```java
@Service
public class BotService {
    // Methoden zur Verwaltung von Bots
}
```

Mittelstufenprojekt
Discord-Bot-Framework

5. Juni 2024

Philipp Batelka, Jan Mahnken, Daniel Quellenberg,
Fabian Reichwald, Justus Sieweke, Christopher Spencer

Was ist BotHQ?

BotHQ ist ein modulares Framework zur Erstellung und Verwaltung von Discord-Bots. Es bietet eine benutzerfreundliche Weboberfläche und eine robuste API zur Integration und Verwaltung von Plugins.

Durch die Verwendung von Plugins können Bots automatisch verschiedene Aufgaben ausführen, wie das Senden von Nachrichten, die Moderation von Benutzern, das Hosten von Spielen und sogar das Abspielen von Musik.

Projektübersicht

Ziel des Projekts:

Die Entwicklung eines modularen Discord-Bot-Frameworks, das als Cloud-Service angeboten wird und eine einfache Verwaltung von Plugins ermöglicht.

Zielgruppe:

- Gamer und Gaming-Communities
- Entwickler und Technikbegeisterte
- Unternehmen und Organisationen
- Content Creator und Streamer

Nutzen:

BotHQ erleichtert die Erstellung, Verwaltung und Konfiguration von Discord-Bots, indem es eine benutzerfreundliche Oberfläche und eine flexible API bietet.

Technologie-Stack

Verwendete Technologien:

- **Backend:** Spring Boot
- **Frontend:** Angular
- **Datenbank:** PostgreSQL
- **API:** REST
- **Bot-Integration:** JDA (Java Discord API)

Warum diese Technologien?

- **Spring Boot:** Framework für robuste Backend-Entwicklung
- **Angular:** Frontend-Framework für dynamische Webanwendungen
- **PostgreSQL:** Skalierbare Datenbanklösung
- **JDA:** Bibliothek für die Integration von Discord-Bots in Java

