Einführung in die Rechnerarchitektur Großpraktikum Parallele Berechnung der Mandelbrotmenge

Erste Ausarbeitung

Florian Lercher[1], Tobias Klausen[2],
Niels Mündler[3], Maximilian Frühauf[4]

[1] florian.lercher@tum.de
[2] tobias.klausen@tum.de
[3] n.muendler@tum.de
[4] max.fruehauf@tum.de

1 Fachliche Spezifikation

1.1 Funktionale Anforderungen

Es soll ein Programm zur Berechnung und Darstellung der Mandelbrotmenge entwickelt werden. Hierbei soll der Nutzer die Möglichkeit haben, in beliebige Bildausschnitte des angezeigten Fraktals hinein und hinaus zu zoomen, sowie den gewählten Ausschnitt zu verschieben. Dabei soll die Berechnung der Menge auf einem Server zentralisiert werden (“Backend”), während die Darstellung im Webbrowser geschieht (“Frontend”). Die Berechnung im Backend soll parallel erfolgen und wird hierzu in Tiles (Bildausschnitte) aufgeteilt.

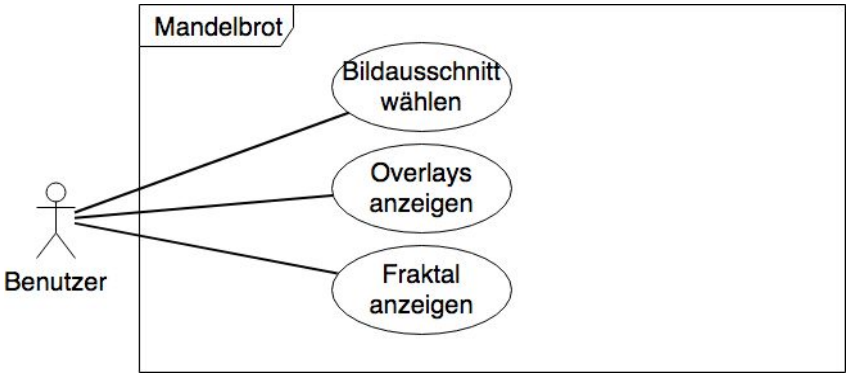

Abb. 1: Use Cases der Basisversion

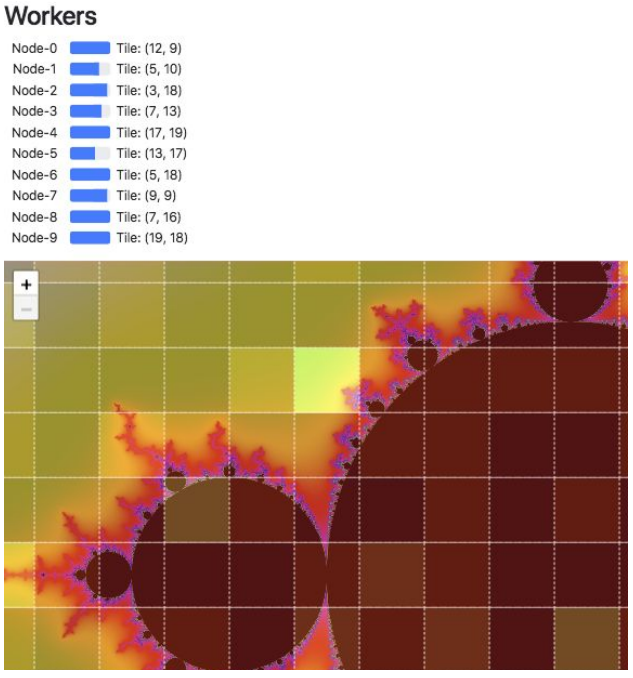

Abb. 2: Erster Prototyp der Benutzeroberfläche

1.1.1 Use Case: Fraktal anzeigen

Name	Fraktal anzeigen
Aktoren	Benutzer
Eintritts-bedingungen	Die Website ist geladen, d.h. die Startseite oder ein zuvor berechnetes Fraktal wird angezeigt.
Austritts-bedingungen	Das errechnete Fraktal wird in der vordefinierten Ausgangsposition angezeigt.
Event-behandlung	Der Nutzer drückt den Knopf “Ausgangsposition”. Daraufhin wird das Fraktal berechnet.

1.1.2 Use Case: Bildausschnitt und Zoomfaktor wählen

Name	Bildausschnitt und Zoomfaktor wählen
Aktoren	Benutzer
Eintritts-bedingungen	Das Fraktal wird bereits angezeigt.
Austritts-bedingungen	Der gewählte Ausschnitt des Fraktals wird angezeigt.
Event-behandlung	Der Benutzer kann mit gedrückter linker Maustaste den Bildausschnitt des Fraktals verschieben. Außerdem ist es möglich mit dem Mausrad in das Fraktal zu zoomen. Daraufhin wird der neue Ausschnitt berechnet.

