Technische Projektdokumentation paintroom

Backend

Ursprünglich war ein PHP-Backend vorgesehen, welches über AJAX-Calls des Frontends alle auf der Leinwand vereinten graphischen Elemente mittels GD-Library reproduziert und in einem Bildelement vereint. Dieses würde dann auf dem Server gespeichert und mit einer Kennung versehen werden, welche an das Frontend zurückgegeben wird. Das Frontend öffnet dann einen neuen Tab mit dem Bild, welches dann heruntergeladen werden kann. Dateiformate können durch das Request des Clients frei gewählt werden, da die Konvertierung serverseitig in GD-Library stattfinden kann.
Im erweiterten Rahmen, welcher früh aus Gründen der Umsetzbarkeit gekürzt wurde, war vorgesehen, dass Nutzer über eine Raumkennung und eine serverseitig geführte virtuelle Leinwand gleichzeitig zusammen an einem Objekt arbeiten konnten. Dies wäre über eine „Eingabenliste“ gelaufen, in der dokumentiert wird, welcher Nutzer wann welche Änderung durchgeführt hat. Der Client würde dann in periodischen, kurzen Abständen die Änderungen des Servers abfragen und bei sich umsetzen, bis auf allen Clients alle Änderungen vorhanden sind, unter Berücksichtigung von Zeitverzögerungen bei der Kommunikation.

Aufgrund von Entwicklungsproblemen im Frontend, welches priorisiert wurde, kam die Entwicklung der im Projektantrag und in der Projektdokumentation beschriebenen Backend-Funktionalität nicht zustande.

Frontend

Die erste Hälfte dieses Artikels soll die Entwicklung beschreiben, während die zweite auf das Endergebnis eingeht.
Zur Erstellung des Projektantrages und Beginn des Projekt sah der Plan folgendermaßen aus: die Programmierung des Frontends läuft über objektorientiertes Javascript, unter Einbindung und Verwendung von paint.js und JQuery. Ein paintroom-Objekt wird mit EventHandlern ausgestattet, um auf Benutzereingaben zu reagieren, und über AJAX-Calls in JQuery wird Kommunikation zwischen dem paintroom-Objekt des Clients und der Serverapplikation ermöglicht. Alle Änderungen werden als JSON im LocalStorage des Browsers gespeichert und gemanagt.
Bei der Entwicklung dieses Objekts taten sich schnell Schwierigkeiten mit dem paint.js Tool auf. Die im Readme beschriebene Installation und Einbettung funktionierte nur oberflächlich, tatsächlich war das Projekt stark verändert, seit die Readme geschrieben wurde. Zum Ermitteln der tatsächlichen Funktionsweise war eine aufwändige Einarbeitung in den paint.js Quellcode erforderlich.
Als Ergebnis dieses Projektes verbleibt aufgrund der aufgetretenen Entwicklungshürden sowie unvorhergesehenen zeitlichen Umständen eine eingebettete paint.js Leinwand mit vier Buttons, „log paint“, „fire draw event“, „output canvas“, und „clear canvas“, welche hauptsächlich mit debug-funktionen bestückt sind, und konsolenausgaben erzeugen.
„log paint“ gibt eine Repräsentation des paint.js Objekts zum Zeitpunkt der Ausgabe aus.

„fire draw event“ fügt ein Line-Objekt auf der Leinwand ein. Dies testet sowohl die Methode, mit der Lines beschrieben werden, als auch **wo** Lines im paint.js Objekt gespeichert werden.
„output canvas“ gibt den Leinwandsinhalt auf der Konsole aus, welcher sowohl aus „userpaths“, also handgezogenenen Elementen, als auch „localdrawings“, welche mit anderen Werkzeugen gezogene Elemente sind, bestehen.
„clear canvas“ entfernt alle Inhalte der Leinwand.

Kompetenzen

Überprüft die für dieses Modul relevanten Funktionen der Applikation anhand eines Testplanes. Dabei testet er / sie mobile und stationäre Plattformen mit verschiedenen Browsern.
Kennt Tools zur Validierung von HTML und CSS und validiert seinen / ihren Code.
Aufgrund der geringen Abdeckung geplanter Funktionen existiert kein nennenswerter Testplan. Im Testprotokoll-Dokument ist ein Test für die Ausgabe der Buttons gegeben.

Kann Multimedia-Inhalte dynamisch für unterschiedliche Geräte, Betriebssysteme und Browser optimieren. Dabei unterstützt er / sie die wichtigsten Betriebssystem und Browser.
Die Projekt-Applikation läuft auf gängigen Browsern welche Javascript und das HTML5 <canvas>-Element unterstützen, u.A. Chrome, Firefox, IE9+, Edge, und Safari. Eine Konvertierung von Inhalten in unterschiedliche Formate findet nicht statt.

Zeigt die Multimedia-Inhalte auf unterschiedlichen Bildschirmgrössen korrekt an. (Responsive Web Design).
Die HTML-Page in der das Projekt eingebettet ist, ist dank Bootstrap in der Lage, sich auf nahezu beliebige Bildschirmgröße anzupassen. Jedoch ist eine Mindesthöhe der Leinwand auf 500px gesetzt.

Setzt aktiv ein Sourcecontrol System für die Entwicklung des Projektes ein.
Alle Inhalte des Projekts, inklusive dieses Dokuments, sind über ein Remote Git Repository auf Github verwaltet: https://github.com/Thunderstonjek/PaintRoom

Kann den Projektantrag in eine Projekt-Dokumentation überführen. In der Projekt-Dokumentation werden die für dieses Modul relevanten Funktionen der Applikation beschrieben und erklärt und die Verwendung der Dateiformate begründet.
Ist diesem Dokument zu entnehmen.

Erstellt Multimedia-Inhalte dynamisch mithilfe geigneter Geräten, Bibliotheken, APIs selbst oder ladet Multimedia-Inhalte über ein Formular in die Applikation. Beim Upload beachtet er / sie die Filegrösse und schrenkt mögliche Dateiformate ein.
Wenngleich das paint.js Tool eingebunden und in Verwendung ist, werden keine konkreten Multimedia-Dateien im Projekt verwendet.

Verabeitet die generierten oder hochgeladenen Multimedia-Inhalte weiter in dem er / sie die Inhalte dynamisch verändert, kombiniert oder analysiert.
Wenngleich auf dem Canvas Inhalte erzeugt und bearbeitet werden können, besteht keine Option, diese zu speichern oder sonstwie greifbar zu machen (abgesehen von Screenshots).

Achtet auf Usability der Webseite. Dazu bietet er / sie benutzerdefinierte Such- / Filterfunktionen wie Filter, Pagination oder Suche an.
Leitet den Benutzer mit sinnvollen Meldungen durch alle Prozesse.
Validiert HTML und CSS.
Es ist eine Mindesthöhe der Leinwand eingerichtet, um die in die Leinwand integrierten Schaltflächen und Werkzeuge, sowie die Leinwand selbst, benutzbar zu gestalten. Es sind keine Meldungen an den Nutzer außerhalb der Konsole vorgesehen. HTML und CSS werden über die in Visual Studio Code integrierten Plausibilitätsprüfen gecheckt, aber nicht extra weiter validiert.

Setzt für die Projektplanung ein geeignetes Planungstool ein und verwendet dieses aktiv während der Entwicklung des Projektes.
Obgleich anfänglich vorgesehen, wurde außerhalb des Lernprozess-Dokuments und handschriftlichen Notizen keine extensive Planung des Projekts geleistet.