visualisierung auf der Website sehen können.

3 Projektorganisation

3.1 Ablauforganisation – Gantt Diagramm

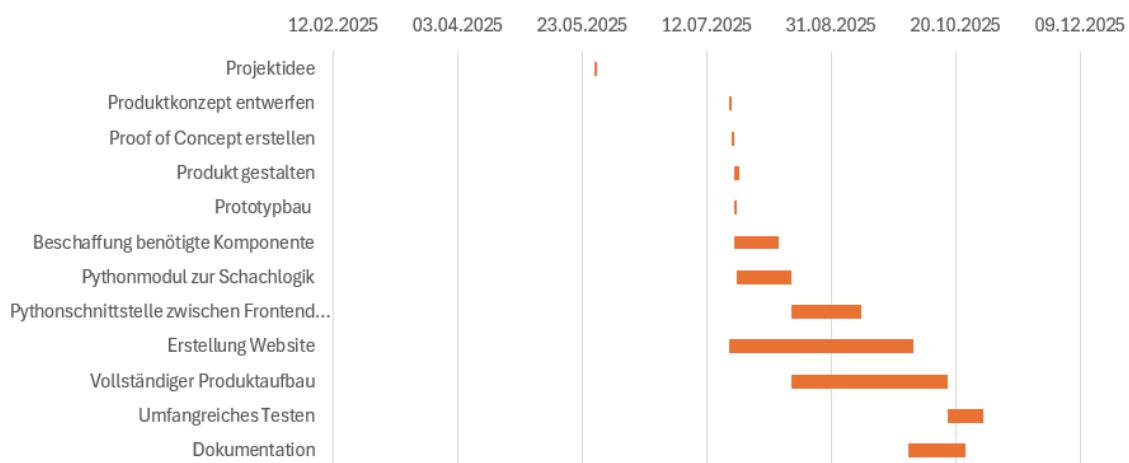


Abbildung 1: Gantt-Diagramm

3.2 Teamsitzungsprotokoll

Protokoll vom 21.07.25

„Smart Chessboard“

Ort:	Raum 029 – Walter-Rathenau Gewerbeschule
Besprechungsleitung:	Kevin Chmolenko
Protokollführender:	Bilal Karali
Anwesende:	Luca Schreiner Kevin Chmolenko Bilal Karali

Tagesordnung:

- I. Begrüßung und Verlesung der Tagesordnung
- II. Konzeptionierung
- III. Ermittlung benötigte Komponente
- IV. Beschluss Softwarestack
- V. Beschluss Projektdokumentationsmöglichkeiten und Projektmanagement

TOP	Ergebnis	Maßnahmen	Verantwortlich	Überprüfen am
I.	Kevin begrüßt die Teilnehmer und verließt die Tagesordnung	-	Kevin Chmolenko	-
II.	Produktentwurf und Umsetzungskonzept	<ul style="list-style-type: none"> • Diskussion über technische Möglichkeiten • Prototypentwurfsplan 	Luca Schreiner Kevin Chmolenko Bilal Karali	26.07.25
III.	Liste aller benötigten Bauteile erstellt	<ul style="list-style-type: none"> • Beschaffung durch Ausbildungsbetrieb 	Kevin Chmolenko	01.08.25
IV.	Einigung Python für Backend und React.js für Frontend	<ul style="list-style-type: none"> • Prüfen nach verschiedenen Faktoren (Schnelligkeit, Libaries) und vorhandenes Know-how 	Bilal Karali Luca Schreiner	heute
V.	Ablauforganisation erstellt	<ul style="list-style-type: none"> • Gantt-Diagramm (Siehe ..) • Kanban Board erstellt und Tasks gesammelt 	Kevin Chmolenko Luca Schreiner Bilal Karali	heute

26.07.25, Walter-Rathenau GS Freiburg
Datum, Ort

Bilal Karali
Unterschrift Protokollführender

4 Das Smart Chessboard

4.1 Umsetzung

Für die Umsetzung wird ein Raspberry Pi 4 Model B verwendet, zunächst aufgrund der hohen Rechenleistung und hohem Speicherplatz, aber auch hinsichtlich der Kompatibilität mit modernen Programmiersprachen wie Python.