1.1.3 Use Case: Overlays anzeigen

Name	Overlays anzeigen
Aktoren	Benutzer
Eintritts-bedingungen	Das Fraktal wird bereits angezeigt.
Austritts-bedingungen	Es wird ein Overlay über dem Fraktal angezeigt, welches dem Benutzer visualisiert, welcher Rechenknoten den Bildausschnitt berechnet hat. Es werden auch Information zur Rechenzeit angezeigt.
Event-behandlung	Der Benutzer drückt den Knopf “Overlay einblenden”. Daraufhin wird durch ein Overlay dargestellt, welcher Knoten welchen Bildausschnitt berechnet hat und wie lange dieser dafür benötigt hat. Durch erneutes Drücken des Knopfes, wird das Overlay wieder ausgeblendet.

1.2 Funktionale Anforderungen - Erweiterte Version

Über die Basisversion hinaus ist geplant das Projekt im Wintersemester 2018 um weitere Features zu erweitern. Das Ziel der meisten dieser Erweiterungen ist es dabei, die Interaktion mit dem Benutzer zu verbessern. Zudem ist geplant, die Performance sowohl des Back- und Frontends weiter zu verbessern.

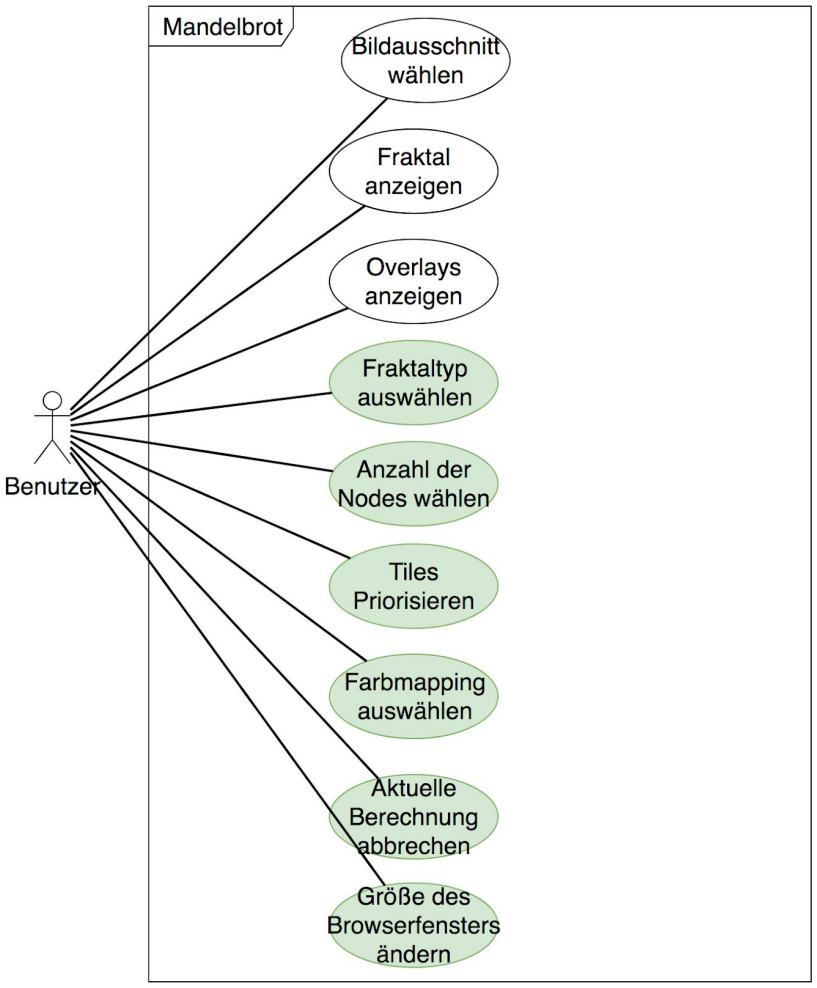

Abb. 3: Use Cases der erweiterten Version

1.2.1 Use Case: Fraktaltyp auswählen

Name	Fraktaltyp auswählen
Aktoren	Benutzer
Eintritts-bedingungen	Die Website ist geladen.
Austritts-bedingungen	Das gewünschte Fraktal wird in der vordefinierten Ausgangsposition angezeigt.
Event-behandlung	Der Benutzer wählt aus dem Drop-Down Menü das gewünschte Fraktal aus und drückt anschließend den Knopf “Berechnen”. Das ausgewählte Fraktal wird berechnet. Zur Auswahl stehen mindestens die Mandelbrotmenge und die Juliamenge[5].

