Clients stets die gleiche Antwort zurück, egal ob dieser zum ersten, oder wiederholten Mal angefragt wurde. Des Weiteren hängt die Antwort nicht vom Client ab. Diese Entkopplung zwischen den Komponenten ermöglicht, dass die Aufgabe des Server, sowie des Clients durch mehrere Computer verrichtet werden kann und somit das System skalierbar ist (Fielding 2000, S. 79).

Der Hauptunterschied zwischen der REST-Architektur und anderen Stilen liegt jedoch in der genauen Bestimmung der zu verwendeten Kommunikationsschnittstellen. So bestimmt die REST-Architektur sehr explizit, welche From zur Kommunikation verwendet werden darf. Anders als andere Stile beruht der Aufruf von Methodiken nicht auf individuelle Funktionalität, sondern auf dem HTTP-Standard. Konkret bedeutet dies, dass die einzelnen Dienste des Servers sich an die HTTP-Optionen (GET, PUT, POST und DELETE) richten und keinen eigenen verwenden (vlg. Starke 2015, S. 128). Somit baut die REST-Architektur auf ein Kommunikationsstandard auf, der sich im Internet etabliert hat.

Auf Grundlade der standardisierten Kommunikation können zwischen Server und Client intelligente Zwischenstationen geschalten werden, die dazu zuständig sind häufig vorkommende Anfragen abzuspeichern (vlg. Fielding 2000, S. 79 f. Starke 2015, S. 128). Somit lässt sich eine Vielzahl von Serveranfragen im vornherein beantworten.

Die Antwort des Servers erfolgt durch Repräsentationen der Daten, wovon es für jede Ressource mehr Formate gibt. So kann eine Schnittstelle abhängig des angeforderten Mediums, sowohl JSON, als auch XML oder HTML zurück geben (vgl. Starke 2015, S. 128).

Verwendet wird die REST-Architektur ausschließlich für Anwendungen im Internet, da es auf die Anwendung des Hypertext Transfer Protokoll[13] (HTTP) angewiesen ist. Dabei findet der Architekturstil, sowohl Anwendung für ganze Systeme, als auch in komplexen Anwendungen mit einer Vielzahl ein einzelnen Services.

2.1.5 Monolithe Architektur

Der Begriff *„Monolith“* leitet sich vom altgriechischen *„monólithos“* ab und bedeutet *„aus einem Stein“* (vlg. *Duden Monolith* 2020; vgl. *DWDS – Digitales Wörterbuch der deutschen Sprache* 2020). In der Gesteinskunde wird da mit ein natürlich entstandener Gesteinsblock bezeichnet, der komplett aus einer Gesteinsart besteht (vgl. *DWDS – Digitales Wörterbuch der deutschen Sprache* 2020).

Nach Rod Stephens liegt eine monolithische Softwarearchitektur vor, wenn jegliche Funktionalität des Systems miteinander verbunden ist. Dabei spricht er auf die Verbindung

[13] Weitere Informationen zum Hypertext Transfer Protokoll kann unter folgender Literatur gefunden werden (Leach u. a. 2020).

von Dateneingabe, Datenausgabe, Datenverarbeitung, sowie Fehlerhandhabung und Benutzeroberflächen (vgl. Stephens 2015, S. 94).

Anders sieht es Sam Newman. Ihm nach liegt ein Monolites System schon vor, wenn die gesamte Funktionalität eines Systems gemeinsam über ein Deployment-Prozess bereitgestellt wird (vgl. Newman 2019, Kap. 2.2). Somit muss nicht zwingend jegliche Logik miteinander verbunden sein. Er unterteilt Monolithe Systeme in drei Kategorien: Einzelprozess Monolithe, Modulare Monolithe und verteilte Monolithe (vgl. Newman 2019, Kap. 2.2).

