Prototyping Projektdokumentation

Name: Kiyan Rassouli

E-Mail: rassokiy@students.zhaw.ch

URL der deployten Anwendung: https://fangmonitor.netlify.app

Inhaltsverzeichnis

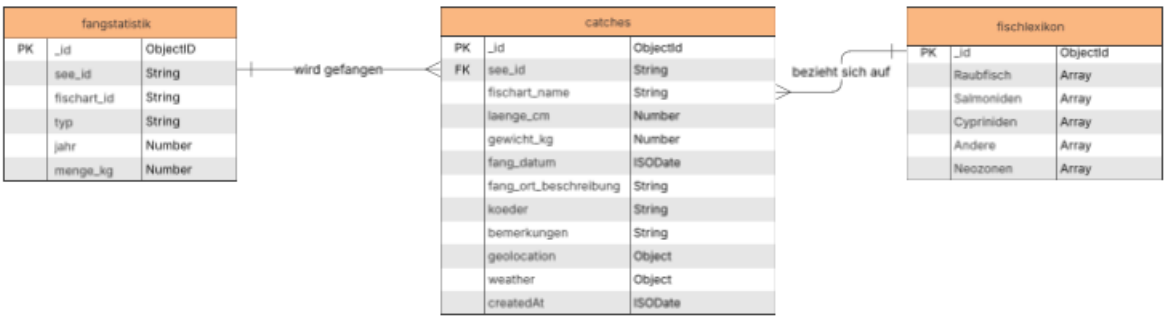
1. Einleitung

In der Schweiz ist die Erfassung der Fischereistatistiken kantonal geregelt. In vielen Kantonen erfolgt dies noch über Stift und Papier. Das Ziel dieser Anwendung soll sein, die Erfassung zu vereinfachen und vereinheitlichen. Ein weiterer Bestandteil der App ist die interaktive Gewässerübersicht: Für jeden unterstützten See können Nutzerinnen und Nutzer die Entwicklung der Fangmengen nach Fischart und Jahr einsehen. Zudem zeigt die Anwendung ein Fischlexikon mit den häufigsten Fischarten der Schweiz, so dass Einsteiger sich direkt einen Überblick verschaffen können, bevor sie ans Wasser gehen.

Zudem bietet FangMonitor eine datenbasierte Webapplikation zur Analyse, Visualisierung und Erfassung von Fischfängen in Schweizer Seen. Die Anwendung vereint amtliche Fangstatistiken mit Echtzeit-Wetterinformationen und ermöglicht eine ortsbezogene Dokumentation individueller Fänge. Ziel ist eine genauere Datenbasis für Fischereistatistiken zu erhalten, wo Ort, Zeit und aktuelle Wetterentwicklung dokumentiert sind, zugunsten der Fischerei und der Fischereibewirtschaftung. Die Daten

FangMonitor richtet sich an Freizeitfischerinnen und Fischer, Behörden und Umweltinteressierte, die Fischfangdaten systematisch auswerten, dokumentieren oder vergleichen möchten wissenschaftlich fundiert, visuell aufbereitet und regional fokussiert. Weg vom Kantönligeist hin zu einer einheitlichen Datengundlage.

2. Datenmodell



Die Relationen in diesem NoSQL-Kontext werden nicht durch Datenbank-Constraints erzwungen, sondern durch die Anwendungslogik in db.js und den Server-Dateien hergestellt, wenn Daten abgefragt und kombiniert werden.

1. Erfasster Fang (catches) und Fischlexikon (fischlexikon):

Die Anwendung verknüpft einen Eintrag in «catches» mit dem «fischlexikon» über das Feld catches.fischart_name. Dieser Name entspricht dem name-Feld eines Fischeintrags im Fischlexikon. Dies ermöglicht es der Anwendung, Detailinformationen zu einem gefangenen Fisch aus dem Lexikon zu ziehen oder Vorschläge im Eingabeformular zu machen (via getAllFischartenNamen in db.js).

2. Erfasster Fang (catches) und dem entsprechenden See (aus Fangstatistik):

Die Anwendung verwendet das Feld catches.see_id, um einen Fang einem See zuzuordnen. Die Liste der gültigen Seen für die Eingabe wird dynamisch aus den eindeutigen see_id-Werten der Fangstatistik-Collection generiert (via getUniqueLakeIds in db.js). Dies stellt sicher, dass Fänge für Seen erfasst werden, für die auch historische Vergleichsdaten existieren könnten.

3. Statistikanzeige (basierend auf fangstatistik) und Fischlexikon (fischlexikon):

Bei der Anzeige der aggregierten Statistiken (z.B. in der Detailansicht einer Statistikgruppe) enthält die fangstatistik-Collection die fischart_id. Die Anwendung nutzt diese fischart_id (die dem name-Attribut im Lexikon entspricht), um den vollständigen, benutzerfreundlichen Namen des Fisches aus dem Fischlexikon darzustellen. Dies geschieht typischerweise über eine Mapping-Tabelle oder direkte Logik im Frontend-Code (Zugriff via db.js auf getAllFischartenNamen in den Svelte-Komponenten).

3. Beschreibung der Anwendung

3.1. Startseite

Route: routes/

Die Startseite ist schlicht und man kann direkt zu den Grundfunktionen der Applikation navigieren.

Fischfangstatistiken anzeigen führt zur Übersicht der erfassten Seen.

Fischlexikon durchstöbern führt zu der Übersicht der verschiedenen Kategorien, wo man schliesslich nach Fischartenkategorie ein entsprechendes Lexikon konsultieren kann.

Neuen Fang eintragen führt zum Formular für das Eintragen von Fängen.