[5] https://de.wikipedia.org/wiki/Julia-Menge

1.2.2 Use Case: Anzahl der Nodes wählen

Name	Anzahl der Nodes wählen
Aktoren	Benutzer
Eintritts-bedingungen	Die Website ist geladen. Die Anzahl der maximal zur Verfügung stehenden Nodes ist vom Backend vorgegeben.
Austritts-bedingungen	Die Anzahl der Nodes, welche verwendet werden, um die parallele Berechnung der Mandelbrotmenge durchzuführen wird verändert.
Event-behandlung	Durch einen Slider kann der Benutzer interaktiv die Anzahl der Nodes einstellen, die im Backend verwendet werden, um die Tiles zu errechnen. Wenn die Anzahl aktiver Nodes verringert, wird mehr Zeit benötigt, um alle Tiles zu berechnen.

1.2.3 Use Case: Tiles priorisieren

Name	Tiles priorisieren
Aktoren	Benutzer
Eintritts-bedingungen	Die Berechnung des Fraktals läuft.
Austritts-bedingungen	Die Reihenfolge, in der die Tiles vom Backend berechnet werden, wird abhängig von der Mausposition des Nutzers angepasst. D.h. diese Ausschnitte werden zuerst angezeigt.
Event-behandlung	Der Nutzer schaltet zunächst per Klick auf den Knopf “Tiles priorisieren” diese Option ein. Wenn er nun mit der Maus über das Fraktal fährt, werden diejenigen Tiles, welche der Mausposition des Benutzers am nächsten, sind priorisiert. Somit werden die Tiles in kreisförmiger Reihenfolge um die Position des Mauszeigers angezeigt. Bewegt sich dieser, wird die beschriebene Priorisierung für alle Tiles, für die die Berechnung noch nicht begonnen hat wiederholt.

1.2.4 Use Case: Farbmapping auswählen

Name	Farbmapping auswählen
Aktoren	Benutzer
Eintritts-bedingungen	Das Fraktal wird bereits angezeigt.
Austritts-bedingungen	Das Fraktal wird mit dem gewählten Farbmapping angezeigt.

Event-behandlung	Der Benutzer wählt entweder eines der vorgefertigten Mappings aus einem Drop-Down Menü oder kann eine eigenes Mapping erstellen, indem er Iterationen Farbwerte zuweist. Ein benutzerdefiniertes Mapping kann gespeichert werden.

1.2.5 Use Case: Aktuelle Berechnung abbrechen

Name	Aktuelle Berechnung abbrechen
Aktoren	Benutzer
Eintritts-bedingungen	Das Fraktal wird aktuell berechnet.
Austritts-bedingungen	Die aktuelle Berechnung ist abgebrochen.
Event-behandlung	Der Benutzer drückt auf den Knopf “Berechnung abbrechen”. Daraufhin werden laufende Berechnungen werden abgebrochen.

1.2.6 Use Case: Größe des Browserfensters verändern

Name	Größe des Browserfensters verändern
Aktoren	Benutzer
Eintritts-bedingungen	Das Fraktal wird bereits angezeigt oder aktuell berechnet.
Austritts-bedingungen	Das Fraktal wird in der neuen Größe angezeigt.
Event-behandlung	Der Benutzer verändert die Größe des Browserfensters. Daraufhin wird das Fraktal in der neuen Auflösung komplett neu berechnet. Laufende Berechnungen werden abgebrochen.

1.3 Nicht-funktionale Anforderungen

Da die Basisversion vorrangig auf Funktionalität fokussiert ist, wird das System erst während der Erweiterungsphase hinsichtlich der nicht-funktionalen Anforderungen optimiert. In der Basisversion soll selbstverständlich trotzdem ein Mindestmaß dieser Anforderungen erfüllt sein.

1.3.1 Leistungsanforderungen

Eine zentrale Anforderung ist es, das Fraktal so schnell wie möglich zu berechnen. Dazu soll die zur Verfügung stehende Hardware so gut wie möglich ausgenutzt werden.
Anfragen von mehreren Benutzern werden parallel verarbeitet.

1.3.2 Qualitätsanforderungen

Qualitativ sind Anforderungen in verschiedenen Bereichen zu erfüllen.

Die Benutzeroberfläche soll so leicht und intuitiv wie möglich zu bedienen sein. Hierbei soll zudem darauf geachtet werden, dass alle Funktionen nur mit einer minimalen Anzahl an Mausklicks auszuführen sind und die Oberfläche nicht überladen wird.

Zudem soll das System robust gestaltet werden. Dies wird durch die Verwendung von Buttons, Drop-Down Menus und Slider gewährleistet, die die Möglichkeit der Eingabe von ungültigen Werten verhindern.

Es sind keine Sicherheitsfeatures (Benutzerauthentisierung, Verschlüsselung) geplant, da keine sensiblen Daten verarbeitet werden und die Anwendung nicht für jedermann über das Internet zugänglich ist.

1.3.3 Einschränkungen

Das Frontend soll in einem Webbrowser lauffähig sein, während das Backend parallel auf mehreren Raspberry Pi’s oder ähnlichen unabhängigen Kleincomputern oder Rechenkernen zum Einsatz kommen soll.

