

Architektur-Dokumentation zur “Bike-Fitting”-Applikation

Projekt im Rahmen des Wahlpflichtfachs “Agile Webentwicklung mit Python” an der Technischen Hochschule Augsburg im Sommersemester 2023

Für diese Dokumentation wird das Template ARC-42 von Stefan Zörner verwendet

Alle Punkte, die nicht relevant sind für die Bike-Fitting App, werden nicht behandelt

Inhaltsverzeichnis

1. Einführung und Ziele	4
1.1. Motivation	4
1.2. Aufgabenstellung	4
1.1. Qualitätsziele	5
1.2. Stakeholder	5
2. Randbedingungen	5
2.1. Technische Randbedingungen	5
2.2. Organisatorische Randbedingungen	6
2.3. Konventionen	6
3. Kontextabgrenzung	6
3.1. Fachlicher Kontext	6
3.2. Technischer- oder Verteilungskontext	6
4. Lösungsstrategie	6
4.1. Architektur gegenüber Lösung	6
4.2. Aufbau der Bike-Fitting-App	7
4.3. Die Anbindung	8
5. Bausteinansicht	8
5.1. Das Datenmodell	9
5.2. Die View-Methoden	10
5.3. Die HTML-Templates	11
5.4. Das Account-System	12
6. Laufzeitansicht	13
6.1. Dynamische Verhalten der Software	13
7. Verteilungssicht	13
7.1. Artefakte	13
7.2. Knoten	13
8. Konzepte	13
8.1. Abhängigkeiten zwischen Modulen	13
8.2. Domänenmodell	13
8.3. Ausnahme und Fehlerbehandlung	14
8.4. Logging, Tracing	14
8.5. Benutzeroberfläche	14
8.6. Testbarkeit	16
9. Risiken	17
9.1. Kritische Punkte und Herausforderungen	17
9.2. Risikostrategie	17
10. Abbildungsverzeichnis	18

11. Tabellenverzeichnis.....	18
12. Literaturverzeichnis.....	18

1. Einführung und Ziele

1.1. Motivation

Moderne Fahrräder sind nicht mehr mit ihren Verwandten aus längst vergangenen Zeiten vergleichbar. Um das Potential dieser modernen High-Tech-Fahrzeuge vollends ausnutzen zu können und um die eigene Gesundheit nicht zu gefährden, ist es essentiell, dass diese modernen Räder perfekt auf deren Fahrerinnen und Fahrer abgestimmt sind. Wir möchten mit dieser Anwendung einen einfach zu bedienenden Leitfaden erstellen, der es auch Laien ermöglicht, das maximale Potential aus ihren Sportgeräten herauszukitzeln und dabei den gesundheitlichen Aspekt nicht außer Acht zu lassen.

1.2. Aufgabenstellung

Was ist das Ziel der Bike-Fitting Software?

- Ist eine Webbasierte Anwendung
- Es soll den Usern Auskunft darüber geben, wie sie das Fahrrad einstellen sollten
- Das Projekt wurde außerdem als Semesterprojekt durchgeführt und dient dem Lernzweck der Entwickler

Wesentliche Features in Form von Use-Case-Diagrammen:

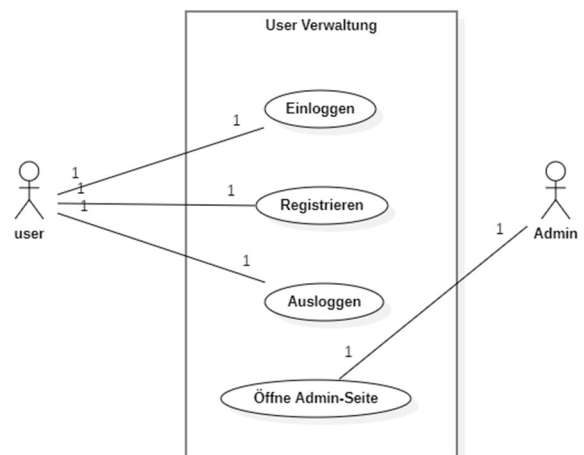
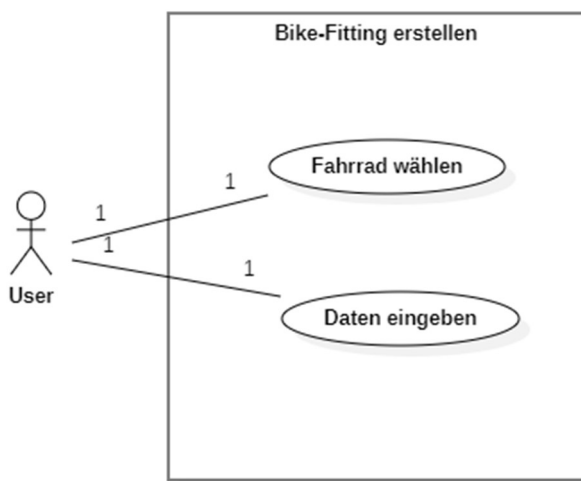


Abbildung 1 Use Case User Verwaltung

Abbildung 2 Use Case Fitting erstellen

1.1. Qualitätsziele

Die Tabelle beschreibt die zentralen Qualitätsziele der Bike-Fitting-App, wobei die Reihenfolge eine grobe Orientierung bezüglich der Wichtigkeit vorgibt.