Datei: routes/+page.svelte

3.1 Fangstatistik nach Seen

Route: /fischfangstatistik

Nach Auswahl von «Fangstatistiken anzeigen» kann man hier das gewünschte Gewässer auswählen, wobei man zu fischfangstatistik/[see_id] gelangt. Im Moment sind die fünf grössten Schweizer Seen erfasst. Durch die Auswahl des gewünschten Gewässers wird man weiter zu den Jahresübersichten seit 2013 geleitet. Wenn man mit der Maus über den gewünschten See fährt, wird dieser dunkelblau eingefärbt (siehe hier Bodensee). Mit Klick auf «Zurück zur Startseite» gelangt man entsprechend auf die Startseite.

Dateien:

- lib/server/db.js → via getUniqueLakeIds
- src/routes/fischfangstatistik/+page.server.js
- src/routes/fischfangstatistik/+page.svelte

3.1.1. Datenstruktur von /fischfangstatistik

Die Daten für die Fischfangstatistiken sind in der MongoDB-Datenbank FangMonitor gespeichert. Die Collection in dieser Datenbank heisst fangstatistik. Die Daten sind wie folgt strukturiert:

Für jeden See gibt es pro Typ, Fisch und Jahr ein Dokument.

- Typ bezeichnet hierbei, ob es sich bei der Statistik um die Angel- oder Berufsfischerei handelt, oder ob beide Statistiken zusammengefasst wurden.
- see_id: Im String-Format, jeder See teilt eine ID
- fischart_id: Gleich wie bei see_id. Die historischen Daten haben Fangstatistiken für die Fischarten- bzw. Übergruppen der Fische Egli, Hecht, Felchen, Cypriniden, Forellen, Saiblinge und Restliche Arten erfasst. Dies hat den Hintergrund, dass diese Fischarten für die Fischerei am interessantesten sind und Fangstatistiken der jeweiligen Kantonalen Jagd- und Fischereiverwaltungen nur die Erfassung bestimmter Arten explizit verlangen.

Die Daten sind dort noch sehr roh und nicht im Format, welches für die Applikation angewendet wird. Aus diesem Grund werden diese mit Mongo-DB Aggregationen in db.js angepasst. Dies wird im Kapitel Erweiterungen genauer erläutert.

Die Daten für die Fischfangstatistik stammen von:
https://www.fischereistatistik.ch/de/statistics?tt=0&dt=0&at=0&st=0&dp=0&ar=CH&wt=0&th=10&un=0&in=0&yr%5Bfrom%5D=2000&yr%5Bto%5D=2023&sp=100102

3.1.2. Statistiken für den ausgewählten See

Route: routes/fischfangstatistik/[see_id]

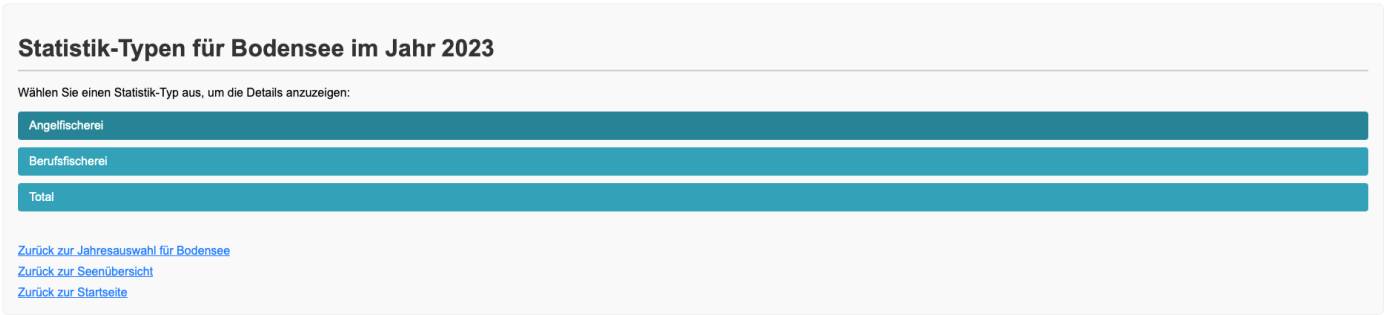
In diesem Beispiel wurde in /fischfangstatistik der Bodensee ausgewählt. Hier kann man für den ausgewählten See jeweils noch das gewünschte Jahr auswählen. Die Anzeige reicht von 2013 bis 2023, da vor 2013 keine einheitlichen Daten verfügbar waren und (Stand 11.5.2025) noch keine Daten für das Jahr 2024 verfügbar waren. Durch «hovern» über das gewünschte Jahr, verändert sich die Farbe entsprechend. Ausserdem kann man unten über die Links zurück zur Seenübersicht oder direkt zurück auf die Startseite navigieren.

Dateien:

- lib/server/db.js → via getUniqueYearsForSee
- routes/fischfangstatistik/[see_id]/+page.server.js
- routes/fischfangstatistik/[see_id]/+page.svelte

3.1.3. Statistik-Typen für den ausgewählten See im ausgewählten Jahr

Route: routes/fischfangstatistik/[see_id]/[jahr]