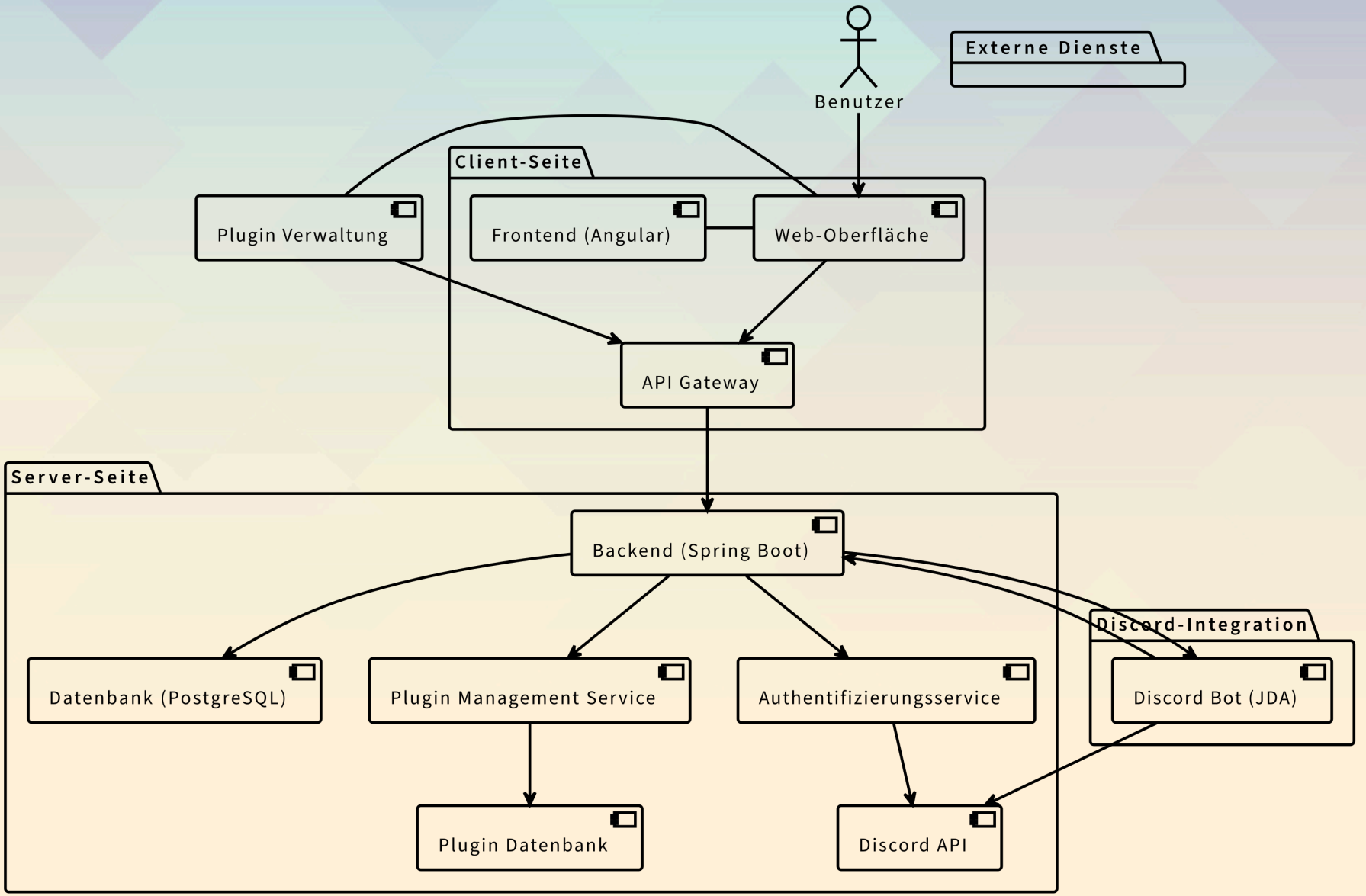
Architektur des Systems

Komponenten:

- **Core:** Grundlegende Logik und Datenverarbeitung
- **Lib:** Wiederverwendbare Bibliotheken und Utilities
- **Frontend:** Benutzeroberfläche zur Verwaltung von Bots und Plugins
- **Discord-Plugins:** Individuelle Plugins für spezifische Bot-Funktionen

Interaktionen:

- **API:** Kommuniziert zwischen Frontend und Backend
- **Datenbank:** Speichert Nutzerdaten, Bot-Konfigurationen und Plugin-Einstellungen


Benutzer
Externe Dienste
Client-Seite
Plugin Verwaltung
Frontend (Angular)
Web-Oberfläche
API Gateway
Server-Seite
Backend (Spring Boot)
Datenbank (PostgreSQL)
Plugin Management Service
Authentifizierungsservice
Discord-Integration
Discord Bot (JDA)
Plugin Datenbank
Discord API

Funktionen und Features

Hauptfunktionen:

- **Plugin-Verwaltung:** Einfache Aktivierung, Deaktivierung und Konfiguration von Plugins
- **Benutzerfreundliche Weboberfläche:** Intuitive Verwaltung über das Frontend
- **Discord-Bot-Integration:** Nahtlose Einbindung in Discord-Server

Anwendungsfälle:

- **Moderation:** Automatische Verwaltung und Moderation von Discord-Servern
- **Engagement:** Interaktive Features für Community-Engagement
- **Automatisierung:** Automatisierung wiederkehrender Aufgaben und Prozesse

Screenshots

- [Hier Screenshot 1 einfügen]
- [Hier Screenshot 2 einfügen]

Entwicklungsprozess

Vorgehensmodell:
Das Projekt wurde nach dem agilen Scrum-Framework durchgeführt, angepasst an die spezifischen Bedürfnisse des Teams.

