

Projekt zum Thema:  
**Adaptive Cruise Control System für Fahrzeuge**

vorgelegt von: Eduard Syrov

Dozent: Prof. Dr. Roland Petrasch

Abgabedatum: 20.03.2025

# Inhaltsverzeichnis

<b>1. Einführung.....</b>	<b>3</b>
<b>2. Datenmodell und Use-Cases/User Stories.....</b>	<b>3</b>
<b>2.1 Use Cases.....</b>	<b>4</b>
<b>3. Backend.....</b>	<b>5</b>
3.1 Backend-Komponenten und Schnittstellen.....	5
3.2 Backend-Design-Patterns.....	6
3.3 Spezielle Themen (Backend).....	8
3.4 Test des Backends.....	9
Unit-Tests für Sensors.....	9
Komponenten-Tests:.....	10
Integrationstests:.....	11
<b>4. Frontend.....</b>	<b>12</b>
<b>4.1 Frontend: Layout.....</b>	<b>12</b>
4.2 Frontend: Logik.....	14
4.3 Spezielle Themen (Frontend).....	16
Spezielles Thema: WebSockets.....	16
4.4 Frontend: Test mit Cypress.....	17
<b>5. Organisatorisches.....</b>	<b>19</b>
5.1 Projektmanagement.....	19
5.2 Versionierung.....	21
<b>6. Referenzen.....</b>	<b>22</b>

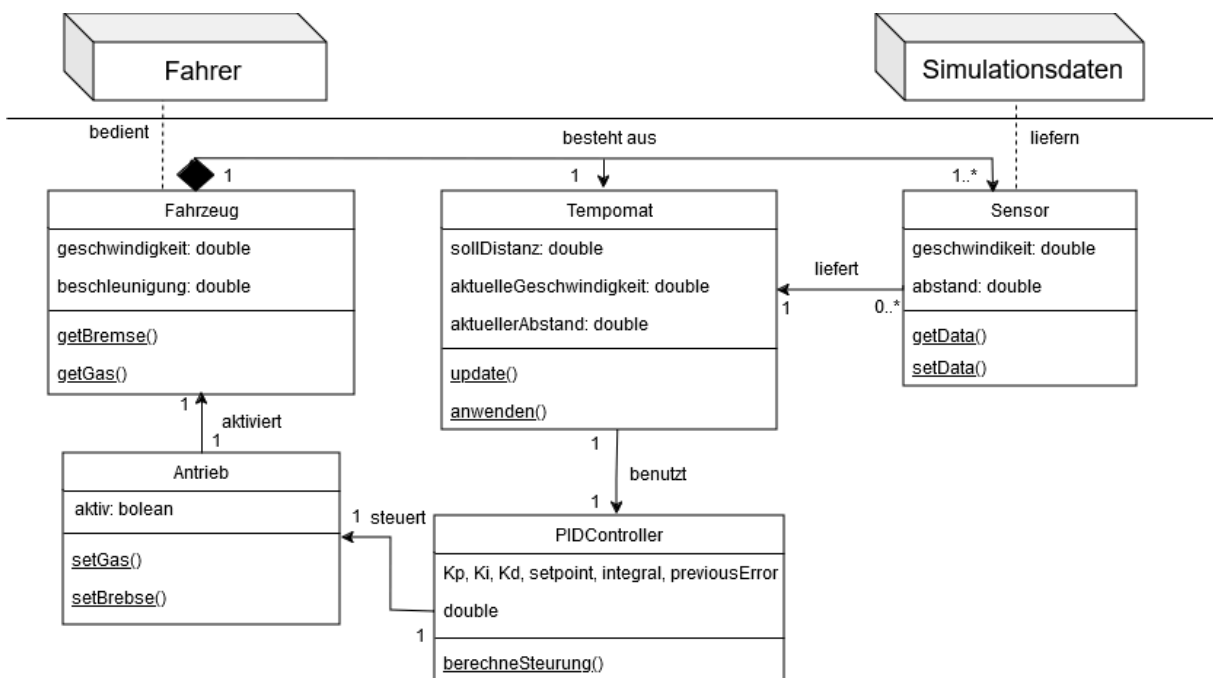
# 1. Einführung

Das Ziel folgender Arbeit ist es, ein simuliertes Adaptive Cruise Control System für Fahrzeuge zu entwickeln. Es soll in der Lage sein, anhand der Geschwindigkeit des vorausfahrenden Fahrzeugs und dem Abstand zu diesem Fahrzeug seine Geschwindigkeit sowie dessen Abstand konstant zu halten.

Hierfür kann ein PID (Proportional, Integral, Derivative) Controller verwendet werden, der als Prozessvariable die Geschwindigkeit des vorausfahrenden Fahrzeugs oder die Entfernung zu ihm hat. Weiterhin gibt es einen Setpoint, der den gewünschten Abstand definiert. Als Steuer- oder Kontrollvariablen sind das Gaspedal und die Bremse zu sehen.

Die Simulationsdaten werden über eine Excel-Tabelle importiert. Probleme können bei plötzlich starkem Bremsen des Vorgängers und nicht-linearen Einflüssen entstehen, z.B. Reibung, Luftwiderstand, etc.) entstehen, die mit Feedforward Control, Model Predictive Control (MPC) o.ä adressiert werden können.

## 2. Datenmodell und Use-Cases/User Stories



## Beschreibung der Entitäten

1. **Fahrzeug**: repräsentiert das Auto, das mit dem adaptiven Tempomat ausgestattet ist. Es hat Attribute wie die aktuelle Geschwindigkeit des Fahrzeugs und aktuelle Beschleunigung.
2. **Antrieb**: repräsentiert das Antriebssystem des Fahrzeugs, das die Steuerbefehle (Gas und Bremse) umsetzt.
3. **Tempomat**: ist das adaptive Tempomat-System, das die Geschwindigkeit und den Abstand des Fahrzeugs regelt. Methoden aktualisieren und wenden die Steuerlogik an.
4. **PIDController**: ist ein (Proportional-Integral-Derivative) Entity, das die Steuerung des Abstands und der Geschwindigkeit optimiert. Er hat Attribute wie  $K_p$ ,  $K_i$ ,  $K_d$  (Reglerparameter), setpoint (Zielwert, z. B. Sollabstand), integral (Summe der Fehler) und previousError (vorheriger Fehler). Die Methode berechneSteuerung() berechnet die Steuergröße basierend auf dem aktuellen Fehler.
5. **Sensor**: ist ein Gerät, das die Geschwindigkeit und den Abstand zum vorausfahrenden Fahrzeug misst.