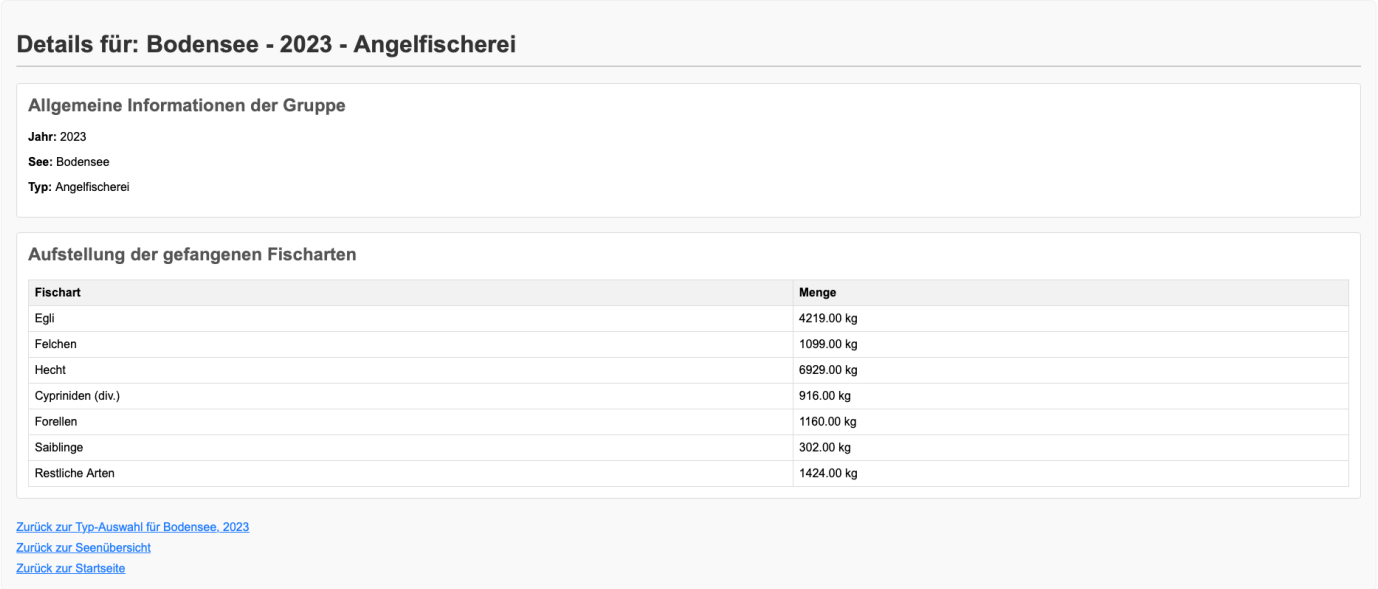


Auf dieser Seite werden für den ausgewählten See und das ausgewählte Jahr die zur Verfügung stehenden, historischen Fangstatistiken angezeigt. Wie bei den Seiten zuvor, wird auch hier via «hover» die gewünschte Route dunkel eingefärbt. Ausserdem gelangt man über die Links «Zurück zur Jahresauswahl für [see_id]», «Zurück zur Seenübersicht» und «Zurück zur Startseite» auf die gewünschte Seite zurück. Mit Klick auf den gewünschten Statistik-Typ gelangt man auf die detaillierte Statistik des gewünschten Sees, für das gewünschte Jahr und den gewünschten Typ.

- lib/server/db.js → via getUniqueTypesForSeeYear
- routes/fischfangstatistik/[see_id]/[jahr]/+page.server.js
- routes/fischfangstatistik/[see_id]/[jahr]/ +page.svelte

3.1.4. Details für den gewünschten Statistik-Typ

Route: routes/fischfangstatistik/details/[see_id]/[jahr]/[typ]



Die Seite zeigt die Übersicht der gewünschten Statistik. Die Statistik wird auf Jahresebene geführt und ausgewertet. Die Menge bezeichnet also die Summe aller Fänge, gemessen in Kilogramm für die verfügbaren Fischarten. Die Funktionalität beschränkt sich auf die Ansicht der jeweiligen Statistik, wobei auch hier jeweils die Zurücklinks implementiert wurden.

Dateien:

- lib/server/db.js → via getSingleFangstatistikGroup
- routes/fischfangstatistik/details/[see_id]/[jahr]/[typ]/+page.server.js
- routes/fischfangstatistik/details/[see_id]/[jahr]/[typ]/+page.svelte

3.2. Fischlexikon

Route: routes/fischlexikon

Hier wurde die Gestaltung von der Fischfangstatistik abgegrenzt, indem grün als Grundfarbe dient. Die Fischkategorien sind hier nach Kategorien aufgeschlüsselt. Insgesamt sind 30 Fischarten erfasst. Diese werden wiederum in fünf Kategorien unterteilt. Mit hovern über die gewünschte Kategorie, wird diese dunkel eingefärbt und unterstrichen. Mit Klick auf die Kategorie wird weiter navigiert auf /fischlexikon/[category].