Sprints und Meilensteine:

- **Sprint 1:** Anforderungsanalyse und Design
- **Sprint 2:** Implementierung der Kernfunktionen
- **Sprint 3:** Testen und Feinschliff

Herausforderungen und Lösungen:

- **Herausforderung:** Datenbankanpassungen während der Entwicklung
- **Lösung:** Flexibles Datenbankschema und kontinuierliche Integration

Technische Details

Wichtige Klassen und Module:

- **Core:** `BotService`, `UserService`
- **Lib:** `UtilityFunctions`, `DatabaseHelper`
- **Frontend:** `DashboardComponent`, `PluginManagerComponent`
- **Discord-Plugins:** `WelcomePlugin`, `ModerationPlugin`

Beispiel: `BotService`

```
@Service
public class BotService {
    // Methoden zur Verwaltung von Bots
}
```

Codebeispiele

Backend (Spring Boot):

```java
@RestController
@RequestMapping("/api/bots")
public class BotController {
    @Autowired
    private BotService botService;

    @PostMapping
    public ResponseEntity<Bot> createBot(@RequestBody Bot bot) {
        return ResponseEntity.ok(botService.createBot(bot));
    }
}
```

```typescript
@Component({
  selector: "app-plugin-manager",
  templateUrl: "./plugin-manager.component.html",
})
export class PluginManagerComponent {
  // Methoden zur Verwaltung von Plugins
}
```

Frontend (Angular):

```
@Component({
  selector: "app-dashboard",
  templateUrl: "./dashboard.component.html",
})
export class DashboardComponent {
  // Logik für das Dashboard
}
```

Tests und Qualitätssicherung

Teststrategie:

- **Unit Tests:** Überprüfung einzelner Methoden und Funktionen (JUnit, Mockito)
- **Integrationstests:** Überprüfung des Zusammenspiels verschiedener Komponenten
- **Systemtests:** Gesamtsystemtests unter realistischen Bedingungen

Testergebnisse:

- **Unit Tests:** 95% Abdeckung
- **Integrationstests:** Erfolgreiche Verbindungen zwischen Komponenten
- **Systemtests:** Alle Hauptfunktionen arbeiten wie erwartet

Qualitätssicherung:

- Regelmäßige Code Reviews
- Kontinuierliche Integration und Deployment (CI/CD)

Nutzerdokumentation

Benutzerhandbuch:

- Schritt-für-Schritt-Anleitung zur Installation und Konfiguration
- FAQ und Problemlösungen

Anleitungen:

- **Installation:** Einfache Schritte zur Installation des Bots
- **Konfiguration:** Anpassen der Bot-Einstellungen über die Weboberfläche

Beispiele:

- **Willkommensnachricht:** Konfigurieren einer Begrüßungsnachricht für neue Mitglieder
- **Moderation:** Automatisiertes Kicken/Bannen von Mitgliedern

Fazit und Ausblick

Zusammenfassung:

- Erfolgreiche Entwicklung eines modularen Discord-Bot-Frameworks
- Bereitstellung als Cloud-Service
- Flexible und benutzerfreundliche Verwaltung von Plugins

Erkenntnisse und Lessons Learned:

- Wichtige Rolle der agilen Methoden in der Softwareentwicklung
- Bedeutung der Flexibilität bei der Anpassung an neue Anforderungen
- Erfolgreiche Teamarbeit und Kommunikation als Schlüsselfaktoren

Ausblick:

- Erweiterung der Plugin-Bibliothek
- Verbesserung der Benutzeroberfläche
- Integration zusätzlicher Funktionen und Dienste

Hauptmerkmale

- **Automatisierte Einrichtung**: BotHQ automatisiert den Einrichtungsprozess für Discord-Bots.
- **Benutzerfreundliche Oberfläche**: Eine intuitive Weboberfläche zur effizienten Verwaltung von Bots und Plugins.
- **Modulares Framework**: Erweiterbar durch Hinzufügen neuer Plugins.
- **Sicher und Zuverlässig**: Fokus auf Sicherheit und Zuverlässigkeit.
- **Cloud-basierter Service**: Hosten Sie Ihre Bots auf unserer zuverlässigen Cloud-Plattform.