Tabelle 1 Qualitätsziele der Bike-Fitting-App

Qualitätsziel	Motivation und Erläuterung
Analysierbarkeit	Das Programm soll leicht zu lesen sein und neue Entwickler sollen sich gut einarbeiten können. Auch internationale Entwickler sollen nicht zu kurz kommen.
Experimentierfreudig Änderbarkeit	Die verschiedenen Bereiche sollen entkoppelt werden können. Speziell das Backend soll so wenig Abhängigkeiten wie möglich haben, sodass jederzeit ein anderes Frontend-Framework eingesetzt werden kann. Die Implementierung soll leichtfallen.
Webanwendung Interpolarität	Die Software soll für alle Nutzer zugänglich sein und jeder Rechner soll damit umgehen können.
Attraktivität	Der Algorithmus, der von der Software verwendet wird, soll ein sinnvolles Ergebnis in effizienter Zeit liefern und das in glatten Maßeinheiten die dem User geläufig sind.
Verständlichkeit	Die Software soll einfach und intuitiv bedienbar sein. Bei der Gestaltung der HTML-Seiten soll auf bewährte Standards zurückgegriffen werden.
Erweiterbarkeit	Die Applikation soll ohne Probleme um zusätzliche Seiten erweiterbar sein.

1.2. Stakeholder

Die Entwickler, Geldgeber und auch Inhaber dieser Software sind Sebastian Niehage und Lukas Konietzka. Die Software wurde im Zuge eines Semesterprojektes entwickelt.

2. Randbedingungen

2.1. Technische Randbedingungen¹

Die Software wird mit dem Python-Framework Django umgesetzt.

Die technischen Voraussetzungen, die diese Software stellt, halten sich aufgrund der sehr simplen Architektur in Grenzen. Es wird ein moderner Browser benötigt, der in der Lage ist, den CSS und HTML-Quellcode anzuzeigen. Außerdem ist eine IDE nötig die die Sprache Python versteht und das Framework Django mit all seinen Paketen installiert hat.

Für das Testframework wird Pytest verwendet.

¹ (Seifert, Sommer Semester 2023) Als Anleitung zur Erstellung einer Python Anwendung

Als Versionskontrolle verwenden wir das Tool Git auf folgendem remote-Server:

<https://github.com/lukaskonietzka/bikefitting>

2.2.Organisatorische Randbedingungen

Die Entwicklung der Applikation findet im Rahmen eines Moduls an der Technischen Hochschule in Augsburg statt und beginnt mit dem Start der Vorlesung Webanwendungen am 01.04.2023. Im Laufe der Veranstaltung handelt es sich die Software von einer lauffähigen Version zur nächsten lauffähigen Version.

Die Anwendung soll bis zum 14.07.2023 vollständig einsatzfähig und dokumentiert sein.

2.3.Konventionen

Es gelten die allgemein üblichen Konventionen für Entwicklungen im Python Ökosystem. Als Beispiel seien hier einige Punkte genannt:

- Funktionsnamen im Snake-Case
- „_“ für private Attribute, die besser nicht verändert werden
- Umbruch nach einer definierten Anzahl von Zeichen innerhalb einer Zeile

3. Kontextabgrenzung

3.1.Fachlicher Kontext

Bei der Software handelt es sich um eine Webanwendung, welche demnach keine Benutzergruppen ausschließt.

Der Programmcode ist seriell geschrieben und nicht verteilt. Demnach interagiert dieses System mit keinem weiteren.

3.2.Technischer- oder Verteilungskontext

Das System verwendet den ASCII-Code bei der Darstellung von Strings sowie Integer und Floats bei der Darstellung numerischer Größen

4. Lösungsstrategie

4.1.Architektur gegenüber Lösung

Die Tabelle stellt die Qualitätsziele aus Tabelle 1 passenden Architekturen gegenüber und erleichtert so einen Einstieg in die Lösung.

Tabelle 2 Lösung Strategie für Qualitätsziele

Qualitätsziel	Ansätze in der Architektur
Analysierbarkeit	<ul style="list-style-type: none">- Architektur gegliedert nach arc42- Explizites objektorientiertes Datenmodell- Module-, Klassen-, Methodennamen in Englisch, Doku in Deutsch
Experimentierfreudig Änderbarkeit	<ul style="list-style-type: none">- Verbreitete und einfach zu lernenden Programmiersprache mit einfachem Framework für Webentwicklung (Python, Django)- Erweiterbarkeit der Datenstrukturen um neue Fittings gegeben durch Objektorientierung
Interpolarität	<ul style="list-style-type: none">- Die Software wurde als Webanwendung implementiert

Attraktivität	<ul style="list-style-type: none"> - Auf bereits bestehende, funktionierende Algorithmen zugreifen - Algorithmus soll auf Konstanten basieren, damit er immer angepasst werden kann, ohne ihn umschreiben zu müssen
Verständlichkeit	<ul style="list-style-type: none"> - Einfache Formulierungen in der UI - Wenig Text in der UI - Intuitive Bedienung der UI
Erweiterbarkeit	<ul style="list-style-type: none"> - Einfach Erweiterung um zusätzliche Fittings soll schnell und einfach möglich sein. - Dies soll geschehen, indem eine zusätzliche Klasse für das entsprechende Fitting implementiert wird. Alle wichtigen Methoden sollen per Vererbung dazukommen.

4.2.Aufbau der Bike-Fitting-App

Die Bike-Fitting-App zerfällt auf Programmebene grob in folgende Teile:

- Datenstruktur
- Die View Methoden
- Die HTML-Seiten, die angezeigt werden
- Das Account-System

Die verschiedenen Teile sind durch Schnittstellen abstrahiert und so auch bei Bedarf austauschbar. Die allgemeinen Konventionen *Separation of Control* und *Separation of Concerns* wurden geachtet. Das Frontend soll vom Backend abkoppelbar sein, sodass ohne Probleme ein anderes Datenmodell mit dem Kontext Bike-Fitting installiert werden kann.

Die App ist mit dem MVC-Pattern gebaut, da dies durch das Framework Django mehr oder weniger vorgegeben ist.

Die folgende Tabelle soll Auskunft geben welche Teile der Software welche Rolle im MVC-Pattern übernehmen.

Tabelle 3 MVC Aufteilung auf Module

Software	MVC-Beschreibung
Datenmodell in models.py	Model
Die HTML-Templates	View
Das Modul view.py	Controller

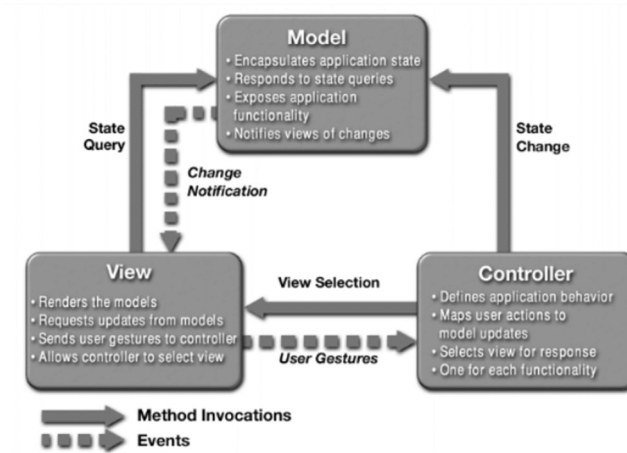


Abbildung 3 MVC Prinzip

4.3. Die Anbindung

Bei dem Computerprogramm handelt es sich um eine Webanwendung, die auf allen standardisierten Browsern laufen soll (Chrome, Firefox, Safari, ...). Es ist demnach keine aufwändige Einbindung nötig.

5. Bausteinansicht

Dieser Abschnitt beschreibt die Zerlegung der Applikation in Module. Wir betrachten in der Bike-Fitting-Anwendung vier verschiedene Bereiche, wie sie bereits oben erwähnt wurden:

- Das Datenmodell
- Die View-Methoden
- Die Templates selbst
- Das Account-System

Tabelle 4 Mögliche Subsysteme

Subsystem	Kurzbeschreibung
Datenmodell	Das Datenmodell verwaltet den Algorithmus zum Erstellen eines Fittings, sowie alle Strukturen des Datenmodells selbst.
View-Methoden	Über die View-Methoden lässt sich das Datenmodell ansteuern und die Informationen an die Templates weitergeben. Die View-Methoden geben jeweils ein HTML-Element zurück.
Templates	Die Templates sind Objekte, die von den View-Methoden zurückgegeben werden. Sie beinhalten den HTML- und CSS-Code für das Anzeigen der Elemente im Browser.
Das Account-System	Hier wird die User-Verwaltung aufgebaut. Django bringt hierfür schon eine fertige Lösung mit, die sich bewährt hat. Wir möchten diese nutzen.

5.1. Das Datenmodell

Das Datenmodell besteht aus einer Eltern-Klasse "Fitting" und deren Kind-Klassen *Mountainbike*, *Roadbike*, *Trekkingbike*. Der Algorithmus zum Berechnen der Fittings ist für alle Arten von Fittings gleich. Sie unterscheiden sich ausschließlich in den Konstanten.

Dies ermöglicht es uns, die Anwendung schnell und einfach um weitere Fittings (z.B. *Downhill*) zu erweitern

Durch die Klasse Fitting wird das Datenmodell der Anwendung definiert.