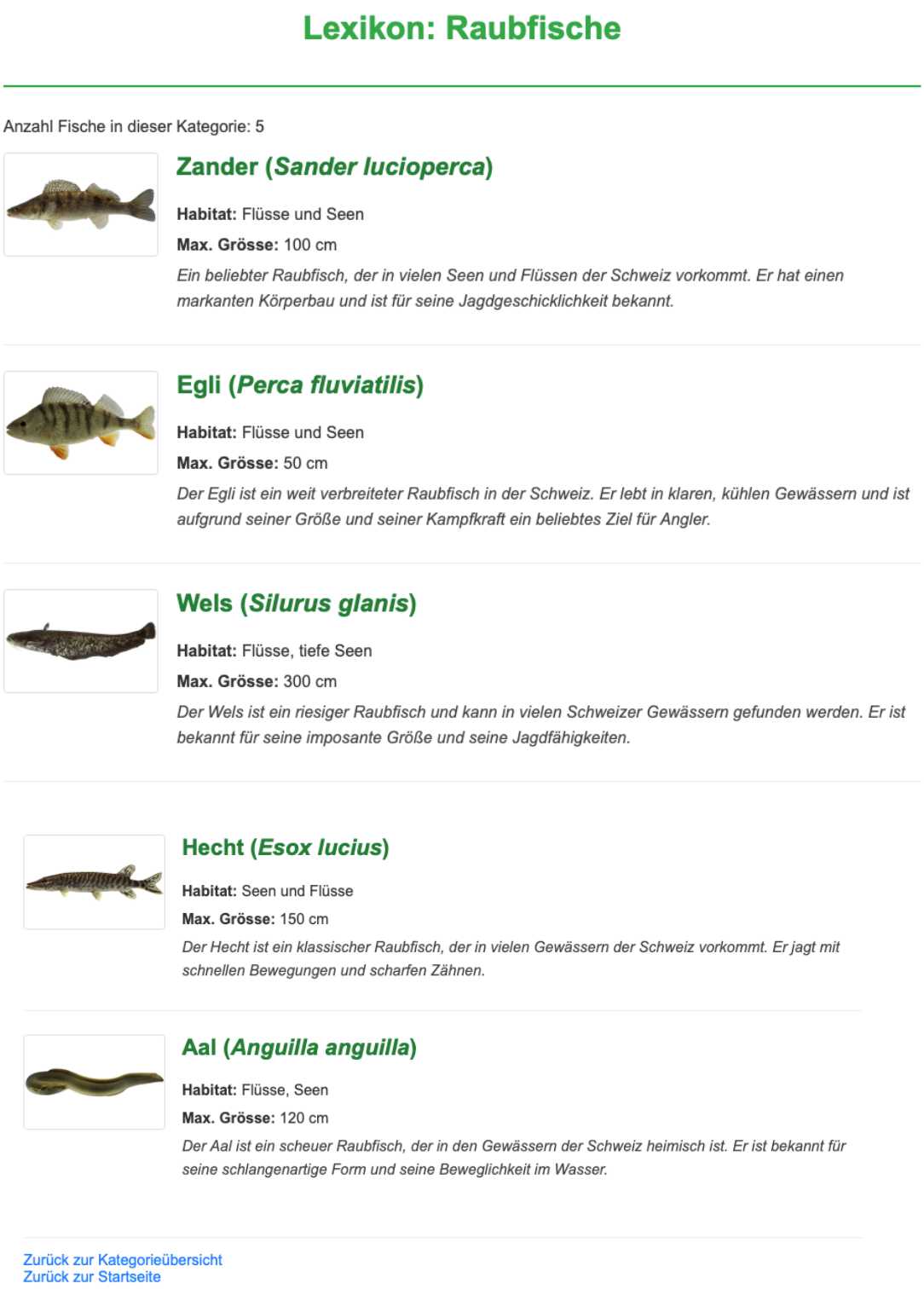
Auch hier kann man direkt auf die Startseite navigieren.

Dateien:

- routes/fischlexikon/+page.svelte
- static/images

3.2.1. Lexikon nach Fischkategorie

Route: routes/fischlexikon/[category]

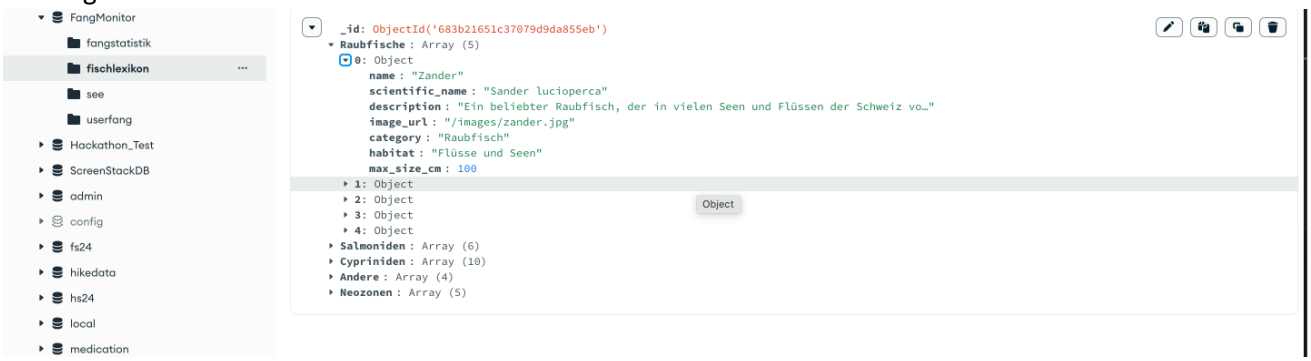


Hier gibt es pro Fischart der entsprechenden Kategorie je einen Steckbrief. Als Titel wird der Name, mit der wissenschaftlichen Bezeichnung in Klammern angezeigt. Ausserdem wird ein neutrales Bild der jeweiligen Fischart abgebildet. Im Steckbrief abgebildet sind das Habitat, die maximale Grösse und eine Kurzbeschreibung.

Auch hier kann man in der Fussnote der Seite entweder auf die Kategorienübersicht oder auf die Startseite navigieren.