4.2 Hardware

Die innerhalb dieses Projekts verwendete Hardware besteht aus:

- 4 eigengefertigte Leiterplattenfragmente
- 64 Reed-Schalter
- 64 adressierbare RGB-LEDs
- 16 10K-Ohm Pull-Down-Widerstände
- 4 3D-gedruckte Schachbrett Unterteil-Fragmente
- 4 3D-gedruckte Schachbrett Oberteil-Fragmente
- 64 transparente 3D-gedruckte Schachfelder
- 1 Raspberry Pi 4 Model B
- 1 3D-gedrucktes Schachfiguren Set mit Aussparung für Knopfmagnete
- 32 Knopfmagnete 10x3mm

Hierbei wurde sich für die Verwendung einer eigengefertigten Leiterplatte entschieden, um Aufwand in Hinsicht auf Verkabelungsarbeiten einzusparen und um einen kompakten und unkomplizierten Endaufbau zu ermöglichen. Aufgrund der Größe des zur Verfügung stehenden Leiterplattenfräzers wurden 4 Leiterplatten-Fragmente angefertigt, welche im Endaufbau mithilfe von Drahtbrücken verbunden wurden.

Die Leiterplatten-Fragmente wurden in der Software KiCad 9 entworfen, da diese kostenlos ist und eine einsteigerfreundliche Benutzeroberfläche bietet.

4.2.1 Reed-Schalter Matrix – Figuren-Registrierung

Das Prinzip hinter der Figuren-Registrierung steckt im Matrix-Multiplexing-Verfahren. Die Reed Schalter sind in einer 8x8 Multiplexing-Matrix miteinander verbunden (**Fehler! Verweisquelle konnte nicht gefunden werden.**), sodass unter jedem Feld ein Reed-Schalter liegt.

Jede Reihe und Spalte ist mit einem Pin des Raspberry Pi verbunden, sodass ausgehend vom Raspberry Pi in jede Spalte ein Signal in Form von Spannung gegeben wird. An jeden Pin ist ebenfalls ein Pulldown-Widerstand angeschlossen, um sicher zu gehen, dass keine falschen Signale durch Reststrom registriert werden.

Steht eine Schachfigur mit einem Magneten im Boden über einem Reed-Schalter, so schließt dieser sich und leitet das Signal die Reihe entlang an einen anderen Pin, mit welchem auf ein Signal geprüft wird.

Demnach lässt sich mit bekanntem Spannungseingang und ausgelesenem Spannungsausgang das Feld bestimmen, auf welchem sich ein Magnet befindet. Dieser Prozess erfolgt insgesamt 4-mal pro Sekunde für die gesamte Matrix.

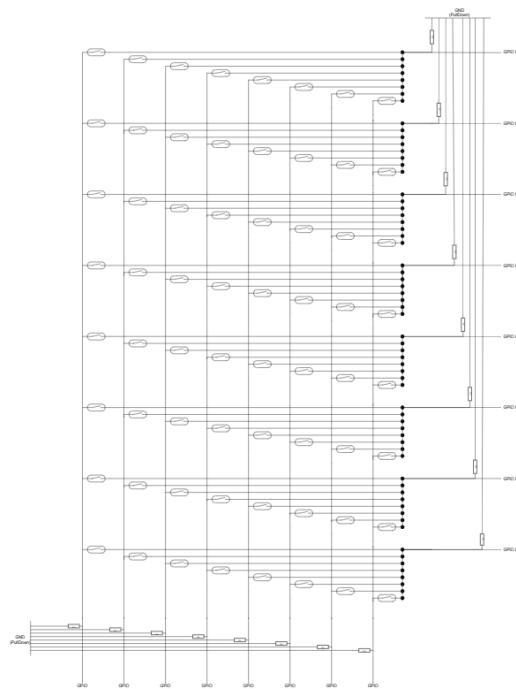


Abbildung 2: Schaltplan Reed-Schalter Matrix



4.2.2 LED-Aufbau und Interface

Bei den LEDs handelt es sich um adressierbare RGB-LEDs. Diese sind in einem eigenen Stromkreislauf unabhängig von der Reed-Schalter-Matrix an den Raspberry Pi geschalten. Dies bietet den Vorteil, dass im Falle einer defekten Komponente keine anderen Komponenten beeinträchtigt werden.

Die LEDs werden lediglich per IN- und OUTPUT-Pins in Reihe geschalten und mit Strom versorgt. Dabei muss der INPUT Pin der ersten LED an einen Pin des Raspberry Pi angeschlossen werden.