2 Technische Spezifikation / Architektur

Das Problem fordert drei wesentliche Bausteine:
Eine Benutzeroberfläche in Webtechnologie (“Frontend”), welches die Benutzerinteraktionen entgegennimmt und per HTTP an den Backend-Master schickt.
Der Backend-Master verwaltet die eingehenden Rechenaufträge und verteilt sie via Message Passing Interface (MPI) an die Backend-Slaves. Der Master liefert das Ergebnis der Berechnung an das Frontend zurück.
Die Backend-Slaves nehmen die zugewiesenen Rechenaufträge entgegen und führen die eigentlichen Berechnungen möglichst effizient aus.

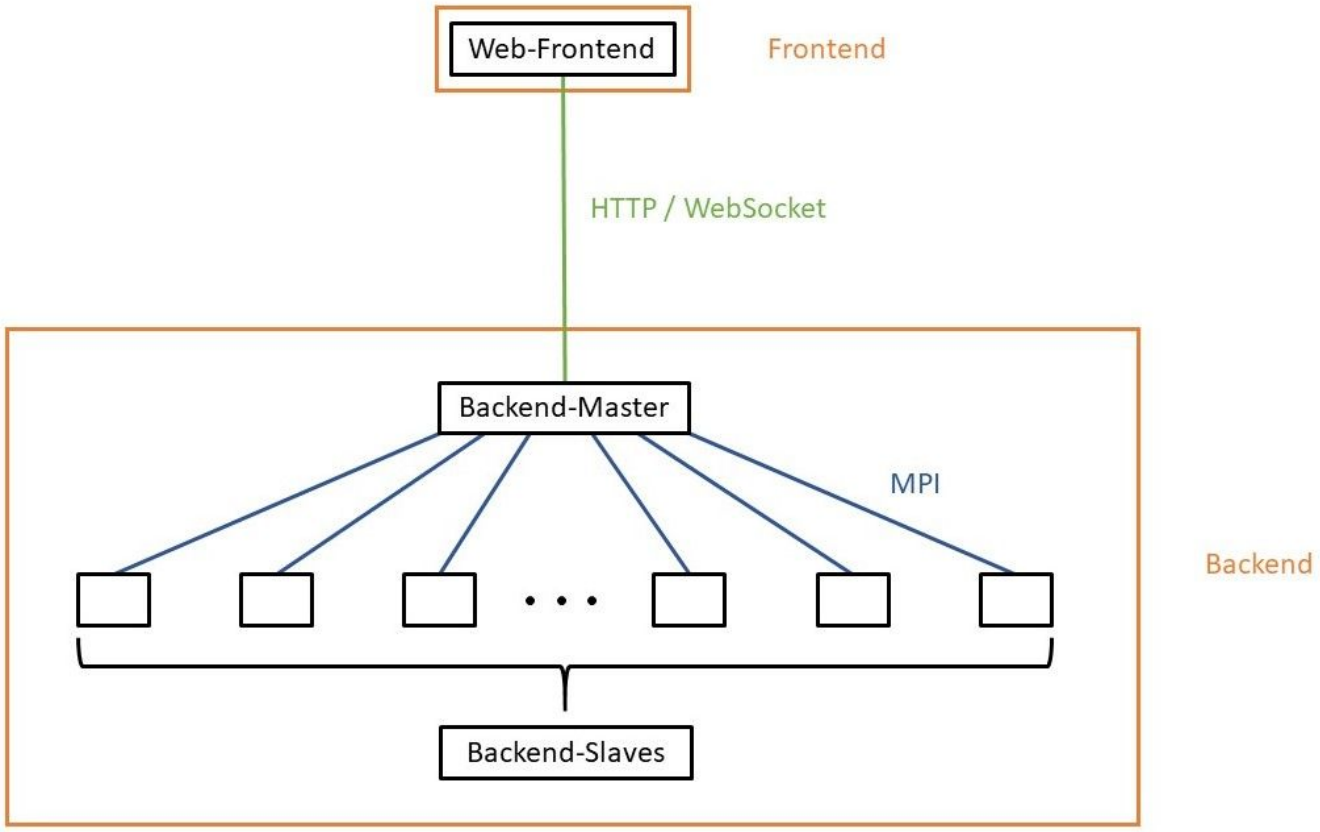

Abb. 4: Architekturübersicht

2.1 Frontend

2.1.1 Eingesetzte Technologien

Um das Frontend zu implementieren, wurde sich für die Programmiersprache JavaScript (ECMAScript 6) entschieden. Diese erlaubt es uns Berechnungen innerhalb des Browsers des Clients auszuführen und somit einen dedizierten Server für das Rendern von HTML zu umgehen.

Um dies zu realisieren verwenden wir das React[6] Framework, um mit JavaScript dynamisch HTML zu generieren. Dieses erlaubt ebenfalls für jede logische Komponente eine eigene JavaScript Klasse zu erstellen, welche dann den betreffenden HTML Code rendert. Somit kann eine Trennung der einzelnen Bereiche der Frontend Applikation erreicht werden, was die Wartbarkeit des Systems erhöht.