Warum BotHQ?

- **Zeit- und Müheersparnis**: Automatisierung des Einrichtungs- und Verwaltungsprozesses für Discord-Bots.
- **Flexibel und anpassbar**: Modulares Framework für endlose Anpassungsmöglichkeiten.
- **Für verschiedene Nutzergruppen geeignet**: Gamer, Entwickler, Unternehmen, Content Creator.
- **Aktive Community und Support**: Lebendige Community und umfassender Support für BotHQ.

Einleitung

Ein Discord-Bot-Framework ist eine Sammlung von Tools und Bibliotheken, die Entwicklern helfen, Bots für die Discord-Plattform zu erstellen. Diese Bots können verschiedene Aufgaben automatisieren, wie z.B. Nachrichten senden, Benutzer moderieren, Spiele hosten oder sogar Musik abspielen. Ein Framework bietet oft eine strukturierte und vereinfachte Art und Weise, diese Bots zu erstellen, indem es die Interaktion mit der Discord API abstrahiert.

Screen portrayals

Year	Title	Actor
1970	Jonathan	Paul Albert Krumm
1995	Monster Mash	Anthony Crivello
2004	Blade: Trinity	Dominic Purcell
2008	Supernatural	Todd Stashwick
2020	Dracula	Claes Bang

Words from the Source

> There are darknesses in life and there are lights, and you are one of the lights, the light of all lights.
>
> -- Bram Stoker, Dracula

Bats - Implementation

```
class Bat:
    def __init__(name:str, age:int):
        self.__name = name
        self.__age = age
    @property
    def name(self):
        return self.__name
    @property
    def age(self):
        return self.__age
    @property
    def speed(self):
        return 10 - self.age
```

Projektbeschreibung

Projektidee und Zielsetzung

- Ein modularer, cloudbasierter Dienst zur Verwaltung von Discord-Bots

Anforderungen

- Cloudbasiert
- Einfachache Nutzung für erfahrene/unerfahrene Nutzer
- Framework zur simplen entwicklung neuer Bots

Planungsphase

Projektplanung

- Jira
- Scrum
- 3 Sprints a ~ 1 Monat

Ressourcenplanung

- Teamzusammensetzung und Verantwortlichkeiten
- Benötigte Hardware und Software

Umsetzung

Entwicklung des Frameworks

- Webapplication auf Sgring-Boot Basis
- Spring Boot, Angular, REST API, PostgreSQL

Hauptfunktionen und Features

- Modulares Plugin-System
- Benutzerfreundliche Weboberfläche
- Integration mit der Discord-API
- Beispiel-Plugins (Willkommensnachricht, Rollenverwaltung, etc.)

Herausforderungen und Lösungen

Technische Herausforderungen

- Anpassungen der Datenbankstruktur
- Integration der verschiedenen Technologien

Rechtliche Herausforderungen

- Datenschutzbestimmungen und Discord-ToS

Lösungsansätze

- Wie wurden die Herausforderungen gemeistert?

Projektergebnisse

Erreichte Ziele

- Übersicht der implementierten Funktionen und Features

Nicht erreichte Ziele

- Was konnte nicht umgesetzt werden und warum?

Änderungen zur ursprünglichen Planung

- Anpassungen während der Projektlaufzeit und deren Gründe

Wirtschaftliche Betrachtung

Marktuntersuchung

- Zielgruppenanalyse und Marktpotenzial

Kostenplanung

- Personalkosten und Sachmittelkosten

Wirtschaftlichkeitsberechnung

- Gewinnschwellenberechnung und Amortisationsrechnung

Fazit

Erfahrungen und Erkenntnisse

- Persönlicher Gewinn und technisches Know-how

Ausblick

- Möglichkeiten zur Weiterentwicklung des Projekts
- Zukünftige Herausforderungen und Chancen

Fragen und Diskussion

- Einladung zur Fragerunde
- Offene Diskussion