Folgende Felder bilden die Attribute im Datenmodell:

- Name
- Height
- StepLength

Alle drei Felder werden vom User eingegeben und in der Datenbank gespeichert.

Die Felder *frameHeight* und *saddleHeight* werden nicht extra abgespeichert, da diese jederzeit mit dem Algorithmus berechnet werden können.

Um ein neues Fitting anzulegen, muss folgender Aufruf gestartet werden:

```
rb = Roadbike()
frame_height, saddle_height, bike_type = rb.create_roadbike_fitting(name, height, step_length)
```

Die Methode gibt ein Tupel aus drei Daten zurück. Bei Bedarf können aber durch sehr wenig Änderungen alle Felder des Datenmodells berechnet und zurückgegeben werden. So ist garantiert, dass das Datenmodell sehr variabel einsetzbar ist.

Das nachfolgende Klassendiagramm zeigt den Aufbau des Datenmodells.

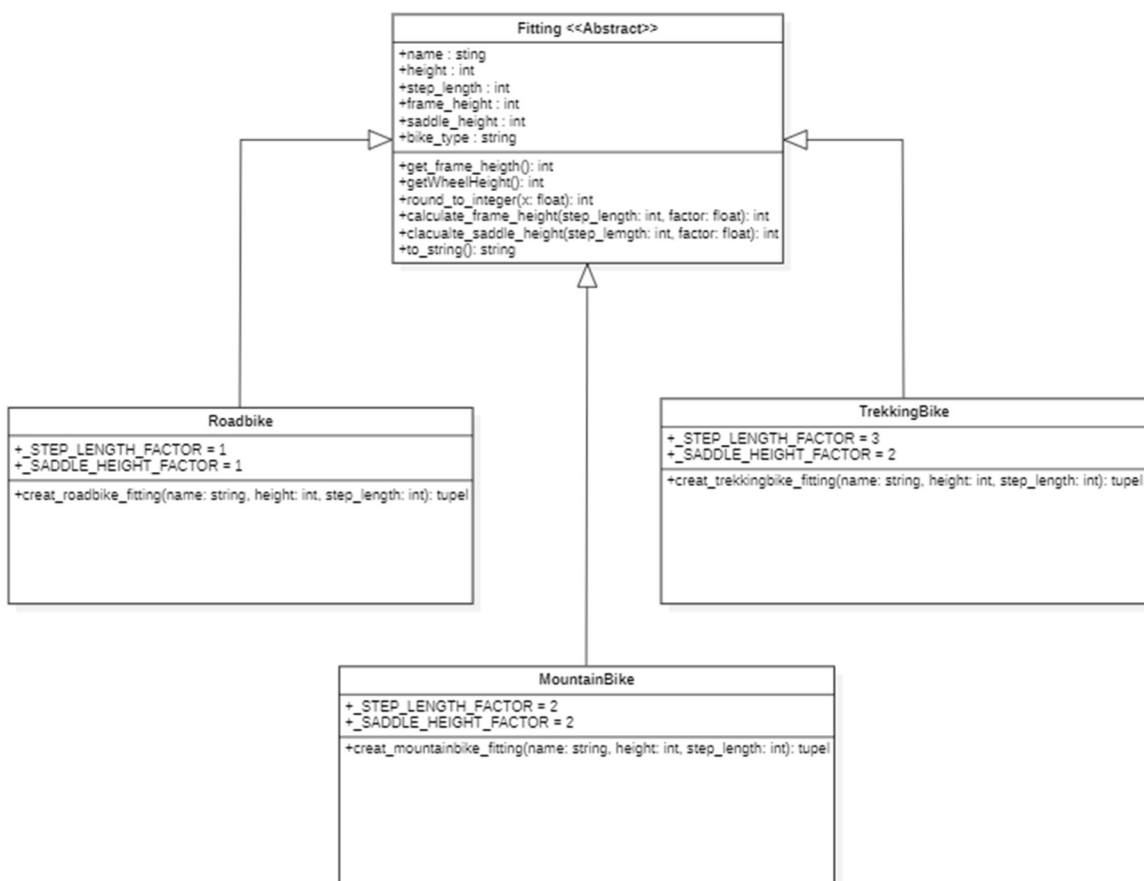


Abbildung 4 Das Datenmodell

Die folgende Tabelle listet die Beschreibung der wichtigsten Methoden auf.

Tabelle 5 Wichtige Methoden des Datenmodells

Methode	Kurzbeschreibung
<code>calculate_Frame_Height(step_length, factor)</code>	Berechnet die Rahmenhöhe auf Basis der Parameter <code>step_length</code> und <code>factor</code> . Gibt einen Integer zurück.
<code>calculate_Saddle_Height(step_length, factor)</code>	Berechnet die Sattelhöhe auf Basis der Parameter <code>step_length</code> und <code>factor</code> . Gibt einen Integer zurück.
<code>creat_Fitting(name, height, stepLength)</code>	Erstellt das entsprechende Fitting, in dem die generischen Methoden verwendet werden. Gibt ein Tupel zurück.

Die nachfolgende Tabelle gibt Aufschluss darüber, wie wir die Faktoren für die Berechnung gewählt haben.

Bei den Faktoren handelt es sich um Erfahrungswerte, die wir als Techniker und ambitionierte Radfahrer im Laufe der Zeit gesammelt haben.

Tabelle 6 Faktoren für die Berechnung

	Rahmenhöhe	Sattelhöhe
Mountainbike	0,254	0,89
Rennrad	0,667	0,88
Trekkingrad	0,663	0,95

5.2. Die View-Methoden

Die View-Methoden sind das zentrale Element in dieser Anwendung. Nach dem MVC-Pattern übernehmen sie die Rolle der Controller. Sie nehmen Informationen des Models entgegen, verwalten diese und geben sie an die View-Elemente weiter.

Insgesamt gibt es 7 View-Methoden, die jeweils eine andere Seite anzeigen. Zwei dieser Methoden sind für den Error 404 und den Error 500. Es existiert eine weitere Methode namens `show_message()`. Diese nimmt eine message als string entgegen und gibt ein Dictionary zurück. Mit dieser Methode lassen sich Nachrichten generisch an ein HTML-Template übergeben. Die Methode muss nur in der entsprechenden Render-Methode aufgerufen werden und es muss die gewünschte Nachricht eingegeben werden.

Die Anwendung verfügt außerdem über eine User-Verwaltung. Benutzerinnen und Benutzer können sich registrieren und anmelden. Dies geschieht über die von Django mitgebrachten Methoden `login()`, `register()` und `logout()`. Für die meisten Seiten der Bike-Fitting-App wird eine Anmeldung benötigt. Nur dann kann die App im vollen Umfang benutzt werden.

Die nachfolgende Tabelle gibt Aufschluss darüber, welche wichtigen Methoden beteiligt sind.

Tabelle 7 Wichtige Methoden der Views

Methode:	Kurzbeschreibung:
<code>index()</code>	Rendert die Startseite.
<code>select_bike()</code>	Render die Seite auf der der Fitting Typ ausgewählt wird. Die Methode Speichert die Daten lokal und übergibt sie der Methode <code>result()</code> .
<code>measure_step_length()</code>	Rendert die Seite Messen
<code>input_data()</code>	Rendert die Seite Daten Eingeben und nimmt die Daten des Users entgegen. Gibt sie dann an die Methode <code>result()</code> weiter.
<code>result()</code>	Rendert die Seite Ergebnis und erstellt das entsprechende Fitting-Objekt. In der Methode <code>result()</code> wird auch der Datenbankaufruf gestartet.

5.3.Die HTML-Templates

Die Software verwendet HTML-Templates, die von den View-Methoden zurückgeben werden. Diese Templates beinhalten den HTML- und CSS-Code für das Rendern der Seiten.

Tabelle 8 wichtige Templates

Template	Kurzbeschreibung
Index.html	HTML-Code für die Startseite, hier herrscht keine Funktionalität
Select_bike.html	Hier kann der User das gewünschte Fitting auswählen.
Measure_step_length.html	Hier erhält der User eine Kurzbeschreibung, wie er seine Schrittlänge ermitteln kann.
Input_Data.html	Hier kann der User seinen Namen, die Körpergröße und die gemessene Schrittlänge eingeben.
Result.html	Der User erhält hier seine angefragten Daten. In Summe sind das Name, Typ, Körpergröße, Schrittlänge, Rahmenhöhe, Sattelhöhe
Base.html	Base.html ist die wichtigste Seite. Diese Seite ist Elternklasse für alle anderen Templates. Hier liegen die CSS-Regeln und die Grundstruktur der Seite ist hier gebaut, d.h. die Navigationsleiste, die Überschriften der Seiten und die Fußleiste.

Wie das Template base.html genau aussieht, sollen folgende Abbildungen verdeutlichen.



Abbildung 5 Generische Navigationsleiste



Abbildung 6 Generischer Footer

5.4. Das Account-System

Damit die App im vollen Umfang genutzt werden kann, soll der User sich registrieren, anmelden und abmelden können. Hierfür nutzen wir die von Django mitgebrachten Funktionen.

Um für eine HTML-Seite dann einen Login zu erzwingen, muss vor die View-Methode nur der entsprechende Decorator geschrieben werden.

```
@login_required()
def select_bike(request):
    """
    Generating the "select Bike" page
    and creat the object that is needed.
    If more types are selected, we switch to the error-page
    An login is required.
    :param request: Data from the html file
    :return: The selectBike.html file to render
    """

    global current_bike
    if request.POST:
        bikes = request.POST.getlist('bike')
        if len(bikes) != 1:
            current_bike = None
            return render(request, 'error.html', show_message("Bitte wählen Sie zuerst ein Fahrrad aus"))
        current_bike = bikes[0]
    return render(request, 'selectBike.html')
```

Abbildung 7 Decorator für das Login

Tabelle 9 Wichtige Methoden Account-System

Methode	Beschreibung
<code>register_view()</code> ²	Der Inhalt dieser Methode ist grundsätzlich sehr generisch und kann deshalb auch in unserem Projekt eingesetzt werden. Die Methode verwendet Django-forms, um die Registrierung sicherzustellen.
<code>LoginView.as_view()</code> ³	Diese Methode kommt aus der Django Bibliothek und baut uns eine fertige Login-Seite
<code>LogoutView.as_view()</code> ⁴	Diese Methode kommt aus der Django Bibliothek und baut uns eine fertige Logout-Seite

² (Patidar, kein Datum) Inhalt der Methode.

³ (Patidar, kein Datum) Die Art und Weiße wie diese Methode verwendet wird.

⁴ (Patidar, kein Datum) Die Art und Weiße wie diese Methode verwendet wird.

6. Laufzeitansicht

6.1. Dynamische Verhalten der Software

Da es sich um eine sehr kleine Anwendung handelt, wird auf eine Performance getriebene Entwicklung verzichtet.

7. Verteilungssicht

7.1. Artefakte

Unter einem Artefakt versteht man z.B. ein ausführbares Programm

Bei diesem Softwareprodukt handelt es sich nicht um ein Verteiltes System, demnach gibt es nur einen Knoten mit einem Artefakt.

7.2. Knoten

Ist das System, auf dem das Artefakt läuft.

Bei diesem Softwareprodukt handelt es sich nicht um ein Verteiltes System, demnach gibt es nur einen Knoten mit einem Artefakt.

8. Konzepte

8.1. Abhängigkeiten zwischen Modulen

Das Softwareprodukt verfügt über eine wesentliche Schnittstelle. Diese ist im Modul `view.py` in der Methode `results()` zu finden. Hier findet der Backend-Aufruf statt. Es wird also hier das Backend mit dem Frontend verknüpft. Die ist die einzige Stelle in der Software, an der das Backend mit dem Frontend kommuniziert.

8.2. Domänenmodell

Das Computerprogramm verwendet eine objektorientierte Architektur und besteht demnach nur aus einfachen Datenstrukturen, die aus Klassen aufgebaut sind. Dieses Kapitel soll nun einen groben Überblick über die Datenstrukturen und deren Abhängigkeiten schaffen. Alle gezeigten Klassen befinden sich im Modul `modles.py`

Das Datenmodell wurde so aufgebaut, dass es ohne Probleme um Fittings erweiterbar ist, ohne den Algorithmus selbst zur Berechnung verändern zu müssen. Um das Modell erweitern zu können, müssen folgende Punkte durchgeführt werden.

1. Schreiben einer neuen Klasse für das entsprechend Fitting
2. Diese neu erstellt Klasse muss von *Fitting* erben
3. Diese Klasse bekommt Felder mit den Faktoren
4. Schreiben einer Methode `create_ ... _fitting()` die die generischen Methoden zur Berechnung der Rahmenhöhe und der Sattelhöhe verwendet.

```

class Mountainbike(Fitting):
    """
    Create a Mountainbike-Fitting depending on the following fields
    Child-class from: Fitting
    """
    _SADDLE_HEIGHT_FACTOR = 3
    _STEP_LENGTH_FACTOR = 3

    2 usages  ▲ LukasKonietzka +1
    def create_mountainbike_fitting(self, name, height, step_length):
        """
        Get and calculate all required datas to work with them
        :param name: user_name
        :param height: user_height
        :param step_length: user_step_length
        :return: frame_height, saddle_height, bike_type
        """
        self.m_name = name
        self.m_height = height
        self.m_step_length = step_length
        self.m_frame_height = self.calculate_frame_height(step_length, self._STEP_LENGTH_FACTOR)
        self.m_saddle_height = self.calculate_saddle_height(step_length, self._SADDLE_HEIGHT_FACTOR)
        self.bike_type = "Mountainbike"
        data = (self.m_name,
                self.m_height,
                self.m_step_length,
                self.m_frame_height,
                self.m_saddle_height,
                self.bike_type)
        return data[3:]

```

Abbildung 8 Beispiel, Erweiterung des Datenmodells um weitere Fittings

Um nun auch ein Fitting erstellen zu können, muss es natürlich auch noch vom Frontend aus erreichbar gemacht werden. Dies sei hier jedoch nicht erwähnt, da es stark vom Einsatzzweck der Weiteren Klasse abhängt.

8.3.Ausnahme und Fehlerbehandlung

Die Bike-Fitting-App ist keine Fail-fast Anwendung. Das heißt, ein Fehler führt nicht direkt zum Abbruch. Die Methoden der Applikation werfen keine Exception. Es wird keiner Fehlerbehandlung nachgegangen. Tritt ein unerwarteter Fehler auf, so landet der User auf der Error-Seite. Er oder Sie kann dann über die Navigation zur gewünschten Seite zurück navigieren.