Die Daten stammen aus dem Buch «Fische der Schweiz» von BirdLife Schweiz in Zusammenarbeit mit dem Schweizerischen Fischereiverband SFV. Die Daten selbst wurden dann via ChatGPT in JSON-Format gebracht und in der Collection «fischlexikon» in der Datenbank FangMonitor in MongoDB gespeichert. Die Datenstruktur dort sieht

wie folgt aus:



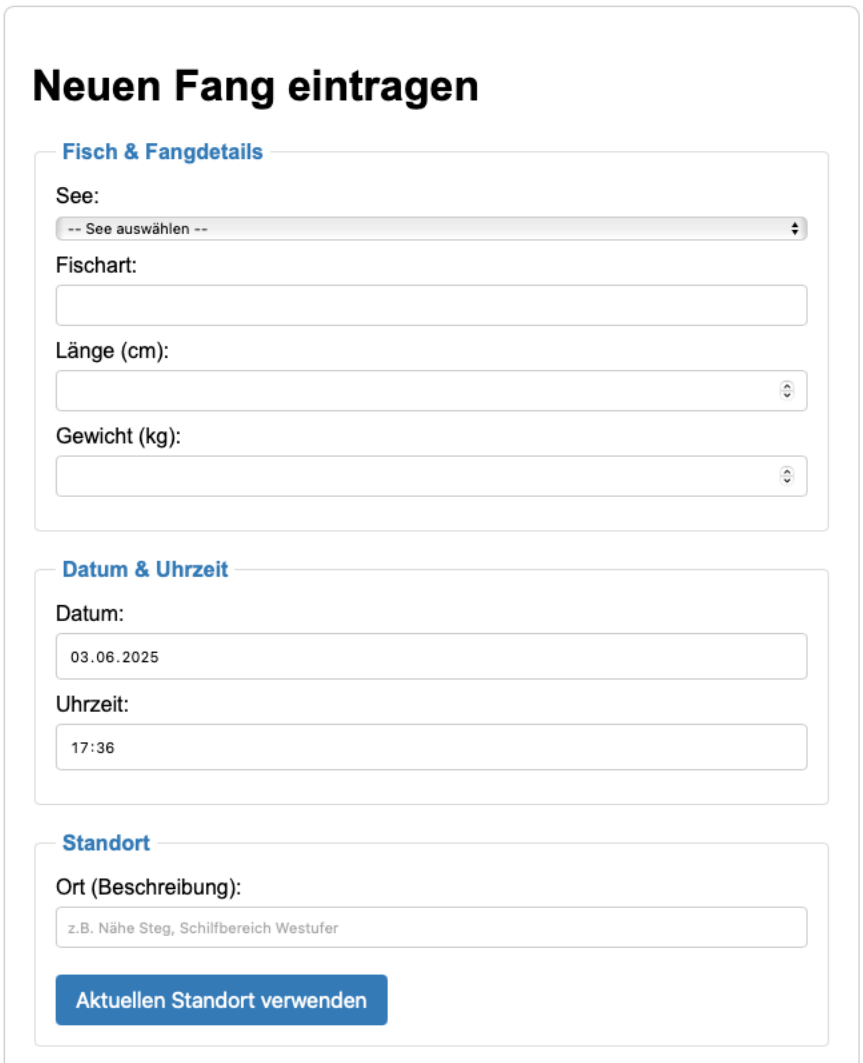
Der Zugriff auf MongoDB folgte via db.js und im Frontend wiederum mit page.server.js. Relevant für diese Seite in db.js ist die Funktion getFischForCategory.

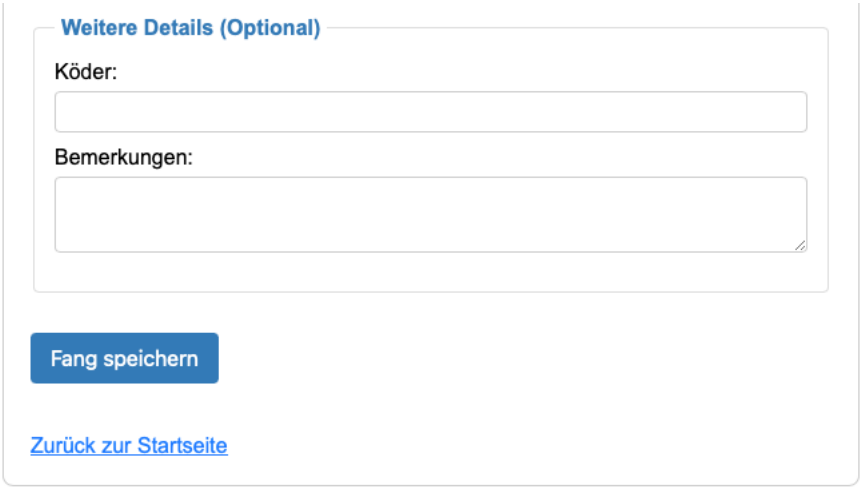
Dateien:

- lib/server/db.js → via getFischForCategory
- routes/fischlexikon/[category]/+page.server.js
- routes/fischlexikon/[category]/+page.svelte

3.3. Neuen Fang eintragen

Route : routes/fang-eintragen





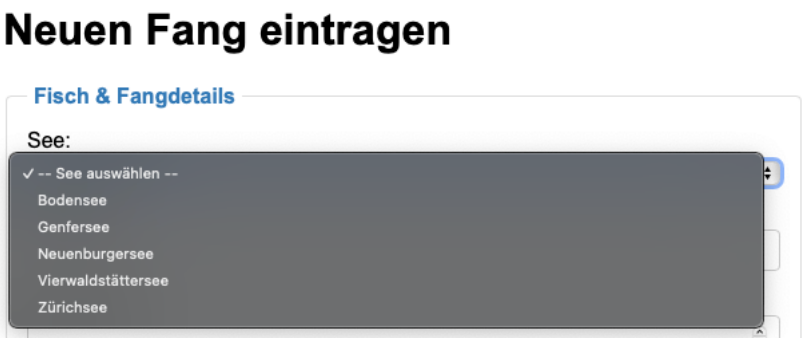
Hier kann der User einen Fang mit Hilfe eines Formulars eintragen.