Der Einzelprozess Monolith ist die gängigste Form und deckt sich mit der Definition von Rod Stephens. Somit handelt es sich dabei, um ein System bei dem das gesamte System ein Prozess abbildet. Dies bedeutet, dass jegliche Funktionalität auf einander aufbauend ist und nur eine Datenspeicherung für die gesamte Anwendung verwendet wird (vgl. Newman 2019, Kap. 2.2.1). Anders ist dies beim Modularen System. Dieses zeichnet sich darin aus, dass die Funktionalität in einzelne Module geteilt wird und sogar einzelne Module eine separate Datenspeicherung besitzen können (vgl. Newman 2019, Kap. 2.2.2). Diese Form von monolithen System ist jedoch seltener und wird nur von einzelnen Unternehmen eingesetzt. Im Gegensatz zu verteilten Systemen sind die einzelnen Komponenten nicht auf separaten Computern verteilt und werden durch einen Deployment-Prozess online gestellt. Des Weiteren sind die einzelnen Module nur leicht entkoppelt, so kann es immer noch Abhängigkeiten geben (vgl. Newman 2019, Kap. 2.2.2). Unterschiedlich davon sind verteilte Monolithe. Diese sind komplett entkoppelt und kommunizieren nur noch über definierte Schnittstellen (vlg. Starke 2015, S. 116). Sie erfüllen somit jegliche Anforderungen an ein verteiltes System, sind jedoch in einem einzigen Bereitstellungsprozess gebündelt. Diese From wird jedoch kaum verwendet, da sowohl Nachteile auf Grund der Verteilung, als auch durch das gemeinsame Bereitstellen, entstehen.

Weder Stephens, als auch Newman geben Vorgaben hinsichtlich der Gliederung innerhalb eines Monolithischen Systems. Demnach kann eine Model-View-Controller Ansatz als Monolithisches System gelten, solange es einheitlich deployed wird. Anders ist es mit einem verteilten System, da nach Definition ein Monolithen System kein verteiltes System sein kann.

Im Rahmen dieser Arbeit wird bei jeglichen weiteren Referieren auf den Begriff Monolithen System stets von einem Einzelprozess Monolithen ausgegangen, außer es wird expliziert von einem Modularen, oder verteilten Monolithen geschrieben. Dadurch sollen beide Definitionen berücksichtig werden.

Im Vergleich zu einem verteilten System gibt es einige Vor- als auch Nachteile (vgl. Newman 2019, Kap. 2.2.4 und Kap. 2.2.5). So ist das bereitstellen eines Monolithen System einfacher, da es ein Bereitstellungsprozess für die gesamte Anwendung gibt. Wiederum führt

dies dazu, dass der Prozess deutlich länger dauert. Diese Tatsache ist insbesondere gravierend, wenn vermehrt kleine Änderungen vorgenommen werden. Anderseits vereinfacht eine Anwendung, die als ein Prozess zu sehen ist, die Fehlersuche und ermöglicht es Funktionen mehrfach zu verwenden. So lassen sich Funktionen und Klassen in einem Einzelprozess Monolithen mehrfach verwenden und schneller neue Funktionen umsetzen. Jedoch verursacht dies, dass schnell Abhängigkeiten entstehen können und Änderungen ungewollte Fehler verursachen. Dadurch wird die Umsetzung von neuen Funktionen mit steigender Codemenge verlangsamt und der Einstieg von neuen Teammitgliedern erschwert.

Bei größeren Unternehmen mit mehreren Team kommt hinzu, dass es leicht zu Konflikten kommen kann, da alle auf die gleiche Codebase zugreifen. So führt ein Monolithen System dazu, dass bei vielen Entwickler viele Absprachen nötig sind und es zu Problemen bei der Zusammenführung von Funktionen kommen kann (vgl. Newman 2019, Kap. 2.2.4). Anders ist es beim Erstellen von System übergreifenden Test. Diese werden durch ein Monolithen System begünstigt und können im Vergleich zu einem verteilten System einfacher umgesetzt werden (vgl. Newman 2019, Kap. 2.2.5).