## 2.1 Use Cases

**Ziel:** Das Fahrzeug fährt mit angepasster Geschwindigkeit im sicheren Abstand zum vorausfahrenden Fahrzeug.

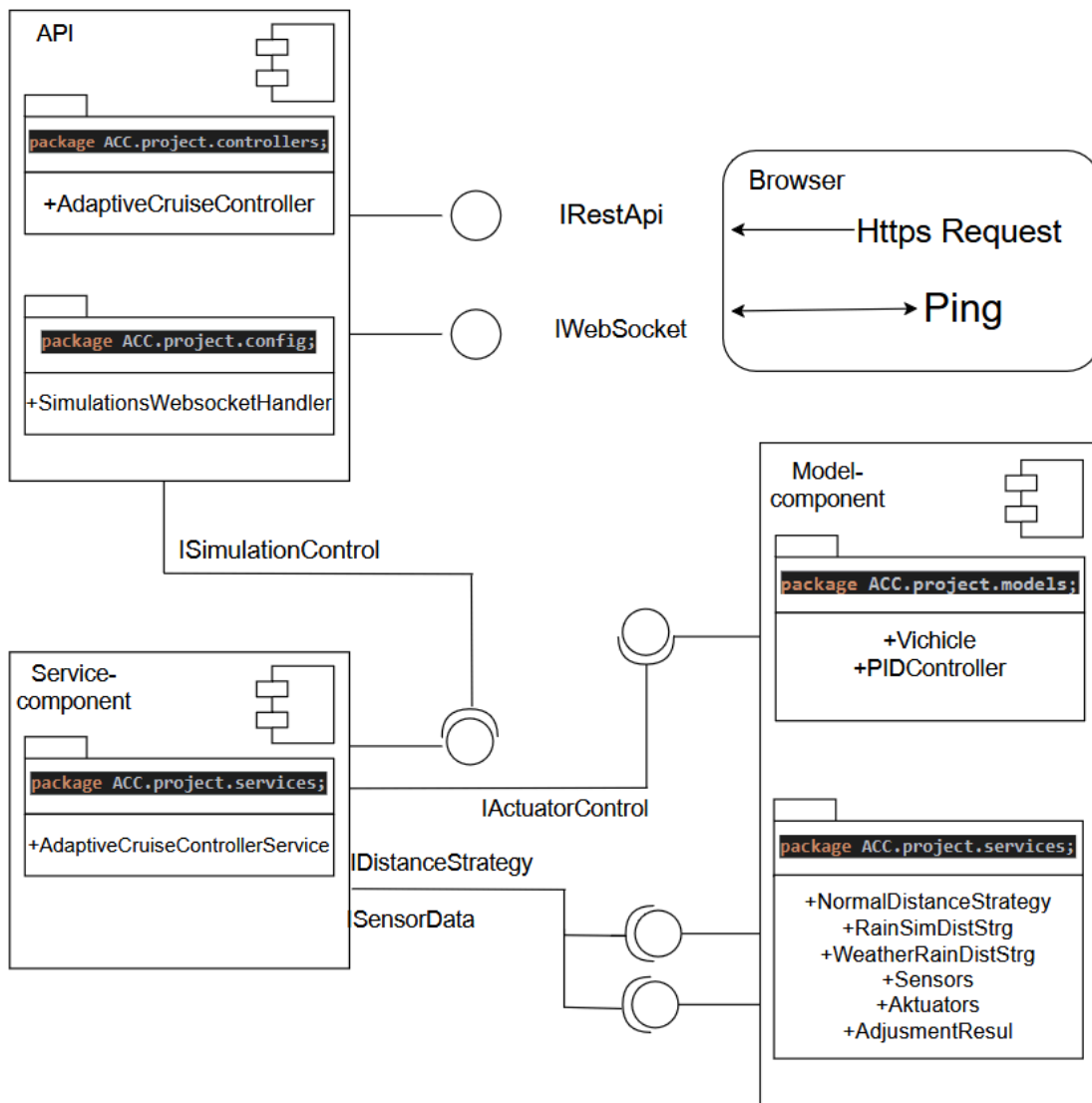
1. Die Sensoren liefern den Abstand und die Geschwindigkeitsdaten zum vorausfahrenden Fahrzeug.
2. Der PID-Controller berechnet das Steuersignal basierend auf dem Sollabstand.
3. Das System sendet Steuerbefehle an den Antrieb, um Gas oder Bremse anzuwenden.
4. Das Fahrzeug passt seine Geschwindigkeit an und hält den gewünschten Abstand ein.

### User Stories/Frontend

1. Ich will eine Excel Tabelle mit bestimmten Werten hochladen.
2. Ich will Werte aus einer Excel Tabelle durchgehen.
3. Ich will eine Simulation mit Standardwerten oder aus der Excel-Datei ausführen lassen.
4. Ich will die Wetterbedingungen, wie z.B. Regen oder starkem Bremsen simulieren.
5. Ich will sehen, wie sich die Fahrzeuge auf der Straße verhalten.

## 3. Backend

### 3.1 Backend-Komponenten und Schnittstellen



**API:** Diese Komponente bildet die Schnittstelle zum Frontend. REST-Controller verarbeiten HTTP-Anfragen (z. B. Start/Stopp der Simulation), während der WebSocket-Handler (SimulationWebSocketHandler) Echtzeitdaten (z. B. Simulationsdaten wie egoSpeed, distance) an das Frontend sendet. Die API-Komponente ist der Einstiegspunkt und delegiert Anfragen an die Services.

**Services:** Hier liegt die Geschäftslogik, z. B. die Steuerung des adaptiven Tempomaten (AdaptiveCruiseControlService). Sie orchestriert die Interaktion zwischen Fahrzeugdaten (egoVehicle), Sensoren (sensors), Aktoren (actuators) und dem PID-Regler. Die Services-Komponente ist das Herzstück der Logik und koordiniert Datenflüsse.

**Modell:** Die Datenmodelle sind einfache POJOs (Plain Old Java Objects) wie Vehicle oder PIDController. Sie definieren die Struktur der Daten, die zwischen den Komponenten ausgetauscht werden, und sind die Grundlage für die Kommunikation (z. B. JSON über WebSocket).

## 3.2 Backend-Design-Patterns

### Problemstellung und Motivation

In der ACC-Simulation müssen unterschiedliche Abstandsregelungen je nach Fahrsituation zur Anwendung kommen: ein normaler Abstand von 6–8 Metern, ein simulierter Regenabstand von 10 Metern ( $\pm 1$  Meter) und ein wetterabhängiger Regenabstand von 15–18 Metern. Diese Vorgaben könnten mit einer großen if-else-Struktur in der Steuerlogik implementiert werden, was jedoch die Wartbarkeit und Erweiterbarkeit beeinträchtigt. Bei neuen Anforderungen (z. B. Schnee mit einem anderen Abstand) müsste die Logik jedes Mal angepasst werden, was den Code komplex und fehleranfällig macht. Die Motivation war, eine flexible Lösung zu schaffen, die verschiedene Abstandsregelungen zur Laufzeit austauschbar macht und die Steuerlogik von den spezifischen Algorithmen entkoppelt. Das Strategie-Pattern wurde gewählt, um diese Flexibilität zu gewährleisten und neue Strategien einfach hinzufügen zu können.