Mit einem Klick auf das Dropdownmenü «-- See auswählen --» wird eine Liste von Seen angezeigt, welche in der Datenbank vorhanden sind.

Dateien:

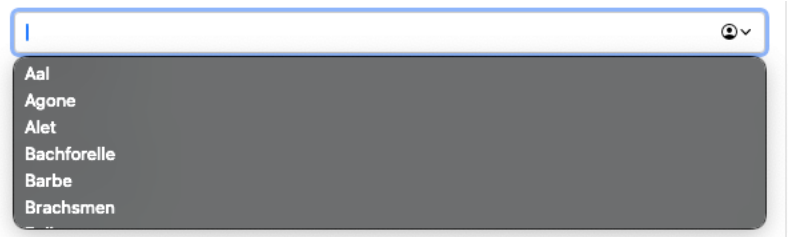
- lib/server/db.js → getUniqueLakeIds und getAllFischartenNamen, schreibt Dateien via addCatch
- routes/fang-eintragen/+page.server.js
- routes/fang-eintragen/+page.svelte

3.3.1. See Auswählen

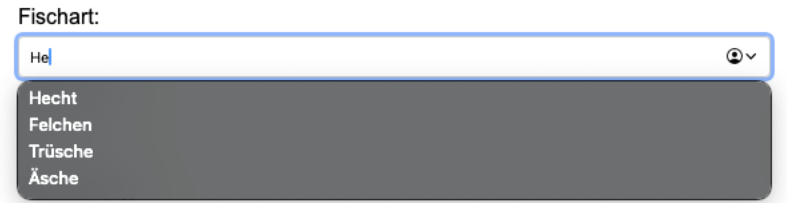


3.3.2. Fischart auswählen

Das Freitextfeld ermöglicht es, aus dem Dropdownmenü eine Fischart einzugeben, oder auch explizit zu suchen. Die Fischarten stammen aus der MongoDB-Collection «fischlexikon» und sind alphabetisch geordnet.



Mit Suche:



3.3.3. Länge eingeben

Bei Länge eingeben kann der User die Menge in cm eintragen.

Ein Klick auf den Pfeil ermöglicht es in 0.1 cm Schritten, die Länge anzupassen

3.3.4. Gewicht eingeben

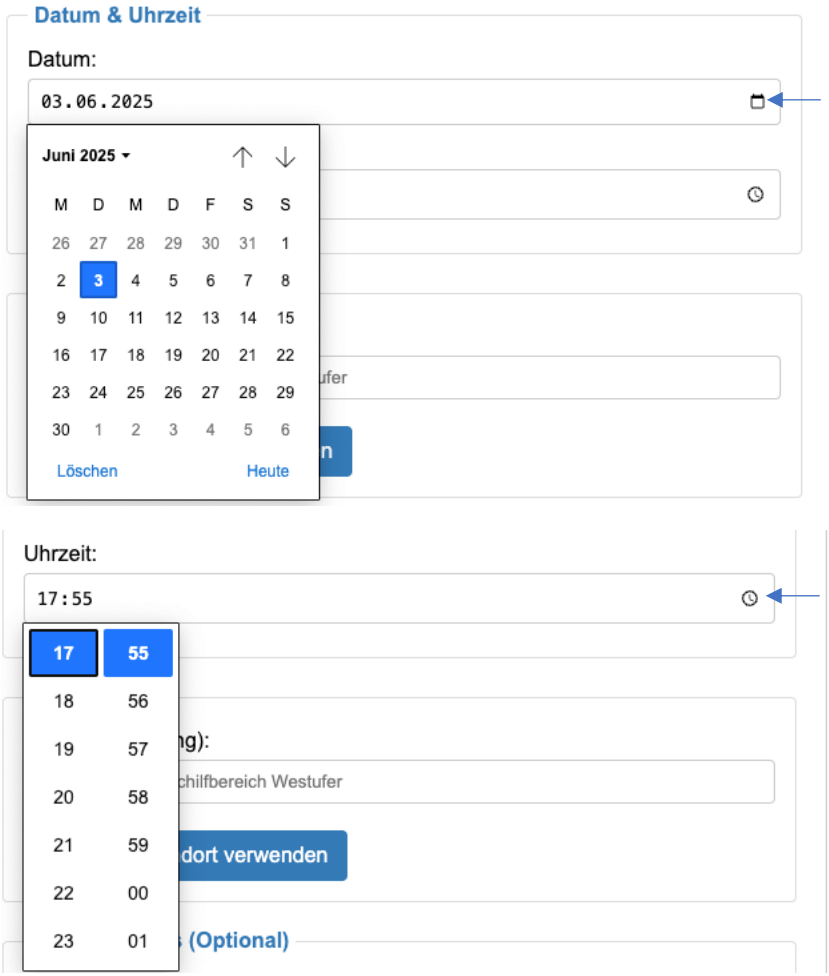
Hier kann der User das Gewicht in kg erfassen

und in 10g, also 0.01kg Schritten adjustieren.

3.3.5. Datum & Uhrzeit erfassen

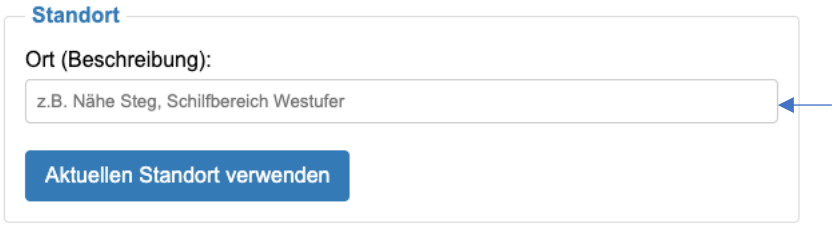
Hier werden Datum- und Uhrzeit angezeigt, bzw. eingegeben.