2.1.6 Einordnung des aktuellen Systems von PluraPolit

Nachdem die einzelnen Softwarearchitekturen vorgestellt wurden, wird nun das System von PluraPolit genauer betrachtet. Hierfür wird zu Beginn die von PluraPolit verwendete Technologie detaillierter beschrieben und einzelne Charakteristiken festgehalten. Anhand dieser Merkmale soll abschließend eine Einteilung für das System getroffen werden.

Wie aus der Einleitung hervorgeht, bietet PluraPolit eine Plattform an, die Jung- und Erstwähler bei der Politischen Bildung hilft. Dafür wurde ein Softwaresystem entwickelt, in welches sich zum einen leicht neuer Content einpflegen lässt und zum anderen die hinterlegten Tonaufnahmen den Benutzern darstellt. Um diese zu erreichen wurde neben der eigentlichen Plattform eine Content Management System (CMS) konzipiert. Umgesetzt wurde dies in einer Ruby on Rails Anwendung mit Zugriffsbeschränkung durch Passwortabfrage.

Ruby on Rails, kurz auch Rails bezeichnet, ist ein quellenoffenes Webframework, welches für die Programmiersprache Ruby entwickelt wurde. Es ist komplett Open Source und wird von einer aktiven Community entwickelt. Es gibt eine Vielzahl von kostenfreien Paketen, sogenannte Gems, die bei Belieben zu einem Projekt hinzugefügt werden können. Im Vergleich zu anderen Frameworks zeichnet sich Rails besonders durch seine Implementierung der REST-Architektur aus. Diese Implementierung führt jedoch dazu, dass eine Vielzahl an Bedingungen an die Entwicklung und Implementierung von Rails Anwendungen gestellt werden. Getreu nach dem Motto “Konventionen vor Konfigurationen” nutz Rails die Bestimmungen als Vorteil und integriert ein System, indem externe Pakete ohne

Konfigurationsaufwand hinzugefügt werden kann. Dies wird erreicht, indem durch Regeln festgelegt wird, welche Dateien für welche Funktionalitäten verantwortlich sind und wie diese Dateien zusammenarbeiten.

Dabei orientiert sich Rails bei der Teilung der Dateien an dem Model-View-Controller-Ansatz.

Für das Einpflegen von neuen Tonaufnahmen bedeutet dies konkret, dass anhand der Eingabe der Url der für das verwalten von Tonaufnahmen zuständige Controller die gewünschte Darstellung anzeigt. Darauf hin kann nun die Aufnahme als Datei hinterlegt werden und hochgeladen werden. Dies geschieht, indem per Klick im CMS ein HTTP-Anfrage mit der geladenen Datei an den Controller geschickt wird. Dieser kümmert sich anschließend darum, dass die Datei über das Model in der Datenbank gespeichert wird und gibt die Bestätigung der Aktion im View wieder.

Dabei wird die Tondatei selbst nicht in der Datenbank gespeichert. Vielmehr wird der Speicherservice von Amazon Web Services (S3) genutzt. Somit sendet, nachdem die Datei im View hinterlegt wurde, der Controller die Datei automatisch an S3 und erhält anschließend eine Url zurück. Diese wird anschließend abgespeichert und dient als Referenz zur eigentlichen Tonaufnahme (siehe Abbildung).

Neben dem Hochladen von Tonaufnahmen dient jedoch das CMS auch dazu bestehende Informationen zu bearbeiten und ggf. anzupassen. Im Detail bedeutet dies, dass die jeweilige Webseite per Url Aufruf angefragt wird, der Controller die angeforderten Daten über das Model aus der Datenbank lädt und die Entsprechende graphische Darstellung rendert. Dabei übergibt der Controller die Informationen an den View, der anschließend die Daten über Embedded RuBy in HTML integriert und anzeigt (siehe Abbildung).

Auch die Plattform ist in Ruby on Rails implementiert. Genauer beschrieben handelt es sich bei der Plattform um die selbe Applikation wie das CMS nur das die graphische Darstellung durch das JavaScript Framework React übernommen wird. Im Detail bedeutet dies, dass beim Aufruf der Plattform eine Anfrage an den Controller geschickt wird und dieser anschließend die kompilierte React Anwendung ausgibt. Diese lägt eigenständig jegliche Informationen über HTTP Anfragen und kümmert sich um interne Seitenaufrufe. Die Anfragen werden dabei asynchrone an die Rails Applikation geschickt und vom Controller beantwortet. Somit erhält die React Anwendung über die Kommunikation von Controller und Module, den Content aus der Datenbank, der vorher eingepflegt wurde (Siehe Abbildung).

Demnach gibt es eine Aufteilung hinsichtlich der graphischen Darstellung in Content Management System und Plattform. Die Datenspeicherung und Verwaltung ist jedoch gleich. Auch wird die gesamte Codebase in einem Bereitstellungsprozesses dem Hosting

Service bereit gestellt. Somit handelt es sich beim System von PluraPolit um ein Monolith, welches teilweise Modular ist, die REST-Standards erfüllt und nach dem Model-View-Controller-Ansatz aufgeteilt ist. Des Weiteren nutz es vereinzelt externe Services von AWS, um Tondateien zu speichern und Transcripierungen vorzunehmen.

2.2 Microservices

Der Begriff Microservices wird mehrdeutig verwendet. So wird je nach Perspektive entweder eine Softwarearchitektur oder eine Komponente einer solchen Architektur beschrieben. Eng verbunden mit diesem Begriff sind auch dynamische Systeme und Konzeptionierung von Unternehmensstrukturen. Demnach sieht Eberhard Wolf unter Microservices ein Modellierungskonzept, welches dazu dient größere Softwaresysteme in kleinere Einheiten zu teilen (vgl. Wolff 2018, Kap. 1.1). Dabei hat die Aufteilung Auswirkungen auf die Organisation als auch auf die Entwicklungsprozesse.

Nach Sam Newman ist ein Microservice ein *„eigenständige ausführbare Softwarekomponente, die innerhalb eines Anwendungssystems mit anderen Softwarekomponenten kollaboriert“* (Newman 2019, Kap. 2.1). Sie zeichnet sich durch das kommunizieren über definierte Netzwerkschnittstellen aus und formt in Vereinigungen eine Microservices-Architektur. Ein Microservice umfasst dabei die Datenspeicherung, Datenverarbeitung und Datendarstellung und besitzt eine gut definierte Benutzeroberfläche (vgl. Newman 2019, Kap. 2.1).

Ergänzend dazu schreibt Wolf, dass sich das Konzept der Microservices-Architektur aus der Philosophie vom Unix Betriebssystem ableitet, welches nach Peter H. Salus folgende drei Leitpunkte umfasst (Salus 1994; vgl. Wolff 2018, Kap. 1.1):

- Schreibe Programme, sodass sie nur eine Aufgabe erledigen und diese gut.
- Schreibe Programme, die zusammen arbeiten.
- Schreibe Programme, welche über definierte Schnittstellen (Textstream) kommunizieren.

Nach James Lewis sind Microservices kleine Anwendungen, die unabhängig bereitgestellt, getestet und skaliert werden. Ebenfalls wie Wolf beschreibt er diese Programme, als einfach zu verstehen, die nur eine Aufgabe übernehmen.

Zusammenfassend lässt sich sagen, dass der Begriff Microservices nicht einheitlich definiert ist und es sich zwei unterschiedliche Perspektiven ergeben. Zum einen beschreibt es ein Modellierungskonzept, welches Auswirkungen auf Unternehmenssturkur und Managemententscheidungen hat und zum anderen kennzeichnet es eine Software-Architektur, die in eigenständige Komponente geteilt ist.