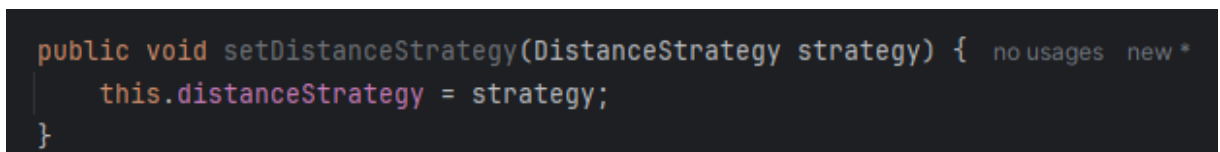
### Beschreibung des Strategie-Patterns

Das Strategie-Pattern ermöglicht die Definition einer Familie von Algorithmen, die in separaten Klassen gekapselt und zur Laufzeit ausgetauscht werden können. Im Projekt wurde eine Schnittstelle **DistanceStrategy** definiert, die die Methode *adjustDistance* vorschreibt.



```
1 package ACC.project.services;
2
3 import ACC.project.models.Vehicle;
4
5 public interface DistanceStrategy { 5 usages 3 implementations
6     void adjustDistance(Vehicle egoVehicle, Sensors sensors, Actuators actuators, float deltaTime);
7 }
```

Konkrete Strategien wie **NormalDistanceStrategy** (6–8 Meter) und **RainSimulationDistanceStrategy** (10 Meter) implementieren diese Schnittstelle mit spezifischen Regelungen. Die Klasse **AdaptiveCruiseControlService** dient als Kontext und

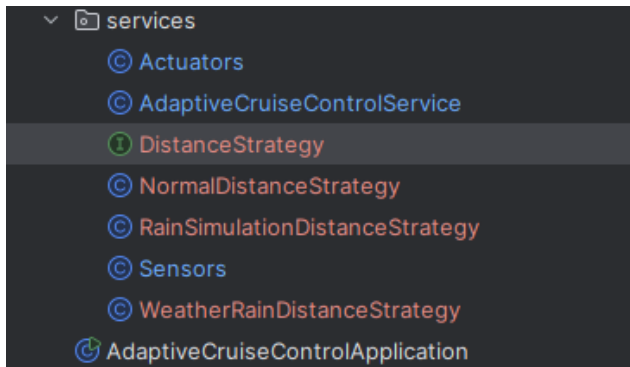


```
public void setDistanceStrategy(DistanceStrategy strategy) { no usages new *
    this.distanceStrategy = strategy;
}
```

enthält eine Instanzvariable *distanceStrategy*, die über *setDistanceStrategy* gewechselt wird. Die Methode *adjustSpeedContinuously* delegiert die Abstandsregelung an die aktuelle Strategie. Dies ermöglicht eine dynamische Anpassung des Verhaltens, z. B. durch Aufruf von *setDistanceStrategy(new RainSimulationDistanceStrategy())* im *toggleRain*-Handler.

## Verweis auf den Code

Die Implementierung findet sich in **AdaptiveCruiseControlService.java**. Die Schnittstelle **DistanceStrategy** und die Klassen **NormalDistanceStrategy** sowie



**RainSimulationDistanceStrategy** sind im gleichen Package definiert. Der Wechsel der Strategie erfolgt in **toggleRain**, wo bei **isRainSimulation = true** die **RainSimulationDistanceStrategy** gesetzt wird. Die Methode *adjustSpeedContinuously* ruft dann *distanceStrategy.adjustDistance(...)* auf, um die jeweilige Regelung auszuführen.

```
private void adjustSpeedContinuously(float deltaTime) { 1 usage new *
    if (sensors.isBraking()) {
        float distance = sensors.getDistanceToVehicle();
        float egoSpeed = egoVehicle.getSpeed();
        float leadVehicleSpeed = sensors.getSpeedOfLeadVehicle();
        float brakeFactor = Math.min(40.0f, (6.0f - distance) * 50.0f);
        if (egoSpeed > leadVehicleSpeed) {
            float brake = brakeFactor * deltaTime;
            egoVehicle.brake(brake);
            actuators.applyBrakes(brakeFactor);
        }
    } else {
        // Delegiere an die aktuelle Strategie
        distanceStrategy.adjustDistance(egoVehicle, sensors, actuators, deltaTime);
    }
}
```

## 3.3 Spezielle Themen (Backend)

### Nebenläufige Programmierung mit Threads und Synchronisation

#### Problemstellung und Motivation

In der ACC-Simulation sollte die Steuerlogik (runControlLoop) kontinuierlich in Echtzeit ausgeführt werden, anstatt nur bei expliziten API-Aufrufen. Dies simuliert realistischer ein Fahrzeugsystem, bei dem Sensoren und Aktoren permanent arbeiten. Eine sequentielle Ausführung wäre ineffizient und würde die Reaktionszeit verzögern. Zudem könnten gleichzeitige Zustandsänderungen (z. B. durch toggleRain oder toggleWeather) während der Simulation zu Race Conditions führen, da mehrere Methoden auf gemeinsame Ressourcen (z. B. distanceStrategy, egoVehicle) zugreifen. Die Motivation war, einen dedizierten Thread für die Simulation einzuführen und thread-sichere Zugriffe mit Synchronisation zu gewährleisten, um Konsistenz und Stabilität zu sichern.

#### Beschreibung

Nebenläufigkeit wurde durch einen separaten Thread implementiert, der runControlLoop in einem 100-ms-Intervall ausführt, solange isRunning = true. Der Thread wird in startSimulation gestartet und in stopSimulation beendet, wobei join() sicherstellt, dass er vollständig stoppt. Für die Synchronisation wurde ein ReentrantLock verwendet, um kritische Abschnitte (z. B. Zustandsänderungen in toggleRain, setDistanceStrategy) zu schützen. Dies verhindert Race Conditions, indem nur ein Thread gleichzeitig auf gemeinsame Ressourcen zugreifen kann. Theoretisch basiert dies auf der Notwendigkeit, Deadlocks zu vermeiden (durch konsistente Lock-Reihenfolge) und Datenintegrität zu wahren.

#### Verweis auf den Code

Die Implementierung befindet sich in **AdaptiveCruiseControlService.java**. Der Simulationsthread wird in startSimulation definiert, mit lock.lock() in runControlLoop und anderen Methoden wie toggleRain und toggleWeather. Der ReentrantLock (lock) schützt alle Zugriffe auf distanceStrategy und andere Zustandsvariablen.



```

public class AdaptiveCruiseControlService {

    public void startSimulation() { 1 usage new *
        if (!isRunning) {
            isRunning = true;
            isAdjusting = true;
            simulationThread = new Thread(() -> {
                while (isRunning) {
                    lock.lock();
                    try {
                        runControlLoop( deltaTime: 0.1f); // 100ms Intervall
                    } finally {
                        lock.unlock();
                    }
                    try {
                        Thread.sleep( millis: 100); // Simuliert Echtzeit
                    } catch (InterruptedException e) {
                        Thread.currentThread().interrupt();
                        break;
                    }
                }
            });
            simulationThread.start();
        }
    }
}

```

## 3.4 Test des Backends

### Unit-Tests für Sensors

```

19  @Test new *
20  void testSetLeadVehicleSpeed() {
21      sensors.setLeadVehicleSpeed(50.0f);
22      assertEquals( expected: 50.0f, sensors.getSpeedOfLeadVehicle(), delta: 0.01f);
23  }
24
25  @Test new *
26  void testSetDistance() {...}
30
31  @Test new *
32  void testUpdateSimulationSpeed() {...}
41
42  @Test new *
43  void testUpdateSimulationDistance() {...}
54
55  @Test new *
56  void testReset() {...}

```

Beispiel: testSetLeadVehicleSpeed() prüft, ob setLeadVehicleSpeed den Wert korrekt setzt.

## Komponenten-Tests:

Testen die Zusammenarbeit innerhalb einer Klasse (z. B. startSimulation in AdaptiveCruiseControlService).

```
31      @Test new *
32      void testStartSimulationSetsValues() {
33          service.startSimulation( websocketHandler: null, leadSpeed: 50.0f, distance: 15.0f, egoSpeed: 40.0f);
34          assertEquals( expected: 50.0f, sensors.getSpeedOfLeadVehicle(), delta: 0.01f);
35          assertEquals( expected: 15.0f, sensors.getDistanceToVehicle(), delta: 0.01f);
36          assertEquals( expected: 40.0f, egoVehicle.getSpeed(), delta: 0.01f);
37      }
38
39      @Test new *
40      void testStopSimulationResetsValues() throws InterruptedException {...}
41
42      @Test new *
43      void testStartAdjustingReducesSpeed() throws InterruptedException {...}
44
45      @Test new *
46      void testStartAdjustingIncreasesSpeed() throws InterruptedException {...}
```

Ergebnis:

```
✖ Tests failed: 1 of 1 test - 382 ms

Simulation gestartet mit leadSpeed=50.0, distance=15.0, egoSpeed=40.0

Expected :15.0
Actual   :15.555555
<Click to see difference>

> org.opentest4j.AssertionFailedError: expected: <15.0> but was: <15.555555>
```

**Problem:** Der Test erwartet 15.0f, aber der tatsächliche Wert ist 15.555555. Der Unterschied könnte an der Zufallskomponente in leadVehicleSpeed liegen ( $\text{Math.random()} * 2 - 1$ ), die die Geschwindigkeit leicht verändert, bevor der Abstand berechnet wird.

## Integrationstests:

**/run** Endpunkt startet Simulation und Prüft, ob der /run-Endpunkt die Simulation mit den übergebenen Werten startet.

```
23     @Autowired 3 usages
24     private MockMvc mockMvc;
25
26     @Test new *
27     void testRunSimulation() throws Exception {
28         SimulationData input = new SimulationData();
29         input.setLeadSpeed(50.0f);
30         input.setDistance(15.0f);
31         input.setEgoSpeed(40.0f);
32         mockMvc.perform(post(uriTemplate: "/run") // Pfad angepasst
33             .contentType(MediaType.APPLICATION_JSON)
34             .content(new ObjectMapper().writeValueAsString(input)))
35             .andExpect(status().isOk())
36             .andExpect(jsonPath(expression: "$.LeadSpeed").value( expectedValue: 50.0));
37     }
```

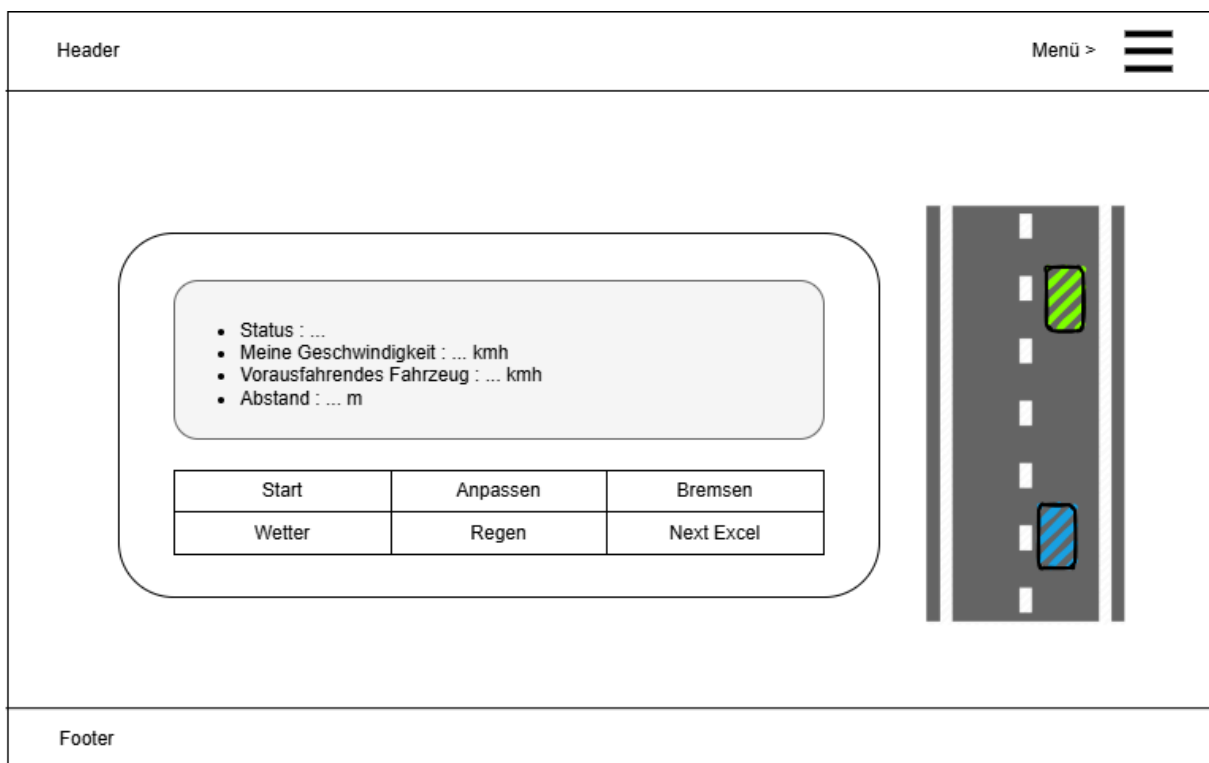
## Ergebnis:

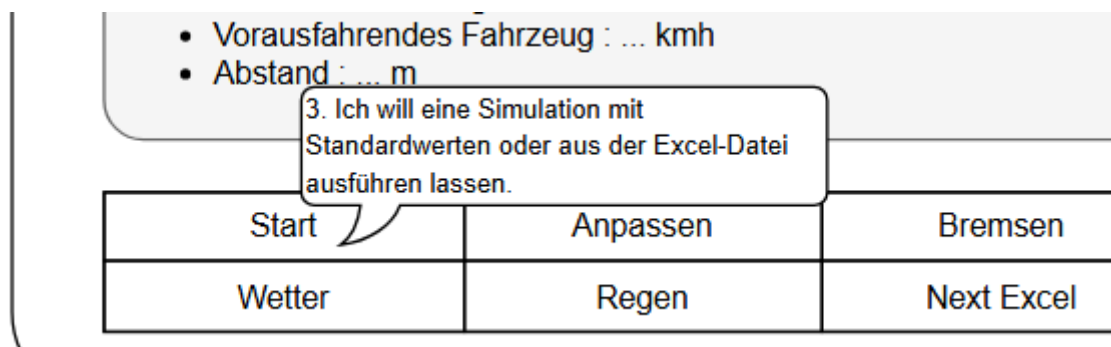
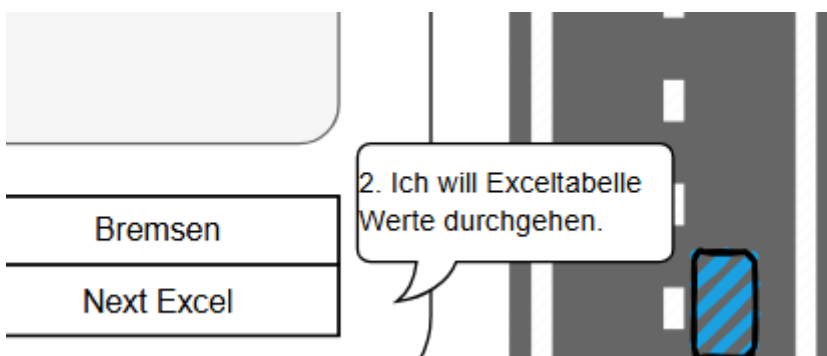
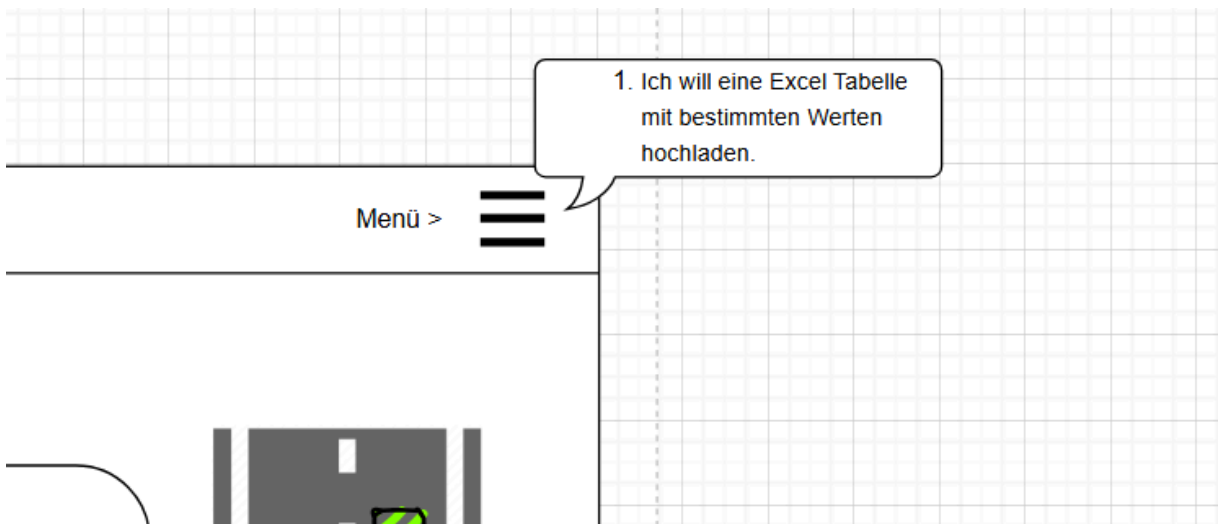
```
✓ Tests passed: 1 of 1 test - 935 ms
2025-03-25T13:16:22.081+01:00 INFO 12632 --- [AdaptiveCruiseControlSystem] [ Test worker] AccControlTest : S
WARNING: A Java agent has been loaded dynamically (C:\Users\Eddy\gradle\caches\modules-2\files-2.1\net.bytebuddy\byte-buddy-agent\1.14.19
WARNING: If a serviceability tool is in use, please run with -XX:+EnableDynamicAgentLoading to hide this warning
WARNING: If a serviceability tool is not in use, please run with -Djdk.instrument.traceUsage for more information
WARNING: Dynamic loading of agents will be disallowed by default in a future release
Simulation gestartet mit leadSpeed=50.0, distance=15.0, egoSpeed=40.0
Keine WebSocket-Clients verbunden
Java HotSpot(TM) 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended
```

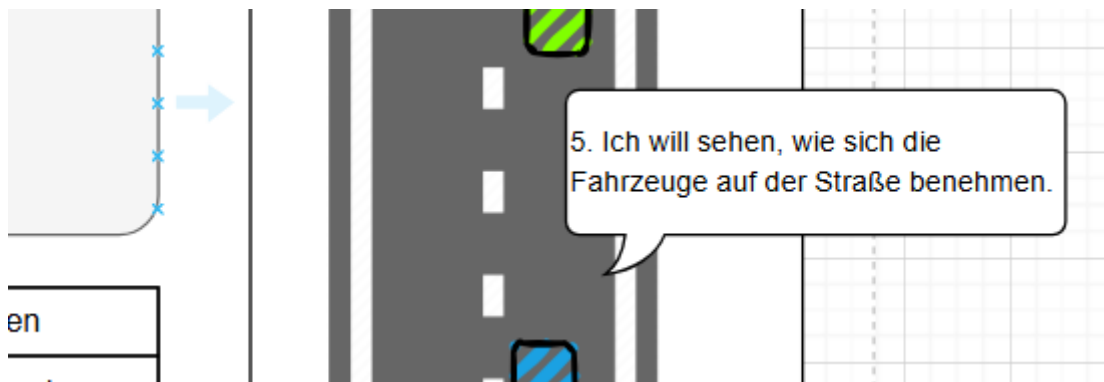
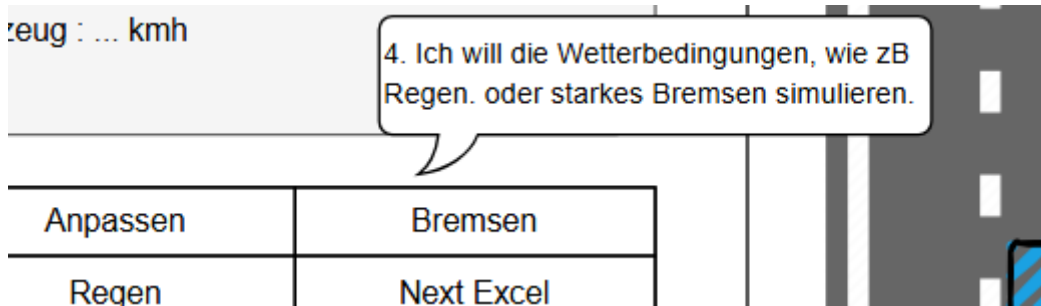
Test passed, obwohl mehrere Warnungen vorliegen.

## 4. Frontend

### 4.1 Frontend: Layout







## 4.2 Frontend: Logik

### 1. Services (API Clients)

**AccService:** Dient als API-Client, um mit dem Backend über HTTP zu kommunizieren.

**ExcelDataService:** Verwaltet Excel-Daten (z. B. Zeilen mit leadSpeed, egoSpeed, distance) und stellt sie der Anwendung bereit.

**WebSocketService:** Stellt eine WebSocket-Verbindung zum Backend her, um Echtzeit-Updates zu empfangen.

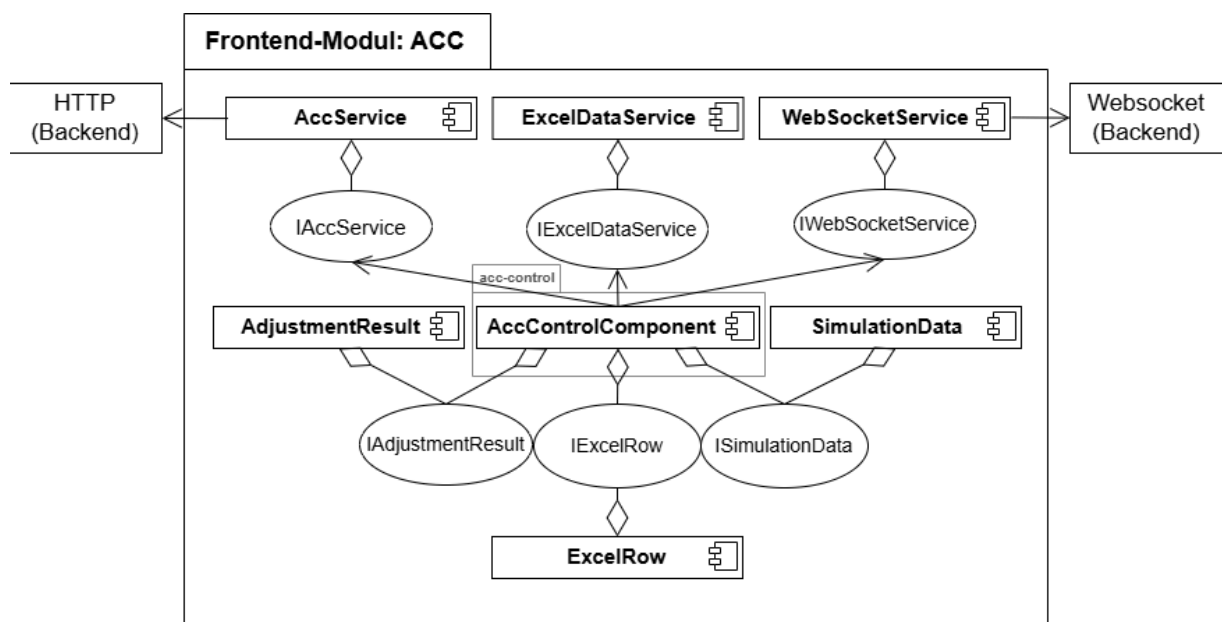
## 2. Model-Classes

**SimulationData:** Repräsentiert die Datenstruktur für die Simulation (entspricht der Backend-DTO ACC.project.models.SimulationData).

**ExcelRow:** Definiert die Struktur einer Excel-Zeile.

## 3. Komponenten

**AccControlComponent:** Hauptsteuerungskomponente für die Benutzeroberfläche der ACC-Simulation.



Der **AccControlComponent** steuert die Simulation über Buttons wie "Start" und "Next" und zeigt Anpassungsergebnisse an. Er nutzt die Interfaces **IAccService** (HTTP-Kommunikation), **IExcelDataService** (Excel-Datenverwaltung) und **IWebSocketService** (Echtzeitdaten) sowie die Datenmodelle **ISimulationData**, **IExcelRow** und **IAdjustmentResult**. Der **AccService** stellt **IAccService** bereit und kommuniziert mit dem Backend über HTTP, während der **ExcelDataService** **IExcelDataService** implementiert, um Excel-Zeilen zu verwalten. Der **WebSocketService** bietet **IWebSocketService** für Echtzeit-Updates. **SimulationData** und **ExcelRow** sind Datenmodelle, und **AdjustmentResult** repräsentiert Anpassungsergebnisse vom Backend.

## 4.3 Spezielle Themen (Frontend)

### Spezielles Thema: WebSockets

#### Problemstellung und Motivation

In der ACC-Simulation sollten Fahrzeugdaten (Geschwindigkeit, Abstand, Wetter) in Echtzeit aktualisiert werden, um eine realistische Darstellung zu gewährleisten. WebSockets wurden gewählt, um eine bidirektionale, eventgetriebene Kommunikation zwischen Backend und Frontend zu ermöglichen. Dies reduziert die Latenz, spart Ressourcen und simuliert besser ein echtes ACC-System, das kontinuierlich Daten liefert.

#### Beschreibung

WebSockets sind ein Protokoll für persistente, bidirektionale Kommunikation über eine einzige TCP-Verbindung. Im Gegensatz zu HTTP-Anfragen bleibt die Verbindung offen, und der Server kann Daten proaktiv an Clients senden (Push). Im Projekt wurde Spring WebSocket (spring-boot-starter-websocket) im Backend integriert, mit einem `TextWebSocketHandler`, der Simulationsdaten als JSON an verbundene Clients broadcastet. Im Frontend wurde `ngx-websocket` verwendet, um eine `WebSocketSubject`-Verbindung zu `ws://localhost:8080/simulation` aufzubauen und Daten als Observable zu empfangen.

#### Verweis auf den Code

Im Backend ist die Implementierung in `SimulationWebSocketHandler.java`

(Broadcast bei jedem `runControlLoop`) und `WebSocketConfig.java` (Endpoint `/simulation`)



```

import ...
@Configuration no usages 1 EddyStone7000
@EnableWebSocket
public class WebSocketConfig implements WebSocketConfigurer {
    private final SimulationWebSocketHandler simulationWebSocketHandler; 2 usages

    public WebSocketConfig(SimulationWebSocketHandler simulationWebSocketHandler) { no usages 1 EddyStone7000
        this.simulationWebSocketHandler = simulationWebSocketHandler;
    }

    @Override no usages 1 EddyStone7000
    public void registerWebSocketHandlers(WebSocketHandlerRegistry registry) {
        registry.addHandler(simulationWebSocketHandler, "/simulation").setAllowedOrigins("*");
    }
}

```

Im Frontend befindet sich der Code in websocket.service.ts (Verbindung)

```

src > app > TS websocket.service.ts
@Injectable({
  providedIn: 'root'
})
export class WebSocketService {
  private socket: WebSocket | null = null;
  private dataSubject = new Subject<SimulationData>();
  private reconnectAttempts = 0;
  private maxReconnectAttempts = 5;

  constructor(@Inject(PLATFORM_ID) private platformId: Object) {
    if (isPlatformBrowser(this.platformId)) {
      this.connect();
    }
  }

  private connect(): void {
    const url = 'ws://localhost:8080/simulation';
    console.log("Versuche, WebSocket zu verbinden mit ${url}");
  }
}

```

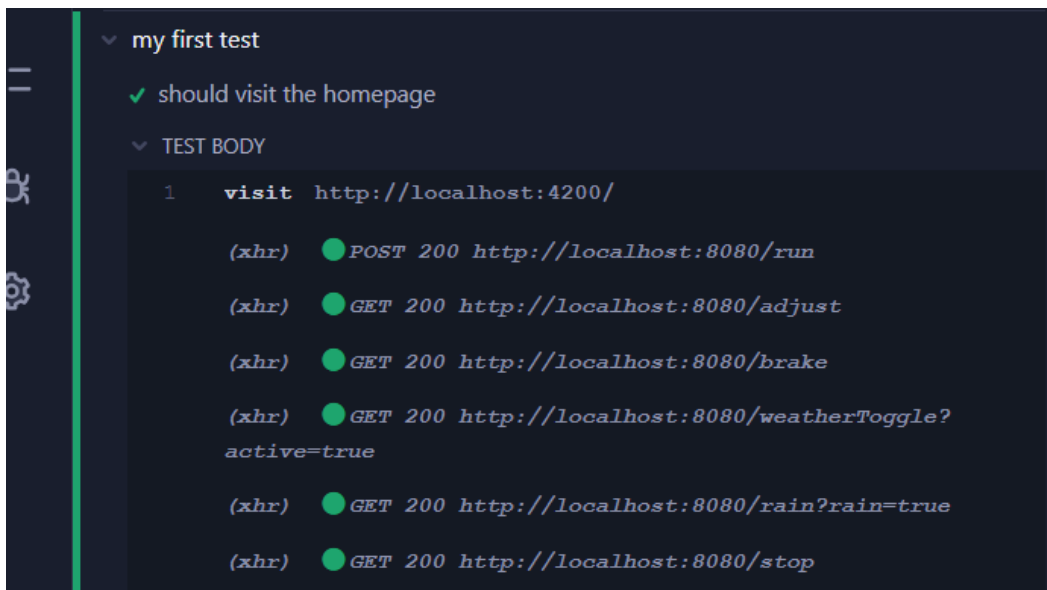
und acc-control.component.ts (Subscription statt Polling).

