Instagram

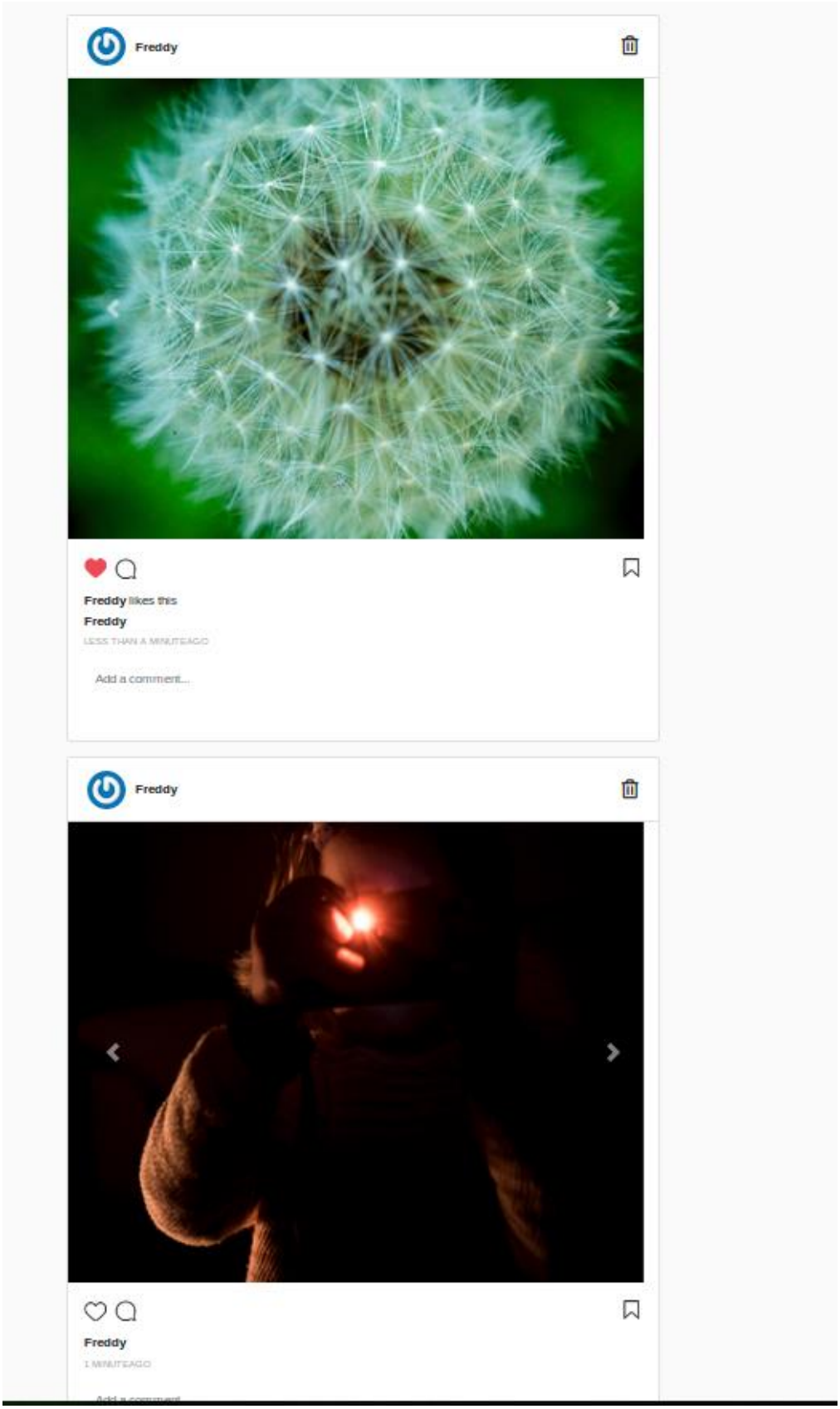
Dokumentation Arbeitsblatt 7

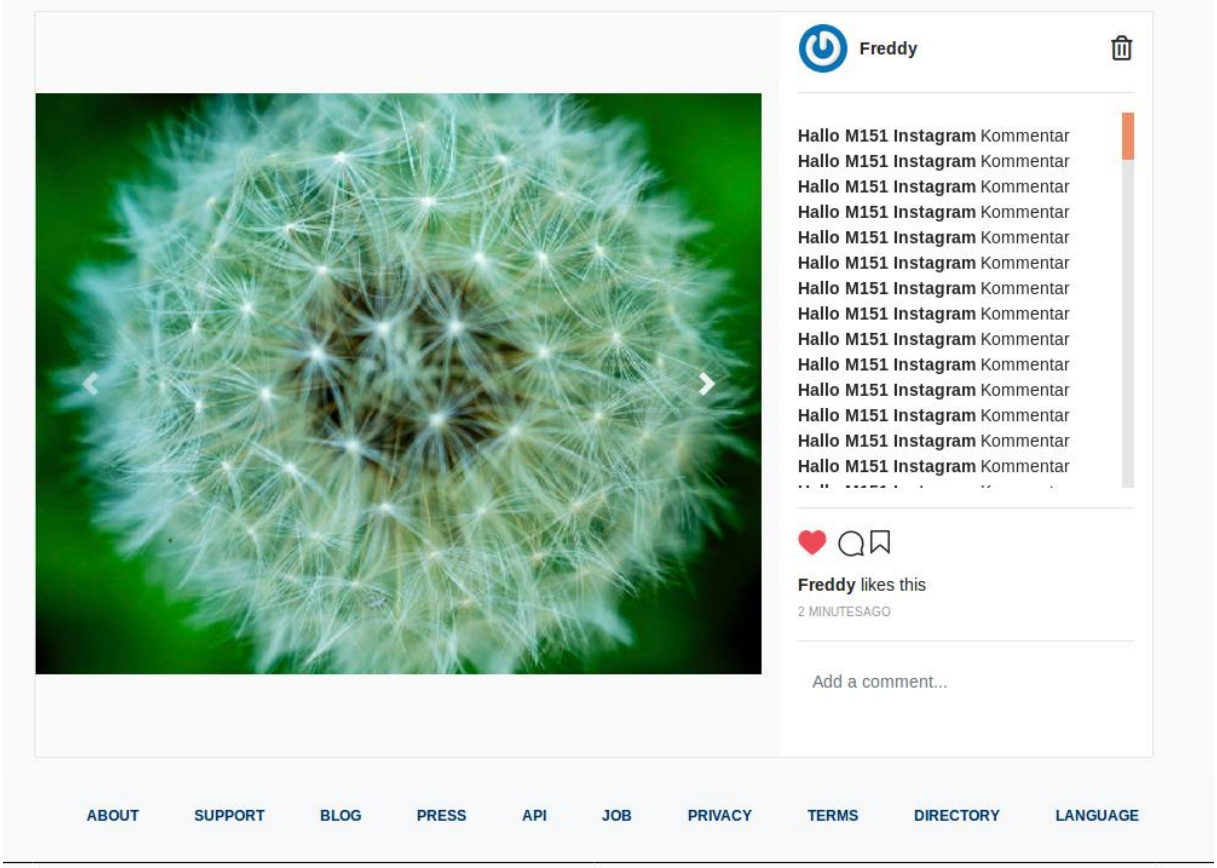
Inhalt

Zusammenfassung

Um den letzten Teil der Applikation zu vervollständigen, habe ich mit dem Post Detail angefangen. Das ist die Detail-Ansicht von einem Post. Als Erstes habe ich den Post-Controller angepasst. Danach habe ich die passende View angefertigt, welche in zwei Spalten unterteilt habe, um die Ansicht zu strukturieren. Da die Detailview nun erstellt war, musste ich sie in der Liste der Posts verlinken. In der ersten Spalte der Detailansicht wird bereits ein Bild angezeigt und nun habe ich noch die rechte Spalte mit den dazu vorhandenen Kommentaren ausgefüllt. Somit wird nun in der Detail-Ansicht eines Kommentars der Benutzer, welcher den Post erfasst hat, die Kommentare und die Likes dazu angezeigt. Auch kann eingesehen werden, wann der Post hochgeladen wurde.
Die Posts konnten noch nicht gelöscht werden, was ich nun umgesetzt habe. Erst habe ich eine Methode im Controller erstellt und danach einen Linkt zu der Methode in die Liste der Posts und in die Detailansicht gebaut. Somit kann ein Post über die Liste oder die Detail-Ansicht gelöscht werden.
Nur der Ersteller eines Posts soll diesen auch löschen können. Dazu habe ich das Model für die Tabelle post so angepasst, dass beim Löschen überprüft wird, ob der User, welcher den Post zu löschen versucht, auch wirklich der Autor des Posts ist.
Jeder Post kann von jedem beliebigen Benutzer geliked werden. Um das zu ermöglichen habe ich ein neues Model erstellt und das Model post mit dem neu erstellten Model verbunden. Auch mit der Tabelle user habe ich eine Verbindung erstellt, damit feststeht, wer den Kommentar geliked hat. Damit ein User einen Kommentar nur einmal liken kann, habe ich eine passende Funktion im Routing erstellt. Bis zu dem Zeitpunkt war aber nur die Grund Struktur erstellt. Ein User hatte noch nicht die Möglichkeit einen Post zu liken. Um das zu ermöglichen musste ich einen Controller für die Liking-Funktion erstellen und Links in die

Listen- und Detail-Ansichten von Posts einfügen. Damit die Likes richtig angezeigt werden, musste ich die Funktionen zum Anzeigen der Daten bei der Listen- und Detail-Ansicht anpassen. Für das Anzeigen der Likes und der Posts habe ich Partial-Views verwendet.
Die Navigationsbar musste ich auch an die neuen Funktionen anpassen. Und zu guter Letzt habe ich das Problem gelöst, dass ein Post nicht gelöscht werden kann, wenn dieser bereits geliked wurde. Das Problem ist aufgetreten, da noch keine Lösch-Constraints für die Tabelle mit den Likes erstellt wurden.
Unten ist die fertige Post-Liste dargestellt und darunter die Detail-Ansicht:


Freddy
Freddy likes this
Freddy
LESS THAN A MINUTE AGO
Add a comment...
Freddy
Freddy
1 MINUTE AGO
Add a comment...



Lösung der Aufgaben

Aufgabe 1, Seite 8

Diese Zeile gibt entweder die Daten vom Post zurück, sofern die ID des Benutzers mit der ID des Posts zusammengehören, oder sie gibt nichts zurück, was bedeutet, der Nutzer hat diesen Post nicht erstellt und kann ihn somit nicht löschen.

Aufgabe 2, Seite 9

Rails g scaffold Like user:references post:references

Aufgabe 3, Seite 12

Mit diesem Schlüsselwort werden nur die Routes erstellt, die man auch wirklich braucht.

Aufgabe 4, Seite 21

Das Problem kann gelöst werden, indem man einen Löschen-Constraint im Post Model hinzufügt. Man gibt mit «dependent: :delete_all» an, dass alle Likes mitgelöscht werden sollen, wenn ein Post gelöscht wird.

Selbstreflexion

In diesem Arbeitsblatt konnte ich nicht viel lernen, da der grösste Teil darin bestand, Code abzuschreiben. Somit konnte ich nicht viel Motivation dafür aufbringen und hatte Mühe den Sinn dahinter zu verstehen. Meinem Lernprozess in der Lehre wird damit nicht geholfen, wenn wir das Projekt unbedingt abschliessen müssen. Der einzige Teil, bei dem man etwas Übung bekommen kann, ist die Dokumentation. Das Problem am Ende des Arbeitsblattes habe ich sehr schnell gefunden, da wir das gleiche Thema bereits ein paar Arbeitsblätter davor schon angeschaut haben.

Abschliessende Reflexion über Projekt

Ich habe gelernt, wie man Cloudinary in Code einbindet und damit einen einfachen Bilderupload erstellen kann. Im Vergleich zu dem Bildupload, den ich bis jetzt ausprobiert habe, ist das eine mit Abstand einfachere Lösung. Auch das Arbeiten mit Gravatar hat einiges erleichtert. Gravatar nimmt einem einiges an Arbeit weg, da nicht noch ein Bildupload für das Profilbild erstellt werden muss.
Die Partial-Views sind eine übersichtliche und performante Lösung, Redundante Codeteile auszulagern. Ein Vorteil ist nicht nur, dass die Webseite dadurch schneller läuft, sondern auch, dass Änderungen nur an einem Punkt vorgenommen werden müssen und für alle Ansichten übernommen werden. Das erleichtert einem ebenfalls einiges an Arbeit.
Ich habe mich durch die Arbeitsblätter durchgearbeitet und bin dabei teilweise auf Probleme gestossen, die meinen Zeitplan um einiges verschoben haben. Ich musste somit einiges zu Hause nachholen. Lernen musste man für die Aufträge nicht viel, da man meistens nur Code abgeschrieben hat. Ich musste mich jedoch ausführlicher über den geschriebenen Code informieren, um die aufgetretenen Probleme beheben zu können.
Ein grosses Problem war, dass ich Probleme mit der Interneteinstellung der VM hatte und somit den Fortschritt nicht auf Github hochladen konnte. Nachdem mir der Lehrer geholfen hatte, hat sich die VM zurückgesetzt und ich habe meinen Fortschritt verloren. Glücklicherweise konnte ich die VM von einem Kollegen kopieren und musste somit die ganzen Arbeitsaufträge nicht erneut durchführen. Schlussendlich hat aber alles geklappt und die geforderten Funktionen und Designs sind implementiert.
Im Bereich, wo man die Zugangsdaten für Cloudinary verstecken musste, musste ich den einfacheren Weg wählen und habe die Zugangsdaten direkt, und somit offen, im File angegeben. Das musste ich so anwenden, da das bei mir mit dem Verstecken nicht funktioniert hat. Ansonsten habe ich mich immer strickt an die Vorgaben gehalten, was auch meistens auf Anhieb funktioniert hat.
Ich glaube nicht, dass es von Vorteil wäre, wenn ich an meiner Vorgehensweise in diesem Projekt etwas ändern würde. Da es recht einfach war, den Code abzuschreiben, und dieser immer auf Anhieb oder nach kurzer Fehlersuche funktioniert hat, bin ich nicht der Meinung, etwas ändern zu müssen. Ich konnte also nicht viel mitnehmen, wie ich ein Webprojekt am besten programmieren kann, da die Struktur des Moduls überhaupt nicht der Realität entspricht. Auch das Suchen der Fehler hat mehr Zeit gekostet als dass es den Lerneffekt wert gewesen wäre. Das Framework wird heutzutage, soviel ich weiss, nicht mehr so viel

verwendet, zumindest nicht in unserem Geschäft. Somit kann ich das gelernte wohl auch nicht so schnell wieder einsetzen. Generell ist mir das viele Abschreiben als sehr nervig und nicht sehr lehrreich vorgekommen. Es lohnt sich, meiner Meinung nach, nicht eine schöne Applikation zu schreiben und dafür so viel Aufwand zu betreiben. Viel lieber hätte ich mit einer interessanteren Programmiersprache gearbeitet und selbst etwas erstellt.