Durch klick auf das Datums- bzw. das Uhrzeitfeld kann beides jeweils angepasst werden:



3.3.6. Standort auswählen

Der Standort kann im Freitextfeld optional näher Beschrieben werden.



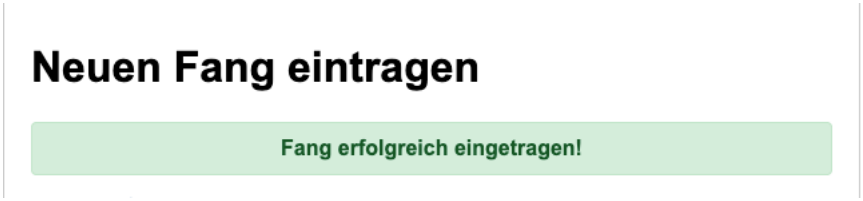
Durch Klick auf «Aktuellen Standort verwenden, wird der aktuelle Standort ermittelt und beim Absenden des Formulars mitgesendet. Die Längen- und Breitengrade sind auf sechs Nachkommastellen genau. Dies bedeutet, orientiert am Äquator, eine Genauigkeit von ca. 11 cm. Aufgrund der Erdabplattung strebt dieser Wert gegen Null, je näher man sich den Polen nähert. In der Schweiz beträgt diese Genauigkeit 7-8 cm.

3.3.7. Weitere Details

Hier kann man jeweils optional noch weitere Details wie Köderwahl und allgemeine Bemerkungen verfassen.

3.3.8. Fang Speichern

Durch Klick auf Fang speichern, erscheint die Bestätigung ganz oben unter dem Titel.



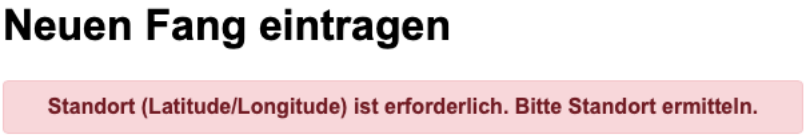
3.3.9. Fehlende Formularelemente

Fehlen Pflichtfelder im Formular, so werden diese jeweils wie folgt angezeigt:

- Fehlender See

- Fehlende Fischart:

- Fehlender Standort:

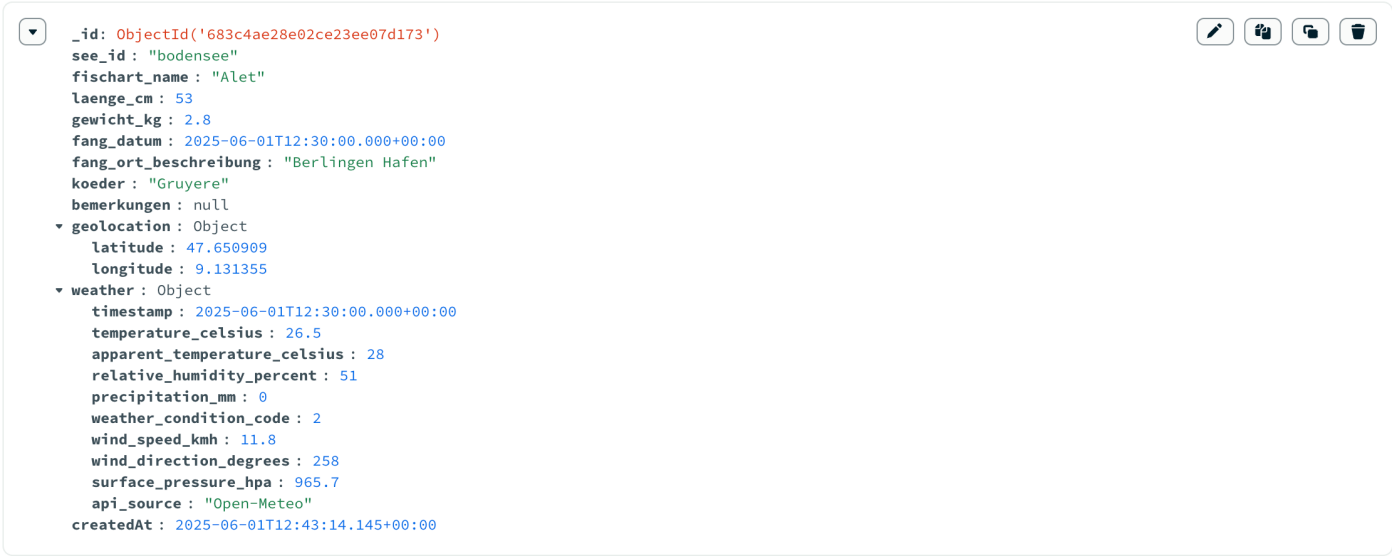


Pflichtfelder sind also Standort, See und Fischart, die anderen Daten sind keine Pflichtfelder.

3.3.10. Gespeicherte Daten

Auf die Details der Datenerfassung soll im Kapitel «Erweiterungen» eingegangen werden. Hier eine grobe Erklärung. Es werden natürlich sämtliche Userdaten erfasst und in der MongoDB-Collection «catches» gespeichert. Die Daten enthalten jedoch nebst den Userdaten auch noch Wetterdaten, die im Hintergrund über die API Open-Meteo mitgegeben werden.