[6] https://reactjs.org/

Die Leaflet Library zerlegt eigenständig einen zu ladenden Bereich anhand von x- und y-Koordinaten in Ausschnitte gleicher Größe und fragt diese beim Backend an. Die einzelnen vom Backend bereitgestellten Tiles werden mit Hilfe der Leaflet[7] Library dargestellt. Diese bietet eine einfache API, um dynamisch Bildausschnitte zusammenzufügen, und abhängig von dem momentan Sichtbaren Ausschnitt des Benutzers Tiles nachzuladen. Somit ist es möglich eine einfache Zoom und verschiebe Funktionalität anzubieten.
Dabei lässt sich die Leaflet Library einfach in den generierten HTML Code einbinden und weiter für spezifische Features erweitern.

Damit eine konsistente Version aller der verwendeten Libraries über die Lebensdauer des Projekts gewährleistet werden kann, verwenden wir npm[8] als Versionskontrollsystem der verwendeten Libraries. Dieses ermöglicht es genaue Versionen der verwendeten Packages zu spezifizieren und bietet eine entwicklerfreundliche Kommandozeilenanwendung, um Packages zu aktualisieren oder sicherzustellen, dass alle benötigten installiert sind.

2.1.2 Funktionsweise

Das Frontend der Applikation ist im Kern als monolithische Applikation realisiert, welche Daten aus dem Backend empfängt und diese dem Benutzer anzeigt. Dabei sind die einzelnen logischen Komponenten, wie Worker und die Darstellung der Mandelbrot Menge auch im Code von einander getrennt.
Durch das Composition Pattern werden diese einzelnen Komponenten in einer Anwendung zusammengefügt. Falls es somit zu einer Änderung der anzuzeigenden Daten kommt, wird nur der entsprechende Bereich der Seite neu berechnet.
Da auch Eingaben des Benutzers zum Backend weitergeleitet werden müssen, unterhält das Frontend eine aktive Verbindung zum Backend, über welche diese übertragen werden.

Das Design der Weboberfläche soll schlicht, aber modern gehalten sein und sich auf die wesentliche Visualisierung der Mandelbrotmenge fokussieren.
Dabei ist, angedacht, wie in dem Screenshot in Abb. 1 eine Liste an den beteiligten Nodes der parallelen Berechnung darzustellen und darunter eine “heat map” der Dauer, die die entsprechende Tile zur Berechnung in Anspruch genommen hat.
Die Leaflet Library, welche zur Darstellung der Mandelbrotmenge im Browser verwendet wird, bietet eine einfache Möglichkeit an, Overlays über die “Karte” zu legen. Dies lässt sich nutzen um diese Informationen direkt im Bild anzuzeigen.
So könnte beispielsweise zusätzlich zum Namen des jeweiligen Kernes die durchschnittliche Auslastung der CPU oder des Speichers angezeigt werden. Außerdem ließe sich so aufzeigen, welche Tiles bereits im Cache vorhanden waren, und welche neu berechnet werden mussten.

2.2 Backend

2.2.1 Eingesetzte Technologien

Als Programmiersprache im Backend wird C++ verwendet. Zum einen bietet C++ hochsprachliche Konstrukte, wie die Möglichkeit Objekte in Klassen zu organisieren. Auch Vererbung und Polymorphie, zwei wichtige Konzepte der objektorientierten Programmierung, können genutzt werden, um die Wartung und Erweiterung des Systems zu erleichtern. Zum anderen ist C++ einen vergleichsweise maschinennahe und somit performante Sprache.

[7] http://leafletjs.com/
[8] https://www.npmjs.com/

Um die HTTP Anfragen des Frontends entgegenzunehmen, wird die Bibliothek C++ REST SDK[9] von Microsoft eingesetzt. Diese bietet alle nötigen Funktionen, die für die Realisierung eines HTTP-Servers nötig sind. Außerdem ist sie auch für verschiedene Plattformen verwendbar.

Die Kommunikation der verschiedenen Prozesse des Backends soll mittels MPI realisiert werden, da die Parallelisierung der Prozesse zur Berechnung des Fraktals direkt kontrolliert werden kann. Als konkrete Implementierung wurde sich für MPICH entschieden. Ein Grund dafür ist, dass MPICH stetig weiterentwickelt und verbessert wird, es also regelmäßig Veröffentlichungen mit neuen Features und Bugfixes gibt. Zudem ist diese Implementierung sehr gut dokumentiert, weit verbreitet und es existiert eine sehr aktive Community mit Blogs und Tutorials. Dies macht es für alle Beteiligten leicht, sich in die Materie einzulesen und kann in Zukunft für eine schnelle und zuverlässige Lösung von Problemen sorgen. Außerdem wird MPICH zusammen mit C++ auf einer Vielzahl von Systemen unterstützt, wozu laut Internetquellen auch der Raspberry Pi gehört. Um hierbei sicherzugehen muss dies zeitnah an einem Raspberry Pi Testgerät überprüft werden.