Abbildung 3: Schaltplan der LED-Matrix

Um die LEDs programmatisch ansteuern zu können wurde ebenfalls unter Verwendung der Neopixel¹ Library ein LED-Interface aufgesetzt, in dem relevante Funktionen für die Nutzung im Schachkontext bereitgestellt werden.

```
def init_chess_matrix(self):
    """Initialize a chessboard pattern on the LED matrix.

    This method sets up a chessboard pattern using two specified colors.

    Args:
        color1 (tuple): RGB color tuple for the first color (e.g., white).
        color2 (tuple): RGB color tuple for the second color (e.g., black).
    """
    color1 = (255, 255, 255)
    color2 = (0, 0, 255)

    for y in range(self.HEIGHT):
        for x in range(self.WIDTH):
            if (x + y) % 2 == 0:
                color = color1
            else:
                color = color2
            index = self._map_leds(x, y)
            self.pixels[index] = color
    self.pixels.show()
```

Abbildung 4: Funktion zum Initialisieren eines Schachbrettmusters

¹ [Neopixel Library](#)

4.3 Software

4.3.1 Backend-Software Architektur

Die Backend-Software wurde vollständig in Python entwickelt. Python wurde aufgrund seiner Vielseitigkeit, einfachen Syntax und der breiten Unterstützung für Hardware-Integration, vor allem der GPIO-Steuerung, ausgewählt. Die Hauptaufgaben der Software umfassen die Steuerung der Hardware, die Verwaltung des Spiels und die Bereitstellung einer Client-Server-Schnittstelle für die Web-App.

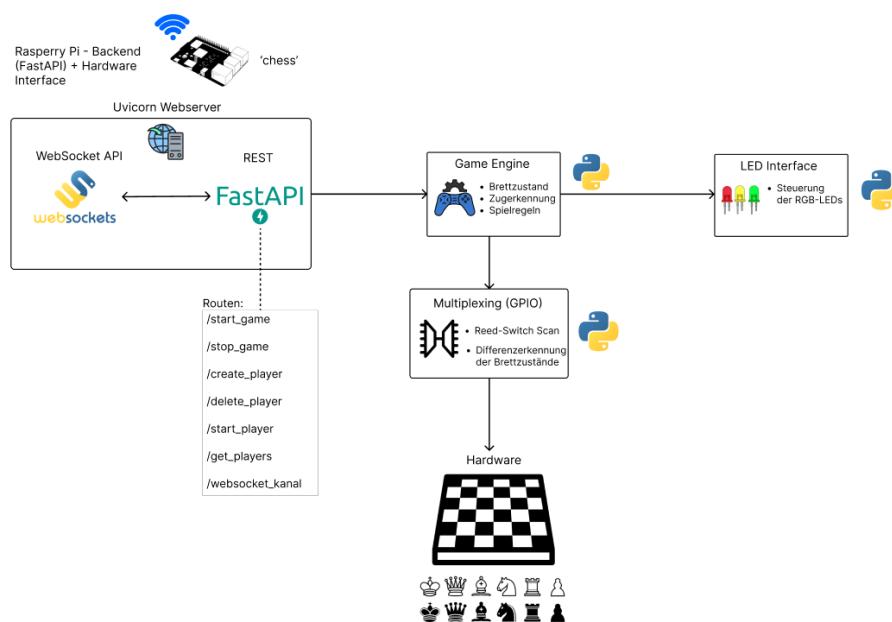


Abbildung 55: Backend-Softwarearchitektur

Die Softwarearchitektur des Backend umfasst vier Hauptmodule: API (Hauptmodul), Game Engine, Multiplexing und das LED-Schnittstellenmodul.

Das Hauptmodul ist zuständig für die Bereitstellung einer REST-API und Übertragung von Spielzuständen über die WebSocket-Schnittstelle an alle verbundenen Clients. Das Game-Engine-Modul ist für die Verwaltung der Spiellogik, einschließlich der Überprüfung von Spielzügen und Spielzuständen, für die Weiterverarbeitung der Daten des Multiplexing und zur Steuerung der LEDs. Das Multiplexing Modul ist für das Abtasten und Speichern der aktiven

Spielfelder zuständig. Die ausgelesenen Daten werden an das Game-Engine-Modul übergeben. Das LED-Interface-Modul wird für die Steuerung der RGB-LEDs, wie zum Beispiel für das Hervorheben von möglichen Spielzügen, verwendet.

4.3.2 Erkennung der Figuren und Auswertung der Rohdaten

Das in 4.2.1 beschriebene Multiplexing-Prinzip wurde folgendermaßen in Python implementiert:

```
active_squares = []

try:
    for row_index, row_pin in enumerate(reversed(self.row_pins)):
        GPIO.output(row_pin, GPIO.HIGH)

        for col_index, col_pin in enumerate(self.column_pins):
            if self.is_signal_stable(col_pin):
                active_squares.append(Square(col_index, row_index))
                self.logger.log_debounced_result(active_squares)

        GPIO.output(row_pin, GPIO.LOW)

return active_squares
```

Abbildung 6: Multiplexing mit Python

Aktive Spielfelder, werden in einer Liste von Objekten mit X- und Y Koordinaten, welche die Spalte und Reihe des aktiven Feldes angeben, gespeichert. Mithilfe der python-chess² Bibliothek ist es möglich die ausgelesenen Koordinaten den Spielfeldern zuzuordnen. In Kombination mit der Nutzung der Forsyth-Edwards-Notation³, kurz FEN, lassen sich Spielzustände speichern und sämtliche Informationen, in Bezug auf den FEN, wie zum Beispiel die möglichen Spielzüge für eine bestimmte Figur oder Spielzustände wie Schach oder Schachmatt, abrufen. Die gesamte Steuerung des Spiels verläuft über die python-chess-Bibliothek, es werden erkannte Spielzüge ausgeführt und jedes Mal ein neuer FEN generiert, welcher dann erneut analysiert wird.

² [python-chess: a chess library for Python – python-chess 1.11.2 documentation](#)

³ [Forsyth-Edwards-Notation – Wikipedia](#)

4.3.3 Client-Server Schnittstelle

Für die Implementierung der Client-Server-Schnittstelle wurde FastAPI⁴ und unicorn als Webserver verwendet, da es eine moderne, performante und flexible Lösung für Webanwendungen bietet. Die API wurde genutzt, um grundlegende Funktionen, wie die Verwaltung von Spielern und die Steuerung des Spiels bereitzustellen. So ermöglichen Endpunkte wie /create_player das Erstellen neuer Spieler, während /start_game das Starten eines neuen Spiels einleitet. Die REST-API war entscheidend für die Verwaltung von Daten und die Interaktion mit der Webanwendung. Zusätzlich wurde ein WebSocket⁵-Endpunkt implementiert, um Echtzeit-Updates des Spiels an alle verbundenen Clients gesendet. Über diesen Endpunkt werden Informationen wie die aktuelle FEN-Stellung, mögliche Züge und der zu ziehende Spieler im JSON-Format übertragen. Die Kombination aus FastAPI und unicorn als Webserver ermöglicht eine performante und skalierbare Architektur. unicorn, ein schneller ASGI-Server, unterstützte die asynchrone Verarbeitung und sorgt für geringe Latenzzeiten, was für die Echtzeit-Kommunikation im Schachspiel entscheidend ist. WebSockets ermöglichen eine bidirektionale, konstante Verbindung zwischen Server und Clients, wodurch Änderungen wie neue Züge sofort synchronisiert werden.

⁴ [FastAPI](#)

⁵ [WebSockets - FastAPI](#)

4.4 Website

Die Website stellt den Spielstand des physischen Schachbretts graphisch dar, ermöglicht Zuschauern einen Überblick über das laufende Spiel und bietet den Spielern eine Stoppuhr, die Spielernamen sowie Funktionen zum Beenden eines Spiels. Ziel war es, eine einfache Benutzer- und Zuschauererfahrung zu schaffen.

4.4.1 Aufbau und Inhalt



Abbildung 7: Website-Startmenü

Die Benutzeroberfläche wurde bewusst kompakt und übersichtlich gestaltet:

Schachbrett: Eine feste Komponente, die sich ab Spielstart aktualisiert, sobald neue Informationen per WebSocket empfangen werden.

Startmenü: Dient zur Konfiguration. Hier können Spieler hinzugefügt, bearbeitet, gelöscht und ausgewählt sowie die Spielzeit eingestellt werden.

Spielinformationen: Während des Spiels läuft die Stoppuhr des aktiven Spielers. Das Schachbrett wird nach jeder physischen Bewegung aktualisiert. Die Spieler können pausieren, aufgeben oder ein Remis anbieten.

Spielende: Nach dem Spielende wird der Gewinner im Startmenü angezeigt und ein neues Spiel kann gestartet werden.

Navigation: Der Wechsel zwischen Startmenü und Spielansicht erfolgt automatisch, sobald ein Spiel gestartet oder beendet wird. Zusätzlich erscheinen Pop-ups zur Konfiguration oder zur Annahme des Remis-Angebots.

Fehler und Statusanzeige: Die Website zeigt Statusmeldungen wie „Spieler aktualisiert“ und „Spieler gelöscht“ oder Fehler bei Kommunikationsproblemen mit der API oder dem WebSocket an.

Responsive Design: Das Layout wurde responsive gestaltet, jedoch mit Fokus auf größere Displays im Landscape-Modus.

4.4.2 Technische Umsetzung

Die Website ist dynamisch und wurde mit React und TypeScript umgesetzt. React ermöglicht einen klar strukturierten, komponentenbasierten Aufbau, der bei wiederverwendeten Elementen wie Schachbrett, Stoppuhr oder Startmenü Vorteile bietet. Änderungen innerhalb einzelner Bereiche lassen sich dadurch unabhängig voneinander entwickeln und warten.

Für Echtzeitaktualisierungen eignet sich React besonders gut, da durch Hooks⁶ wie useEffect nur relevante Komponenten neu gerendert werden, ohne die gesamte Seite neu zu laden. Dies ist zum Beispiel wichtig, wenn neue Informationen über das Schachbrett eintreffen, sodass nur die betroffene Komponente aktualisiert wird.

TypeScript sorgt durch statische Typisierung für weniger Laufzeitfehler und verbessert die Zusammenarbeit mit API und WebSocket. Besonders bei mehreren Modulen bleibt der Code übersichtlich und stabil.

4.4.2.1 Zustandsverwaltung

Zur Strukturierung der Anwendung wurden verschiedene Zustände in getrennte Provider, durch Nutzung von createContext, ausgelagert.

Die Provider sind aufgeteilt unter PlayerProvider, GameTimeProvider, NotificationProvider und GameInfoProvider. Durch diese Trennung und Auslagerung bleibt die Logik klar voneinander getrennt und die Komponenten erhalten nur die Daten, die sie tatsächlich benötigen. Zudem müssen Variablen

⁶ [React Documentation – Hooks Overview](#)

und Funktionen nicht untereinander weitergegeben werden, da der Provider die benötigten Daten und Funktionen sammelt und zur Verfügung stellt. Dies ermöglicht eine saubere Code-Struktur und eine einfache Wartbarkeit und Erweiterbarkeit. Der Zugriff erfolgt über eigene Hooks, wie zum Beispiel „`const [whitePlayer, blackPlayer, getPlayers] = usePlayer();`“, wodurch die Datenverwaltung übersichtlich und einheitlich bleibt. React-Hooks wie useMemo und useCallback werden verwendet, um unnötiges Re-Rendering zu vermeiden und nur relevante Komponenten bei Änderungen neu zu rendern. Somit bleibt die Website auch bei Echtzeitaktualisierungen performant.

4.4.2.2 Kommunikation mit API und WebSocket

Die Website kommuniziert sowohl mit einer API als auch mit einem WebSocket. Hierfür wurden ebenfalls getrennte Provider eingesetzt, wie der PlayerProvider für die Spieler-Verwaltung per API und der GameInfoProvider für die WebSocket-Verbindung und den Spielstatus. Die Logik ist bewusst aus UI-Komponenten ausgelagert, um eine saubere Trennung zwischen Darstellung und Datenverarbeitung zu gewährleisten. Die UI erhält nur vorbereitete Daten und aktualisiert daraufhin Elemente wie das Schachbrett oder Statusmeldungen.

5 Abschlussbericht, Eindrücke und Reflektion

Das Projekt Smart-Chessboard wurde an der Hausmesse von den Besuchern sehr positiv aufgenommen.

Rückblickend sind wir mit der Aufgabenteilung sowie dem gesamten Projektablauf sehr zufrieden. Das Projekt ist größtenteils funktionsfähig, bis auf die Figurenerkennung, da diese nicht vollständig funktioniert, was vermutlich auf mangelnde Qualität der Reed-Schalter zurückzuführen ist. Für zukünftige Projekte nehmen wir daher mit, bei zentralen Bauteilen auf qualitativ hochwertigere Komponenten zu setzen. Insgesamt war das Projekt eine wertvolle und lehrreiche Erfahrung. Sowohl das Projektmanagement als die Zusammenarbeit im Team konnten erfolgreich umgesetzt werden und das Endergebnis eignete sich gut für Präsentationszwecke.

6 Literaturverzeichnis

- 1 [Neopixel Library](#)
- 2 [python-chess: a chess library for Python – python-chess 1.11.2 documentation](#)
- 3 [Forsyth-Edwards-Notation – Wikipedia](#)
- 4 [FastAPI](#)
- 5 [WebSockets - FastAPI](#)
- 6 [React Documentation – Hooks Overview](#)
- 7 [Smart-Chessboard public repository](#)