## 4.4 Frontend: Test mit Cypress

Am anfang sind folgende Parameters gesetzt:

Eigene Geschwindigkeit: 100.0 km/h, Vorfahrende Geschwindigkeit: 120.0 km/h,  
Abstand: 5.0 m.

**Eigene Geschwindigkeit: 100.0 km/h**  
**Vorfahrende Geschwindigkeit: 120.0 km/h**  
**Abstand: 5.0 m**



Mit **run** wird die Simulation mit Parametern: Eigene Geschwindigkeit: 100.0 km/h, Vorfahrende Geschwindigkeit: 120.0 km/h, Abstand: 5.0 m gestartet, wobei sich der Abstand kontinuierlich vergrößert.

**Status:** Simulation läuft

**Eigene Geschwindigkeit:** 100.0 km/h

**Vorfahrende Geschwindigkeit:** 119.9 km/h

**Abstand:** 29.3 m

Mit **adjust** passt sich das Ego Fahrzeug an mit Parametern: Eigene Geschwindigkeit wird beschleunigt. Der Abstand verringert sich auf ca 5-6 m.

Mit **brake** wird Starkes Bremsen simuliert. Parametern: Vorfahrende Geschwindigkeit fällt sofort auf ca 80 km/h. Der Abstand verringert sich bis auf den kritischen Bereich. Die eigene Geschwindigkeit verringert sich auf ca 80 km/h.

Mit **weatherToggle** wird ein Fremd-API aufgerufen.

Mit **rain** werden Wetterbedingungen simuliert. Parametern: Geschwindigkeit der Autos bleibt gleich. Der Abstand vergrößert sich auf ca 10 m.

Mit **stop** stoppt die Simulation, die Werte werden zurückgesetzt.

## Adaptive Cruise Control Simulation


ACC Steuerung

**Status:** Simulation ist aus

**Eigene Geschwindigkeit:** 100.0 km/h

**Vorausfahrende Geschwindigkeit:** 120.0 km/h

**Abstand:** 5.0 m

 Clear, 0.0 °C, 0.0 m/s, Berlin

Simulation Starten

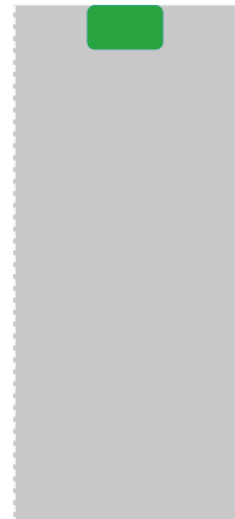
Pilot

Bremsen

Wetter an

Rain

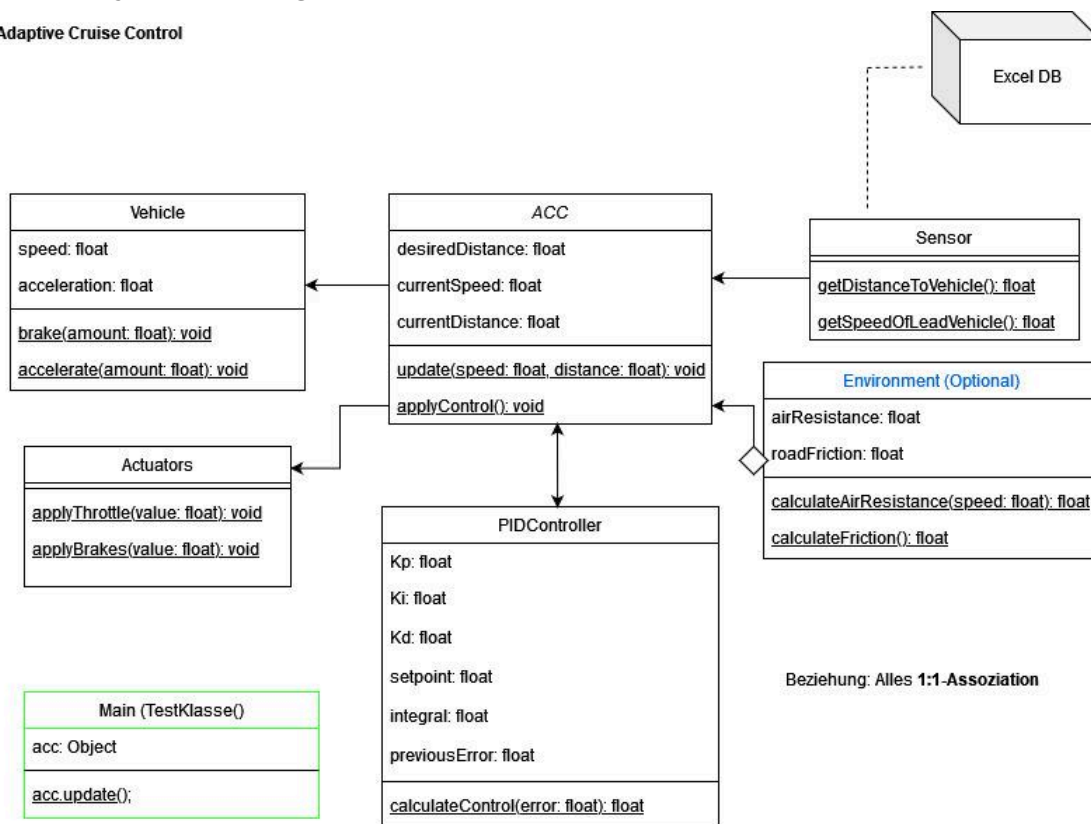
Next



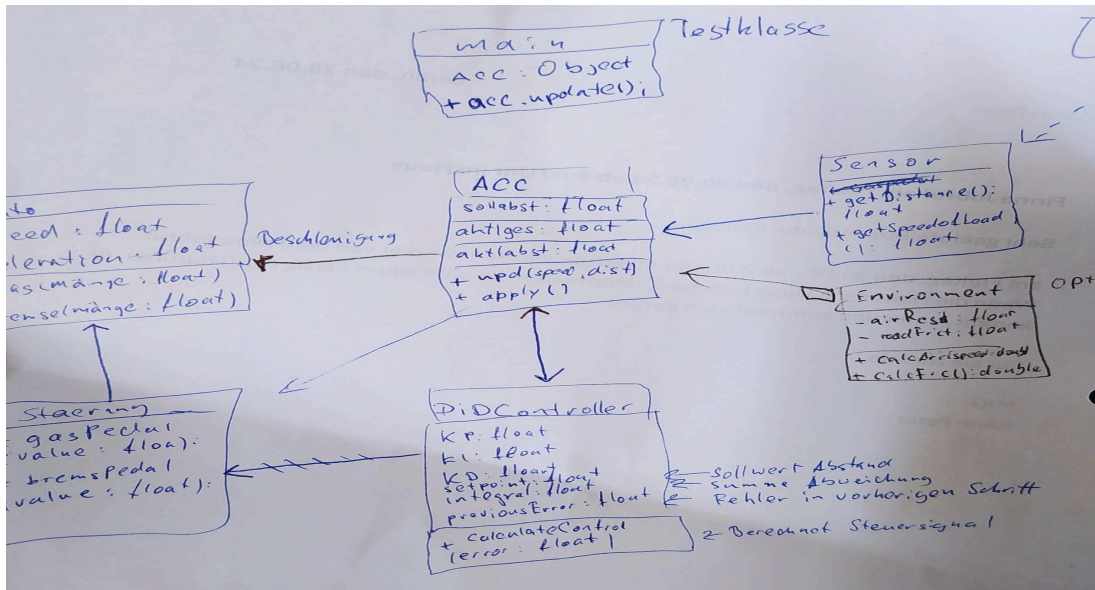
## 5. Organisatorisches

### 5.1 Projektmanagement

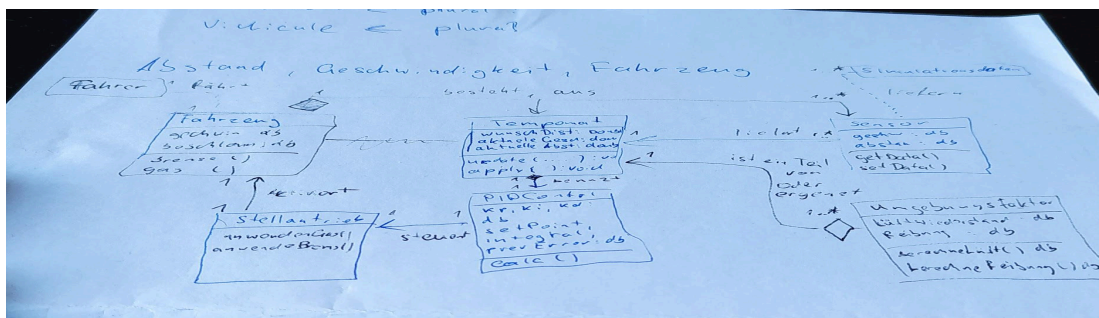
Adaptive Cruise Control



Zu Beginn wurden verschiedene Datenmodelle entwickelt, wobei einige mehr oder weniger Entitäten enthielten.



Es gab unterschiedliche Beziehungen zwischen den Klassen, die gehandhabt und kombiniert wurden.



Viel Zeit wurde in die Logik des PID-Controllers und die Entwicklung seiner verschiedenen Algorithmen investiert. Besonders das Eintauchen in neue Frameworks wie Spring und Angular nahm den Großteil der Zeit in Anspruch. Die Implementierung verlief anfangs eigentlich gut, doch je tiefer man in die Details eintauchte und neue Features integrierte, desto komplexer, riskanter und unübersichtlicher wurde das Projekt. Dennoch wurde es am Ende erfolgreich umgesetzt und funktioniert, was wirklich zufriedenstellend ist. Aufgrund von Zeitmangel blieben jedoch Aspekte wie Code-Pflege, Kommentierung, Vereinfachung, Javadoc, Analyse und das Entfernen von Rudimenten leider vernachlässigt. Für die Zukunft wäre es sinnvoll, solche großen Projekte stärker in kleinere Abschnitte zu unterteilen und mehr Zeit einzuplanen.

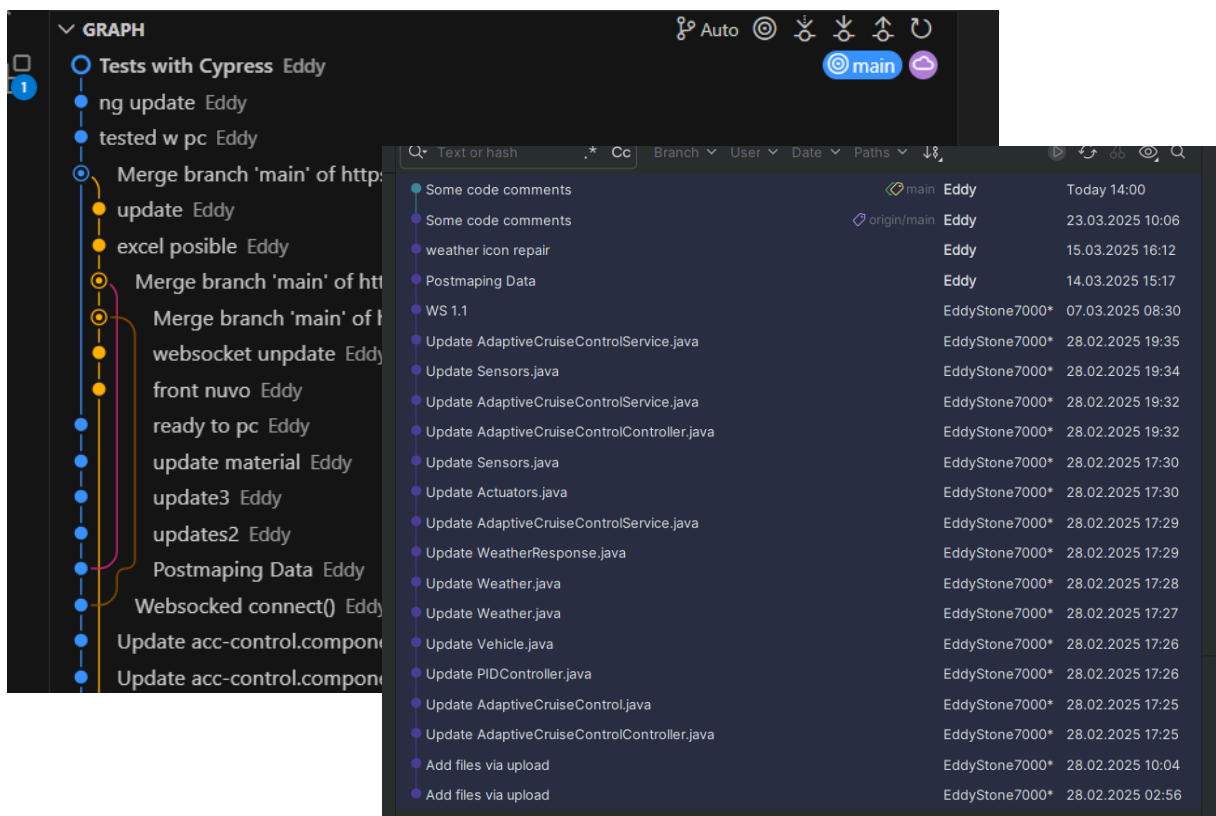
## 5.2 Versionierung

Da ich am Projekt allein gearbeitet habe, sind meine Commits und Merges nicht besonders spektakulär.

**Github:**

<https://github.com/EddyStone7000/accFrontend.git>

<https://github.com/EddyStone7000/accBackend.git>



## 6. Referenzen

Referenzen:

[https://www.ni.com/en/shop/labview/pid-theory-explained.html?srsId=AfmBOoq5Z8MTqAZOgr\\_ra3ivGuB8rlp9PW09JDlc1r2MG0Vcnnlg7UdS](https://www.ni.com/en/shop/labview/pid-theory-explained.html?srsId=AfmBOoq5Z8MTqAZOgr_ra3ivGuB8rlp9PW09JDlc1r2MG0Vcnnlg7UdS)

<https://docs.oracle.com/javase/tutorial/essential/concurrency/>

[https://developer.mozilla.org/en-US/docs/Web/API/WebSockets\\_API](https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API)

<https://datatracker.ietf.org/doc/html/rfc6455>

<https://www.reactivemanifesto.org/>