Um die Performance zu verbessern, kann in einer späteren Ausbaustufe Caching vorgesehen werden. Dies könnte entweder mittels eines Reverse-Proxy zwischen Frontend und Backend-Master oder durch Implementierung von Cache-Algorithmen direkt im Master realisiert werden.

2.2.2 Funktionsweise

Im Backend sind ein Master-Prozess und mehrere Slave-Prozesse vorgesehen. Die Aufgabe des Masters ist es, Anfragen des Frontends für einen bestimmten Ausschnitt des Fraktals (Tile) via HTTP entgegenzunehmen und die Ergebnisse anschließend zurückzuliefern. Für die Darstellung des Fraktals werden somit mehrere HTTP Anfragen gestellt.
Zudem hat der Master die Möglichkeit, eine Anfrage in mehrere kleine Rechenaufträge für die Slaves zu unterteilen. Damit ist die Flexibilität gegeben, die Anfragegröße bezüglich Latenz der Kommunikation und zur Verfügung stehender Rechenleistung zu optimieren.
Der Master reiht die Rechenaufträge in eine Warteschlange ein. Aus dieser Warteschlange werden die Slaves mit Rechenaufträgen versorgt. Dabei wird darauf geachtet, dass alle Slaves ausgelastet sind.
Für die Kommunikation zwischen Master und Slave wird MPI eingesetzt.
Hat der Master eine Anfrage des Frontends in mehrere Rechenaufträge zerlegt, werden die Ergebnisse gesammelt an das Frontend als HTTP Response zurückgegeben.

Dieses Verfahren funktioniert auch für gleichzeitige Anfragen von mehreren Frontends.

2.2.3 Schnittstelle zum Frontend

Der Master erwartet vom Frontend in einem HTTP Request folgende Daten:

- Kennung des zu berechnenden Fraktals
- Gesamtgröße des Gesamtbildes
- Koordinaten des Bildausschnitts
- Zoomstufe

Als Ergebnis der Berechnung sendet der Master folgende Daten via HTTP Response:

- Array mit den Ergebnissen der Berechnungen als Rohdaten
- Koordinaten des Bildausschnitts
- Zusatzinformationen zur Anzeige im Overlay (z.B.: Nummer des Slaves)

[9] https://github.com/Microsoft/cpprestsdk

2.3 Kommunikation

Im Folgenden noch einmal das Zusammenspiel der einzelnen Komponenten im Überblick als Sequenzdiagramm:

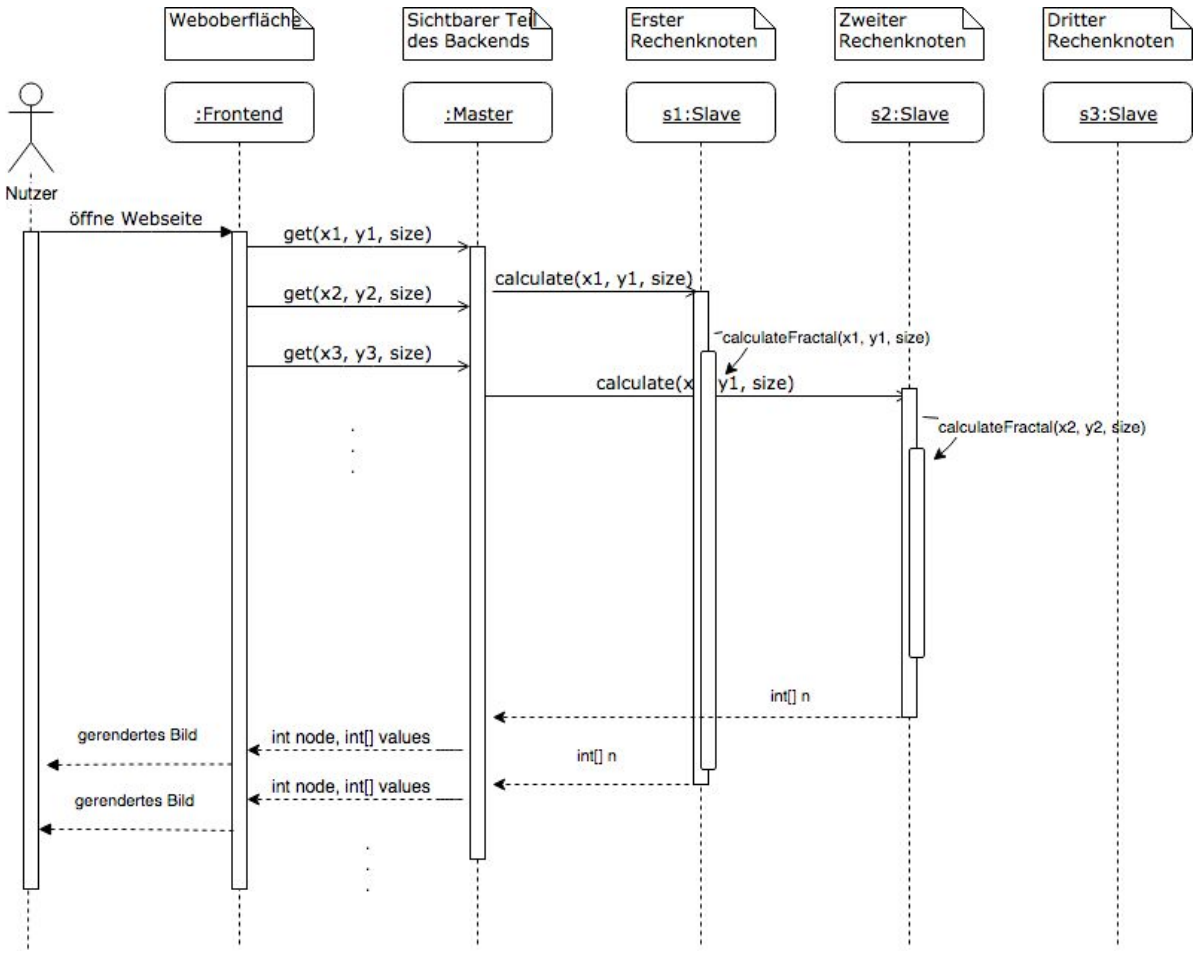

Abb 5: Ein Beispielhafter Aufruf des Webinterfaces. Hierbei ist zufällig Rechenknoten 1 langsamer, was jedoch das System nicht blockiert.

Die Trennung der verschiedenen Komponenten erfordert eine Kommunikationsebene zwischen dem Back- und Frontend der Anwendung. Somit muss ein gemeinsames Interface (HTTP) spezifiziert werden, welches diese ermöglicht.

Um die berechneten Teilbereiche des Fraktals zwischen den Systemen zu transportieren werden HTTP-Get-Requests verwendet, da primär Daten von Back- zu Frontend übertragen werden. Backendseitig wird dazu die C++ REST SDK[10] eingebunden, die asynchrones Hosten von Webservicen ermöglicht. Sie beantwortet Anfragen, die vom Frontend aus über die nativen JavaScript Funktionen gestellt werden (siehe z.B. fetch[11]). Dazu ruft das Backend über MPI die eigentlichen Rechenfunktionen auf den Slaves auf.

[10] https://github.com/Microsoft/cpprestsdk
[11] https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

2.4 Zielplattformen

Während der Entwicklung wird der C++ Code des Backends in einer Dockerumgebung[12] mit Linux kompiliert und ausgeführt. Dies ermöglicht ein problemloses Arbeiten über mehrere Systeme mit unterschiedlichen Betriebssystemen und Entwicklungsumgebungen.
In der finalen Version soll das Backend auf einem Raspberry Pi oder ähnlichen unabhängigen Kleincomputern lauffähig sein. Zudem kann es durch Docker auch auf anderen Zielplattformen problemlos laufen. Allerdings muss hierbei bedacht werden, dass dies die zu erwartende Leistung senkt.

[12] https://www.docker.com/

3 Teamkommunikation

Die Kommunikation zwischen den einzelnen Teammitgliedern haben wir, abhängig vom Kontext des Gesprächs auf zwei Onlineplattformen aufgespalten. Einerseits verwenden wir WhatsApp für eine schnelle und informelle Kommunikation, welche sich um grundlegende Konzepte oder Organisatorisches dreht. Andererseits verwenden wir GitLab für detaillierte Gespräche zu dem verwendeten Code oder der Diskussion unterschiedlicher Lösungsmöglichkeiten für ein aufgetretenes Problem.

Dabei wird für jedes auftretende Problem oder eine vorgeschlagene Erweiterung von dem entsprechenden Teammitglied in GitLab ein Issue erstellt, welcher das Problem beschreibt und optional ein paar mögliche Lösungen aufführt.
Dieser wird dann von einem Teammitglied auf einem eigenen Branch vom master mit dem gleichen Namen wie der Issue bearbeitet. Sobald eine Lösung gefunden wurde oder weiterer Diskussionsbedarf besteht, wird dies entweder weiter in dem zugehörigen Issue besprochen oder ein Merge Request mit dem Master erstellt.
Dann ist angedacht, dass ein anderes Mitglied sich den fertigen Code und den zugehörigen Issue durchliest. Dieses gibt dann Feedback zu der implementierten Lösung und womit eine hohe Codequalität innerhalb des Repositories gesichert wird. Sobald alle Verbesserungsvorschläge eingearbeitet sind, wird das Merge Request geschlossen und in den master Branch eingefasst.