In der Datenbank sieht dies dann so aus:



4. Erweiterungen

Die folgenden Erweiterungen wurden über die Grundanforderungen hinaus implementiert und tragen zur Funktionalität, Effizienz und zum Wert der Anwendung bei:

4.1. Umfangreicher Einsatz des MongoDB Aggregation Frameworks zur Datenstrukturierung

Beschreibung: Die Rohdaten der fangstatistik-Collection (ein Dokument pro Fischart/See/Jahr/Typ) werden serverseitig durch das MongoDB Aggregation Framework transformiert. Die Funktion getSingleFangstatistikGroup nutztStages wie $match für dynamische Filterung und insbesondere $group zur Umstrukturierung der Daten.

Im folgenden erkläre ich die einzelnen Aggregationen im Detail.

Datei/Route: lib/server/db.js

4.1.1. async function getSingleFangstatistikGroup(jahrParam, seeIdParam, typParam)

Ziel: Diese Funktion holt die Details für genau eine spezifische Statistikgruppe, identifiziert durch eine exakte Kombination von Jahr, See-ID und Typ, um dies dann entsprechend auf den Detailseiten anzuzeigen. Hier wird bereits eine Detailliste zurückgegeben, die den jeweiligen Parametern entspricht, so wie sie schliesslich im Frontend angezeigt wird.

```
        { $match: { jahr: jahr, see_id: seeIdParam, typ: typParam } },
        { $group: { _id: { see_id: "$see_id", jahr: "$jahr", typ: "$typ" }, fischarten: { $push:
{ fischart_id: "$fischart_id", menge_kg: "$menge_kg" } } } }
```

4.2. Effiziente Nutzung von distinct() für dynamische Elemente und Navigation

Beschreibung: Die Funktionen getUniqueLakeIds, getUniqueYearsForSee und getUniqueTypesForSeeYear verwenden die MongoDB-Operation distinct(). Diese wird teilweise mit spezifischen Filterkriterien kombiniert (z.B. distinct("jahr", { see_id: seeId })), um effizient Listen eindeutiger Werte direkt aus der Datenbank zu generieren. Diese Listen dienen der dynamischen Erstellung von Navigationslinks (z.B. für die hierarchische Statistikansicht) und der Befüllung von Dropdown-Optionen im Frontend. Dies wird vor allem verwendet, um die unstrukturierten Daten in MongoDB durch distinct nicht durcheinander zu bringen.

Relevanz: Zeigt den Einsatz spezialisierter Datenbankoperationen zur Optimierung der Datenbeschaffung für UI-Komponenten, anstatt alle Daten zu laden und clientseitig nach eindeutigen Werten zu filtern. Dies ist eine effiziente Methode der "Datenverwaltung".

Dateien/Routen:

- src/lib/server/db.js (Funktionen: getUniqueLakeIds, getUniqueYearsForSee, getUniqueTypesForSeeYear)
- src/routes/fangstatistik/+page.server.js (nutzt getUniqueLakeIds)
- src/routes/fischfangstatistik/[see_id]/+page.server.js (nutzt getUniqueYearsForSee)
- src/routes/fischfangstatistik/[see_id]/[jahr]/+page.server.js (nutzt getUniqueTypesForSeeYear)
- src/routes/fang-eintragen/+page.server.js (nutzt getUniqueLakeIds)

4.3. Spezifische Datenzugriffslogik für das Single-Document Fischlexikon

Beschreibung: Die Datenbankfunktionen für das Fischlexikon und die Fang-Eintragen Route (getFischForCategory, getAllFischartenNamen, unterstützt durch den Helfer getFischlexikonDataDoc) sind auf die Abfrage eines einzelnen MongoDB-Dokuments zugeschnitten, in dem die gesamte Lexikoninformation mit Kategorien als Top-Level-Keys gespeichert ist. Dies beinhaltet Logik zur Extraktion von Keys als Kategorienamen, eine Case-insensitive Suche nach Kategorien und das Sammeln aller eindeutigen Fischnamen über alle Kategorien hinweg.

Relevanz: Demonstriert eine angepasste Datenzugriffslogik für eine spezifische NoSQL-Datenmodellierungsentscheidung, die über die Handhabung einfacher, flacher Dokumentstrukturen hinausgeht.

Dateien/Routen:

- src/lib/server/db.js (Funktionen: getFischlexikonDataDoc, getFischForCategory, getAllFischartenNamen)
- src/routes/fischlexikon/+page.server.js (nutzt getFischlexikonCategories)
- src/routes/fischlexikon/[category]/+page.js (nutzt getFischForCategory)
- src/routes/fang-eintragen/+page.server.js (nutzt getAllFischartenNamen)

4.4. Integration der Browser Geolocation API zur Standortermittlung

Beschreibung: Auf der Seite "Neuen Fang Eintragen" kann der Benutzer per Knopfdruck seinen aktuellen Standort über die Browser Geolocation API ermitteln lassen. Relevanz: Erhöht den Datenwert der Fänge durch präzise Ortsangaben, angegeben in Längen- und Breitengraden. Die Geolocation API ist ein Interface und kann durch navigator.geolocation aufgerufen werden. Es wird durch die meisten Browser unterstützt und kann verwendet werden, insofern die User die entsprechende Browser-Erlaubnis freigeschaltet haben.

Für die Entwicklung verwendete Dokumentation: https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API

Relevante Dateien/Routen:

- src/routes/fang-eintragen/+page.svelte (JavaScript-Logik für getCurrentLocation)
- src/routes/fang-eintragen/+page.server.js (Form Action empfängt Koordinaten)

4.5. Anbindung der externen Open-Meteo Wetter API

Beschreibung: Beim Speichern eines neuen Fangs wird der durch Geolocation ermittelte Standort sowie dem Datum/der Zeit und dem Fangdatum/Uhrzeit, automatisch Wetterdaten (Temperatur, gefühlte Temperatur, relative Luftfeuchtigkeit, