

ClearYourRide

Abschlussaufgabe

**Seminar: Mapping und App-Entwicklung im Kontext von
Web-Anwendungen**

SoSe 2021

M.Sc. Marcel Storch, AG Umweltinformatik und Kommunalplanung



Erstellt von:

Moritz Lucas

Geoinformatik M.Sc. 2.FS

Matrikelnummer: 988853

molucas@uni-osnabrueck.de

Nele Wichert

Informatik M.Sc. 4.FS

Matrikelnummer: 964635

nwichert@uni-osnabrueck.de

Inhaltsverzeichnis

1. Einleitung	3
2. Leaflet-Karte	3
3. Hinzufügen eines neuen Missstandes	4
4. Darstellung der Marker und Filterfunktion	5
4.1 Abfrage der Missstände vom Server	6
4.2 Darstellung der Marker in der Karte	6
4.3 Popup der Marker	7
4.4 Legende mit Filterfunktion	8
5. Verarbeitung der Daten auf dem Server und Aufbau der Datenbank	9
6. Designideen	10
7. Genutzte Plugins und Bibliotheken	11
8. Bildquellen	12

1. Einleitung

Im Rahmen der Abschlusssaufgabe der Veranstaltung Mapping und App-Entwicklung im Kontext von Web-Anwendungen sollte eine Android-App auf Basis des Cordova-Frameworks programmiert werden, welche die Möglichkeit bietet Missstände in der städtischen Fahrradinfrastruktur zu melden und sich diese anzeigen zu lassen. Die hier im Folgenden beschriebene App trägt den Namen "ClearYourRide", was soviel wie "Fahrt frei machen" bedeutet. Nachstehend werden die Implementationen der verschiedenen umgesetzten Funktionen erläutert. Als Basis wurde eine Leaflet-Karte verwendet, welche den Standort des Nutzers anzeigt und dessen automatische Zentrierung sich ein- und ausschalten lässt. Anschließend wurde die Möglichkeit geschaffen einen neuen Missstand aufzunehmen und diesen in die Datenbank einzupflegen. Außerdem sollte es die Möglichkeit geben, alle in der Datenbank vorhandenen Einträge auf der Karte darzustellen. Dabei werden je nach Art der Einträge Marker mit verschiedenen Designs angezeigt. Zusätzlich zu diesen Pflichtaufgaben wurde eine Legende mit Filterfunktion implementiert mit welcher die Missstände nach verschiedenen Arten, der Zeit und nach bereits behobenen gefiltert werden können. Des Weiteren gibt es die Möglichkeit den Behoben-Status eines Eintrags innerhalb seines Popups zu ändern. Im Folgendem wird die Datenverarbeitung auf dem Server und der Aufbau der Datenbank beschrieben und die genutzten Plugins und Bibliotheken dargelegt. Abschließend gibt es ein Verzeichnis für die genutzten Bildquellen.

2. Leaflet-Karte

Die Hintergrundkarte für die App basiert auf den Daten von OpenStreetMap und wurde mithilfe der Leaflet-Bibliothek hinzugefügt. Zur leichteren Bedienbarkeit wurden die standardmäßig vorhandenen Zoom-Buttons beibehalten.

Zu Beginn ist der Kartenausschnitt standardmäßig auf Osnabrück gezoomt. Falls die tatsächliche aktuelle Position des Nutzers ermittelt werden konnte, wird auf diese zentriert. Um den Gerätestandort ab zu fragen wurde das Cordova-Plugin *Geolocation* verwendet.

Die aktuelle Position des Nutzers wird durch ein blauen Pin markiert und regelmäßig aktualisiert. Um diese Aktualisierung auszuschalten wurde in der oberen linken Ecke unter

den Zoom-Buttons ein Button hinzugefügt. Durch einen Klick auf diesen Positions-Button wird die Zentrierung auf den aktuellen Standort deaktiviert, der aktuelle Standort wird allerdings weiter aktualisiert, sodass dem Nutzer die Orientierung erleichtert wird und es nicht zu Verwirrungen kommt. Wird der Positions-Button erneut gedrückt, wird wieder auf letzten bekannten Standort gezoomt und dieser weiterhin aktualisiert.

Um dem Nutzer weiter die Übersichtlichkeit zu erleichtern, kann anhand des Aussehens des Positions-Buttons intuitiv der aktuelle Status der Standortverfolgung entnommen werden. Ist sie aktiviert erscheint ein blauer Pin, wird sie deaktiviert ist dieser Pin durchgestrichen und ausgegraut.

3. Hinzufügen eines neuen Missstandes

Nachfolgend wird erläutert, wie der Nutzer Mithilfe eines HTML-Formulars einen neuen Missstand aufnehmen kann und wie dieser an die Datenbank versendet wird.

Um einen neuen Missstand hinzuzufügen, kann der Nutzer am unteren rechten Bildschirmrand auf einen orangenen Button tippen, dieser öffnet mittels eines *onclick-Events* ein *PopUpDiv*. Durch die Größe und Farbe soll er einem schnell ins Auge fallen und durch das "Plus" eine Assoziation zum Hinzufügen erzeugen. Die Verwaltung der Anzeige dieses *divs* erfolgt über das Style-Element *display*. Innerhalb des *divs* befindet sich ein zweiseitiges HTML-Formular, in das der Nutzer Art des Missstandes, Erläuterung, Name und E-Mail angeben sowie ein Foto aufnehmen kann. Die Aufteilung auf zwei Seiten wurde vorgenommen, um für eine stärkere Übersichtlichkeit zu sorgen, insbesondere auf den recht kleinen Smartphone Bildschirmen. Schließen kann man das *PopUpDiv* am oberen rechten Rand mit einem Button, dadurch werden die Eingaben zurückgesetzt und bei erneutem Öffnen des Formulars beginnt man wieder auf der ersten Seite. Dort befindet sich ein *Select-Input* zum Hinzufügen der Art des Missstandes, daran schließt sich ein *Textarea-Element* an, in dem man den Missstand weitergehend erläutern kann. Am Ende der ersten Seite befinden sich zwei Buttons, die es ermöglichen noch ein Bild des Missstandes hinzuzufügen. Dabei hat man die Wahl es mit der Kamera aufzunehmen oder eines aus der Galerie zu wählen. Um dies zu Implementieren wurde das Cordova-Plugin *Camera* genutzt. Das hinzugefügte Bild wird oberhalb der Kamera-Buttons dargestellt und als *base64* codierter

String einem versteckten *Input* hinzugefügt, um ihn bei einem *submit* des HTML-Formulars mit abfragen zu können. Anschließend kann der Nutzer mittels des Weiter-Buttons auf die nächste Seite gelangen. Dabei verwaltet die Funktion *nextPrev()* die Darstellung der Seiten und ruft auf der ersten Seite die Funktion *validateForm()* auf, um zu prüfen, ob die Felder ausgefüllt sind. Außerdem wird die Funktion *showTab(n)* aufgerufen, welche die richtige Seite des Formulars aufruft und die Darstellung der Buttons verwaltet. Am unteren Rand des Formulars zeigen zwei Punkte an, auf welcher Seite des Formulars man sich befindet, dies wird durch die Funktion *fixStepIndicator(n)* verwaltet. Auf der zweiten Seite kann der Nutzer seinen Namen und seine E-Mail-Adresse angeben und falls er die Felder ausgefüllt hat, kann das Formular abgeschickt werden. Um zu verhindern, dass es zu mehrmaligen Abschieken eines Formulars kommt, wenn zum Beispiel die Datenmenge durch ein Bild hoch ist und es dadurch zu einer Verzögerung kommt, wird der *submit-Button* ausgegraut und deaktiviert. Dabei wird ein *EventListener* ausgelöst, in dem die eingegebenen Daten in ein *JSON* übertragen und Datum, Koordinaten und *behoben=false* hinzugefügt werden. Dieser *JSON* wird anschließend mittels eines *AJAX POST-Request* an den Server geschickt. Innerhalb des *Callbacks* wird noch ein *div* mit der gleichen oben beschriebenen Methode dargestellt, welches dem Nutzer eine Rückmeldung über Erfolg oder Misserfolg gibt. Am Server werden die Daten empfangen und mittels der Verbindung zur *abschluss_db* und einer *SQL-Query* in die Datenbank eingepflegt.

4. Darstellung der Marker und Filterfunktion

Nachfolgend wird erläutert, wie die Missstände vom Server empfangen werden, wie sie und die dazugehörigen Popups dargestellt werden und wie die Legende mit Filterfunktion implementiert wurde.

4.1 Abfrage der Missstände vom Server

Jedesmal bevor die Missstände in der Karte neu gezeichnet werden, erfolgt eine Abfrage der Datenbank auf dem Server. Dadurch wird sichergestellt, dass die Missstände auch während der Nutzung der App aktualisiert werden. Eine solche Aktualisierung passiert beispielsweise, wenn die Missstände nach bestimmten Kriterien gefiltert werden. Für die Abfrage vom Server wird die Funktion *getReportedGrievances(callback)* genutzt. Diese fragt alle auf dem

Server liegenden Misstände mit einem *AJAX-Post-Request* als *JSON* ab. Besonderheit hierbei ist, dass sie als Attribut eine *Callback-Funktion* beinhaltet. Dadurch kann sichergestellt werden, dass die Funktion *showGrievancesMarkers(problems)*, welche die Anzeige der Marker verwaltet, erst aufgerufen wird, wenn alle Misstände vollständig vom Server geladen wurden.

4.2 Darstellung der Marker in der Karte

Für die Anzeige der Misstände wird ein Marker-Clustering-Plugin für Leaflet verwendet. Wenn der Kartenausschnitt verkleinert wird, liegen viele Marker in einen kleinen Bereich und können nicht mehr richtig erkannt werden. Die Karte wirkt schnell unübersichtlich. Durch die Verwendung des Markercluster werden diese zu einen einzelnen Marker zusammengefasst. Der Bereich in welchem die zusammengefassten Punkte liegen, kann durch längeres Drücken als Markercluster angezeigt werden. Durch kurzes Drücken wird auf die zugehörigen Misstände dieses Clusters gezoomt.

Mithilfe des Befehls *new L.MarkerClusterGroup()* wird ein neues Markercluster erstellt. Um die Karte möglichst schlicht zu halten wurde das Design der Icons für die geclusterten Marker angepasst. Bei den Icons handelt es sich nun um schlichte schwarze Kreise, mit einem Durchmesser welcher der Höhe der Misstandsmarker entspricht. In jedem Icon wird die Anzahl der zusammengefassten Misstände vermerkt. Dies wurde in der Funktion *iconCreateFunction* und durch anpassen der CSS die Klasse *marker-cluster* implementiert.

Über die einzelnen Misstände wird in einer for-Schleife iteriert. Dabei wird geprüft, ob der aktuelle Misstand mit den aktuellen Filtereinstellungen angezeigt werden soll. Anschließend wird ein Popup erstellt und je nach Misstandskategorie ein unterschiedliches Bild für den Marker gewählt. Die Marker für verschiedene Kategorien unterscheiden sich sowohl in der Farbe als auch vom Symbol. Durch die verschiedenen Farben soll eine bessere Übersichtlichkeit gewährleistet werden und eine schnelle Unterscheidung möglich sein. Durch die verschiedenen Symbole soll die genaue Kategorie intuitiver und auch ohne Blick auf die Legende erkennbar sein. Der Marker wird anschließend mit *L.marker([item[4], item[5]], {icon: meldungMarker})* an den entsprechenden Koordinaten mit dem zugehörigen Bild erstellt. Zusätzlich wird das zugehörige Popup mit *bindPopup()* hinzugefügt. Anschließend wird der so erzeugte Marker mit der Funktion *addLayer()* zu dem

Markercluster hinzugefügt. Nachdem alle Missstände abgearbeitet wurden, wird schließlich auch das fertige Markercluster mit der Funktion *addLayer()* zu der Karte hinzugefügt.

4.3 Popup der Marker

Jeder Marker enthält ein Popup, welcher durch einen Klick auf den entsprechenden Missstand geöffnet werden kann. In diesem werden weitere Informationen über den einzelnen Missstände angezeigt. Zu diesen Informationen gehören die ins Deutsche übersetzte Missstandsart, eine kurze Beschreibung und das Datum an dem der Missstand hinzugefügt wurde. Weitere Informationen wie der Nutzernamen oder die Mailadresse werden aus Datenschutzgründen nicht angezeigt. Ebenfalls wird ein vorhandenes Bild nicht angezeigt, da dies insbesondere bei mobiler Nutzung, die Performance deutlich einschränkt.

Optional ist für einen Missstand auch noch vermerkt, ob diese bereits behoben oder noch aktuell ist. Wenn bei den Missständen von anderen Gruppen kein Behoben-Status vorhanden ist, wird dieser als unbekannt angenommen und kann nicht geändert werden. Ist dagegen diese Information vorhanden, enthält das Popup noch einen Button um diesen Status zu ändern. Der Text in diesen Button unterscheidet sich je nachdem in welchem Behoben-Status der Missstand geändert werden kann.

Wenn der *Button* "Missstand ist behoben" oder "Missstand ist nicht behoben" gedrückt wird, wird auf dem *onclick-Event* eine Funktion ausgeführt, die als Attribute die *id* und den aktuellen *behoben-Status* übernimmt. Diese überführt die Attribute in einen *JSON* und sendet sie mittels eines *AJAX-Post-Request* an den Server, wo der *behoben-Status* an der korrekten *id* geupdatet wird.

4.4 Legende mit Filterfunktion

Um den Bildschirmplatz platzsparend zu verwendet und somit die Anwendung möglichst übersichtlich zu gestalten, wurde die Legende und die Filterfunktion in einem Fenster zusammengefasst. Dieses Fenster wurde in der oberen rechten Ecke des Bildschirms hinzugefügt und ist einklappbar. Zusätzlich zu der CSS-Klasse mit dem allgemeinen Styling der Fenster und der Legende, wurde je nachdem ob die Legende aktuell ein- oder ausgeklappt ist

eine weitere CSS-Klasse hinzugefügt. Dadurch ist es möglich die Legende je nach aktuellen Zustand verschieden zu stylen.

Um zu erreichen, dass der Knopf zum ein- und ausklappen an einer ähnlichen Position bleibt, muss auch bei der ausgeblendeten Legende ein gewisse Zeilenhöhe und Padding erhalten bleiben. Dies wurde in der CSS-Klasse *legendDiv* umgesetzt. Damit dieses bei einer ausgeblendeten Legende nicht stört wurde in *legendDiv_hide_true* die Hintergrundfarbe und die Rahmenfarbe auf transparent angepasst. Ist die Legende hingegen eingeblendet, wurde in *legendDiv_hide_false* mit *overflow-y: auto* erreicht, dass die Legende falls notwendig scrollbar ist. Die maximale Höhe der Legende wurde auf maximal 72% des gesamten Fensters festgesetzt. Dadurch wird bei den gängigen Handygrößen ein guter Kompromiss aus nicht zu größer und zu kleiner Legende erreicht und der Button zum Hinzufügen eines Missstandes nicht verdeckt.

Als oberstes Element in der Legende befindet sich ein Button um diese ein- oder auszublenden. Dieser ist immer vorhanden, das Aussehen und das Bild des Buttons unterscheidet sich allerdings je nach Zustand. Der Rest der Legende wird nur angezeigt, wenn diese eingeblendet ist. Zu Beginn wird alles angezeigt, um den Benutzer direkt eine Übersicht zu bieten.

Insgesamt bietet die Legende drei verschiedene Filterfunktionen. Zum einen können einzelne Missstandsarten ausgeblendet werden. Der Bereich hierfür wurde tabellenartig strukturiert. In der ersten Spalte befindet sich eine *checkbox*, über welche die entsprechende Kategorie ein oder ausgeblendet werden. In der zweiten Spalte befindet sich der jeweilige Kategorienamen und in der dritte Spalte das Symbol des Missstandsmarker. Durch diese Strukturierung ist auch direkt die Legendenfunktionalität gegeben. Es ist erkennbar welche Missstandsmarker zu welchem Missstandsarten gehören und ob diese ein- oder ausgeblendet sind. Die zweite Möglichkeit der Filterung ist es einen gewissen Zeitraum festzulegen. Durch die Verwendung eines *date* Eingabefeldes, lässt sich dies einfach über ein Kalender einstellen. Wurde hier keine Änderung vom Nutzer vorgenommen, wird in dem Kästchen das früheste und späteste Datum aller Missstände angezeigt. Diese Daten werden, nachdem die Missstände vom Server geladen wurden und bevor sie in der Karte eingezeichnet werden, ermittelt. Zuletzt ist es noch möglich, behobene Missstände auszublenden. Dabei werden auf der Karte nur explizit als nicht behoben sowie alle Missstände mit unbekannten Status

angezeigt. Für die drei verschiedenen Filter wurde jeweils ein Event-Handler implementiert, welcher die entsprechenden Zustandsvariablen anpasst und dann mit der Methode *showGrievancesMarkers()* die entsprechenden Marker neu zeichnet.

5. Verarbeitung der Daten auf dem Server und Aufbau der Datenbank

Auf dem Server wird mittels *Node.js* der Webserver gestartet, der die vom Client gesendeten Daten empfängt und weiter an die Datenbank leitet. Die Verwaltung des Webserver läuft dabei über das Webframework *Express.js*. Die Kommunikation mit der SQL-Datenbank wird mittels *node-postgres* durchgeführt. Diese werden zu Beginn eingeladen und initialisiert. Die maximale Datenmenge, die mittels *JSON* verschickt werden kann, wird auf 10 Mb begrenzt, was für das Versenden der Bilder mittels Base64 notwendig ist. Als *Access-Control-Allow-Origin* wird einzig *http://localhost:8000* erlaubt, auf dem dann POST und GET Methoden durchgeführt werden können. Dadurch wird die Sicherheit gesteigert, da die einzige Aufgabe des Servers ist, mit der Datenbank und dem Client zu interagieren.

Dann wird die Verbindung mit der *abschluss_db* aufgenommen, auf der die Misstände gespeichert werden. Diese besteht aus einer Relation *meldungen*, welche aus 10 Attributen besteht. Dabei sind bis auf *photo* und *behoben* alle Attribute erforderlich, um einen Eintrag vorzunehmen.

Zum einpflegen der Daten wurde ein *POST-Request (newRequest)* implementiert, welcher die als *JSON* versendeten Formulardaten eines neu eingetragenen Misstandes empfängt und diese mittels *INSERT INTO* Befehls der Datenbank hinzufügt. Um den behoben-Status zu aktualisieren liegt ein weiterer *POST-Request (changeBehoben)* vor, welcher einen *JSON* mit der *id* und dem aktuellen *Behoben-Status* empfängt und mittels eines *UPDATE* Befehls *behoben* an der Stelle mit der übereinstimmenden *id* neu setzt. Abschließend wird ein *GET-Request (getProblems)* genutzt, damit der Client die Misstände vom Server abfragen kann, dabei werden alle Attribute bis auf *photo* abgefragt, da dieses aufgrund des hohen Speicherbedarfs die Abfrage stark verlangsamt.

6. Designideen

Da es sich um eine mobile Anwendung handelt, sollte eine leichte Bedienbarkeit am Handy und ein ansprechendes Design erreicht werden. Hierfür wurde darauf geachtet, dass die App nicht überladen wirkt. Ebenfalls wurden die verschiedenen Elemente einheitlich und schlicht gestaltet.

Die selbst erstellten Buttons wurden, in der Größe und durch die abgerundeten Ecken mit einem grau-transparenten Rand, vom Aussehen ähnlich zu den Leaflet-Buttons geformt. Außerdem wurden die Fenster, wie beispielsweise das von der Legende oder dem Missstand-Hinzufügen-Menü, mit demselben Rahmen und denselben Ecken wie die Buttons versehen. Um dieses Design einheitlich und ohne zu viel Coderedundanz zu erreichen, wurden meist übergeordnete CSS-Klassen verwendet. Beispielsweise wurden eine Schriftart und Schriftfarbe für alle *div* festgelegt. Da der gesamte Text der App in einem *div* eingebettet ist, wird alle Schrift gleich gestylt. Für bestimmte (Unter-)Bereiche wurden dann zu den allgemeinen Einstellungen explizit zusätzliche Anpassungen vorgenommen. So wurde zum Beispiel allen beschreibenden Überschriften die Klasse *description* mit angepasster Schriftgröße zugewiesen. Das selbes Vorgehen wurde unter anderem auch für die Buttons angewendet, wobei hier beispielsweise nach Art des Buttons unterschieden wurde. So gibt es eine Klasse für das allgemeine Aussehen der Buttons und unter anderem eine Klasse für Buttons mit Bildern.

7. Genutzte Plugins und Bibliotheken

Kamera-Plugin “cordova-plugin-camera”

Cordova eigenes Plugin welches die Aufnahme von Bildern und die Wahl von Bildern aus der Galerie ermöglicht.

<https://cordova.apache.org/docs/en/10.x/reference/cordova-plugin-camera/>

Positions-Plugin “cordova-plugin-geolocation”

Cordova eigenes Plugin welches Informationen über den Standort des Gerätes gibt. Dabei werden neben GPS, Netzwerksignale, RFID, WiFi und Bluetooth genutzt, um den Standort zu ermitteln.

<https://cordova.apache.org/docs/en/10.x/reference/cordova-plugin-geolocation/>

Leaflet “Markercluster”

Eine Plugin für interaktive Karten, welcher eine übersichtliche animierte Marker-Clustering-Funktionalität für Leaflet bietet.

<https://github.com/Leaflet/Leaflet.markercluster>





Google Font “Raleway”

Google eigene Schriftart, welche mithilfe eines *css-stylesheets* von den Google-Servern eingeladen wird.

<https://fonts.google.com/specimen/Raleway>

8. Bildquellen

Für verschiedene Buttons wurden Bilder verwendet dessen Quellen hier im weiteren angegeben werden. Das Icon der App, der Positionsmarker sowie die Marker für die verschiedenen Missstände sind Eigenkreationen, weshalb für diese im Weiteren keine Quellen angegeben werden. Die Bilder sind unter [www/img/](http://www.img/) zu finden.

Name	Bild	Quelle
Hinzufügen eines Missstandes (add.png)		https://icons8.com/icon/60953/add (zuletzt abgerufen am 24.09.2021)
Schließen des Formulars (close.png)		https://www.flaticon.com/free-icon/close_1828778?term=close&page=1&position=2&page=1&position=2&related_id=1828778&origin=search (zuletzt abgerufen am 24.09.2021)
Bild mit Kamera hinzufügen (camera.png)		https://www.flaticon.com/free-icon/camera_2965705?term=camera&page=1&position=12&page=1&position=12&related_id=2965705&origin=search (zuletzt abgerufen am 24.09.2021)
Bild mit Galerie hinzufügen (galerie.png)		https://64.media.tumblr.com/5ca9f08da33f02974bda7ceca9de9de4/e09d9c4fae037515-ab/s1280x1920/865daff9d20becd770b8988494ec688c37443d4c.png (zuletzt abgerufen am 24.09.2021)