Außerhalb der online Kommunikation, treffen wir uns ca. einmal pro Woche persönlich. Diese Treffen haben meist weder eine festgelegte Länge, noch vordefinierte Themen. Der Zweck ist es, aufgekommene Fragen zu diskutieren insbesondere wenn persönliche Besprechung sinnvoll ist um Details zur Implementierung oder Modellideen auszutauschen.

Zur gemeinsamen Arbeit an Textdokumenten wie dieser Spezifikation, wird Google Docs verwendet. Google Docs bietet ein gemeinsames Dokument, in dem alle Änderungen von allen Teilnehmern durch den integrierten Versionsverlauf gut nachvollziehbar sind. Kontroverse Stellen und Änderungsvorschläge können durch die Kommentarfunktion direkt im Dokument diskutiert werden.

4 Ablauf des Projekts

Um eine gewisse Spezialisierung auf Teilgebiete in diesem recht breite Ansprüche stellenden Projekt wurde eine Aufgabenteilung vorgenommen.

Tobias Klausen und Florian Lercher beschäftigen sich mit der Umsetzung des Backends und setzen sich dazu verstärkt mit C++ und MPI auseinander. Maximilian Frühauf kümmert sich unterdessen um die Entwicklung der Weboberfläche. Zudem stellt er das Grundgerüst für die plattformunabhängige Entwicklung des Backends auf. Niels Mündler kümmert sich um das Zusammenspiel von Front- und Backend und ist für die funktionierende Kommunikation der Systeme verantwortlich.

Wir gestalten das Projekt überwiegend agil mit folgendem Projektplan:

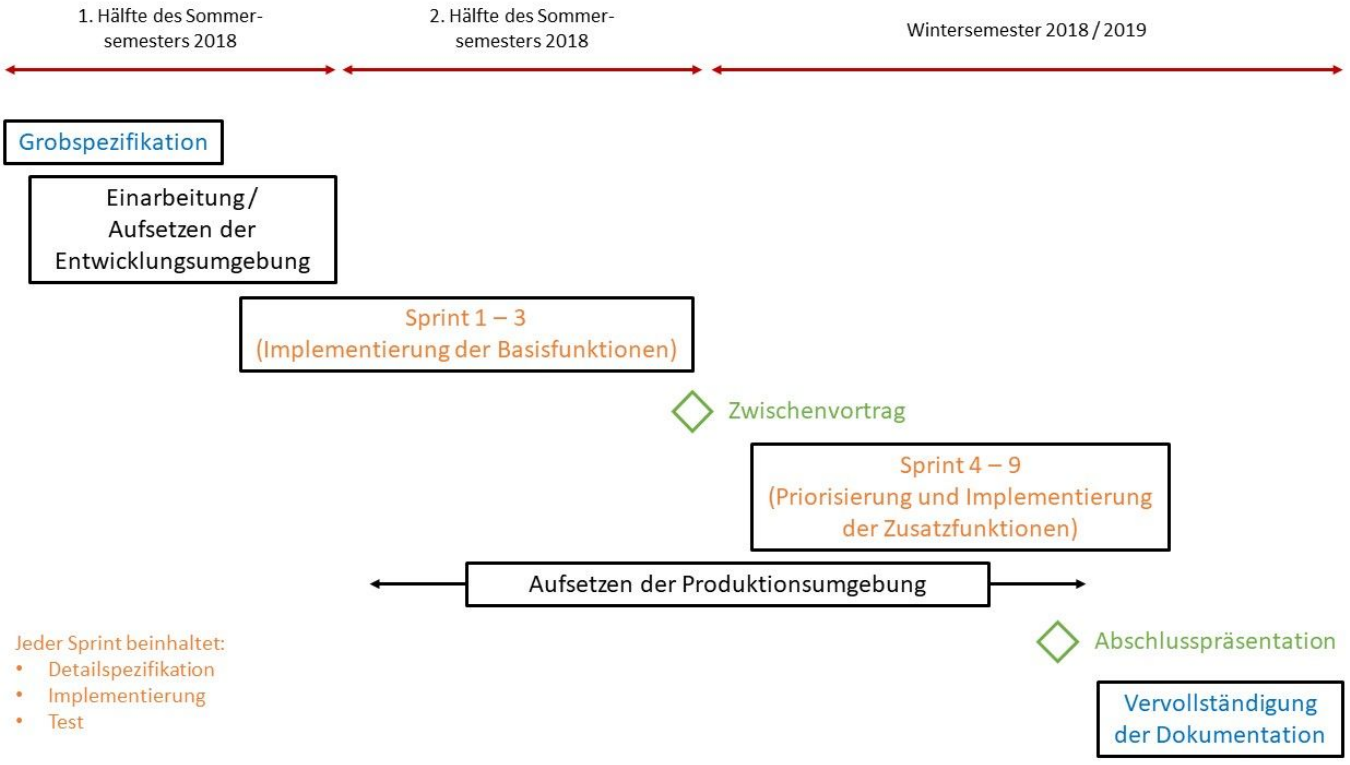

Abb. 6: Projektplan