8.4.Logging, Tracing

Da es sich bei diesem Softwareprodukt um eine eher kleine Anwendung handelt wird auf einen Einsatz eines Loggers oder eines Trace verzichtet. Bei Bedarf ist eine Implementierung möglich.

8.5.Benutzeroberfläche

Das Computerprogramm ist ein Webanwendung und soll demnach in jedem aktuellen Browser laufen. Tabelle 7 zeigt das Layout der Anwendung wie es in einem solchen Browser aussieht.

Alle Seiten der Bike-Fitting-App verfügen auch über einen Scroll-Mechanismus. Das bedeutet, dass auf den verschiedenen HTML-Seiten auch durchaus mehr Inhalt angezeigt werden kann. Die Navigationsleiste ist hierbei ein Sticky-Element.

Start

Fahrrad

Messen

Daten Eingeben

Ergebnis

Logout

Registrieren

Login

Start

In drei Schritten zum eigenen Bikefitting

Wenn Sie Ihre Schrittlänge schon kennen, können Sie Schritt 2 überspringen.

1. Fahrradtyp wählen Sie wählen aus, für welche Fahrradtypen Sie ein Fitting erstellen möchten	2. Schrittlänge messen Wir geben eine Anleitung, wie Sie Ihre Schrittlänge ermitteln können	3. Daten eingeben Sie geben die benötigten Daten ein, um ein Fitting zu erstellen. Hier wird auch die Schrittlänge benötigt
Start Fahrradwahl	Start Messen	Start Eingabe

This side was made by Sebastian and Lukas!

Abbildung 9 Startseite

Start

Fahrrad

Messen

Daten Eingeben

Ergebnis




Logout

Registrieren

Login

Fahrradtyp wählen

Wähle bitte den Typ deines Fahrrads aus

Mountainbike Dein Zuhause ist jenseits der Zivilisation. Du ernährst dich von Staub, Dreck und Höhenluft?	Rennrad Dein Zuhause ist die Straße. Du frisst Kilometer und gönnst dir zur Belohnung neue Titanschrauben?	Trekkingbike/SUV Dein Zuhause ist überall und nirgendwo. Du befestigst dein Smartphone mitsamt dem Flapcase hochkant in der Lenkerhalterung?
		
<input type="checkbox"/> Ich fahre Mountainbike	<input type="checkbox"/> Ich fahre Rennrad	<input type="checkbox"/> Ich fahre Trekkingrad
Abschicken		

This side was made by Sebastian and Lukas!

Abbildung 10 Fahrrad Seite

Start

Fahrrad

Messen

Daten Eingeben

Ergebnis

Logout

Registrieren

Login

Schrittlänge ermitteln

So ermittelst du deine Schrittlänge

Um deine Schrittlänge zu messen, nimm dir ein Buch oder einen ähnlichen Gegenstand mit einer geraden Kante zur Hand. Stelle dich ohne Schuhe mit den Beinen etwa schulterbreit mit dem Rücken an eine Wand. Achte darauf, dass du den Gegenstand zwischen deinen Beinen waagrecht nach oben ziehst, bis es wirklich unangenehm wird. Pass dabei auf, dass deine Beine am Boden bleiben! Wenn beide Füße den Boden nicht mehr berühren, ziehst du zu fest. Bitte nun eine zweite Person, den Abstand vom Boden bis zum Buchrücken zu messen. Notiere dir dieses Maß für den nächsten Schritt.



This side was made by Sebastian and Lukas!

Abbildung 11 Messen Seite

Startseite

Fahrradtyp wählen

Schrittlänge messen

Start
Fahrrad
Messen
Daten Eingeben
Ergebnis
Logout
Registrieren
Login

Daten eingeben

Hier kannst du deine ermittelten Daten eingeben

Bitte gebe die Maße in ganzen Zentimetern an (cm)

Name: This field is required.

Height: This field is required.

StepLength: This field is required.

Daten eingeben

This side was made by Sebastian and Lukas!

Abbildung 12 Eingabe Seite

Start
Fahrrad
Messen
Daten Eingeben
Ergebnis
Logout
Registrieren
Login

Ergebnis

Laut unserem Algorithmus sollte dein Fahrrad folgende Maße aufweisen:

Fahrradmaß	Unsere errechneten Ergebnisse
Name:	Cube
Fahrradtyp:	Roadbike
Körpergröße:	180cm
Schrittlänge:	90cm
Benötigte Rahmengröße:	90cm
Benötigte Sattelhöhe:	90cm

Ergebnis erhalten

This side was made by Sebastian and Lukas!

Abbildung 13 Ergebnis Seite

8.6. Testbarkeit

Die Funktionalität der einzelnen Methoden soll ausgiebig durch Unit-test geprüft werden. Soll also konkret eine Klasse getestet werden, dann heißt die Testklasse genauso wie die zu testende Klasse nur mit "Test" am Beginn der Bezeichnung. Mit Hilfe dieser Tests soll die Korrektheit der Anwendung sichergestellt werden.

Die Tests beziehen sich nur auf das Modul `model.py`, da hier die eigentliche Logik stattfindet. Alle Tests finden sich in dem Ordner `Tests`. Als Testframework verwenden wir `Pytest`. Hierfür wurde eine Datei namens `pytest.ini` angelegt, die dafür sorgt, dass Ihre IDE die Tests im entsprechenden Ordner sucht.

Der Inhalt dieser Datei verweist auf zwei Module, `settings.py` und `tests.py`.

Der Inhalt sei hier kurz gezeigt:

```
[pytest]
DJANGO_SETTINGS_MODULE = bikefitting.settings
python_files = tests.py
```

Abbildung 14 Inhalt `pytest.ini`

9. Risiken

9.1. Kritische Punkte und Herausforderungen

Eine Software soll intuitiv und einfach zu bedienen sein, bei der Entwicklung dieser App war das unser wichtigster und gleichzeitig schwierigster Punkt. Unser Anspruch war, dass die Software zu keiner Zeit in einen Zustand gerät, denn wir uns nicht erklären können.

Um so eine Software zu entwickeln, haben wir uns dafür entschieden, weniger Features einzubauen, die dafür umso stabiler sind.

Der User kann also sehr grob mit der Software umgehen, ohne dass er sich verloren fühlt.

Das Erstellen eines Fittings kann z.B. auch in umgekehrter Reihenfolge erfolgen, also konkret können erst Daten eingegeben und im Nachhinein das entsprechende Fahrrad ausgewählt werden.

Auftretende 404-Fehler werden auch wie bereits erwähnt durch eine extra view-Methode abgefangen. Dies ermöglicht es uns sicherzustellen, dass der User in keine Zustand landet, aus dem er nicht zurück kommt.

9.2. Risikostrategie

Die Tabelle beschreibt die Risiken der App, wobei die Reihenfolge eine grobe Orientierung bezüglich der Wichtigkeit vorgibt.

Tabelle 10 Risiko gegenüber Methodik

Risiko:	Methode:
Software wird nicht intuitiv	<ul style="list-style-type: none"> - Wir haben dort wo es ging die Forms von Django verwendet. Diese bieten Codestücke, die bereits eine etablierte Möglichkeit bieten Inputdaten abzufragen - Wir achteten darauf, dass die Software aus allen Richtungen bedienbar ist. Es ist zwar eine Reihenfolge empfohlen, es funktioniert aber auch in die andere Richtung
Software gerät in Zustand, den wir nicht kennen	<ul style="list-style-type: none"> - Durch entkoppelte Komponenten haben wir versucht diesem Risiko entgegenzuwirken. Wir bauten eine Error-Seite die immer aufgerufen wird, wenn es ein Problem gibt. Sollte es wirklich zu dem Fall kommen, dass der Zustand undefiniert ist, dann kann der User über die Navigation einfach zu einer anderen Seite zurückspringen.
Software wird nicht stabil:	<ul style="list-style-type: none"> - Der Umfang wurde klein gehalten mit wenigen Features. Die bestehenden Features wurden aber sauber implementiert, sodass es wenig Abhängigkeiten gibt (Patidar, kein Datum)

10. Abbildungsverzeichnis

Abbildung 1 Use Case User Verwaltung	4
Abbildung 2 Use Case Fitting erstellen	4
Abbildung 3 MVC Prinzip	8
Abbildung 4 Das Datenmodell.....	9
Abbildung 5 Generische Navigationsleiste	12
Abbildung 6 Generischer Footer	12
Abbildung 7 Decorator für das Login	12
Abbildung 8 Beispiel, Erweiterung des Datenmodells um weitere Fittings	14
Abbildung 9 Startseite.....	15
Abbildung 10 Fahrrad Seite	15
Abbildung 11 Messen Seite	15
Abbildung 12 Eingabe Seite	16
Abbildung 13 Ergebnis Seite	16
Abbildung 14 Inhalt pytest.ini	16

11. Tabellenverzeichnis

Tabelle 1 Qualitätsziele der Bike-Fitting-App.....	5
Tabelle 2 Lösung Strategie für Qualitätsziele.....	6
Tabelle 3 MVC Aufteilung auf Module.....	7
Tabelle 4 Mögliche Subsysteme	8
Tabelle 5 Wichtige Methoden des Datenmodells	10
Tabelle 6 Faktoren für die Berechnung.....	10
Tabelle 7 Wichtige Methoden der Views	11
Tabelle 8 wichtige Templates	11
Tabelle 9 Wichtige Methoden Account-System	12
Tabelle 10 Risiko gegenüber Methodik	17

12. Literaturverzeichnis

- Patidar, B. (n.d.). *CODEDEC*. Retrieved from <https://codedec.com/tutorials/how-to-create-login-and-registration-in-django/>
- Seifert, E. (Sommer Semester 2023). *Agile Webanwendungen mit Python (Moodle)*. Retrieved from <https://smt.sytes.net/fha/2023ss/pyweb/skript/index.html>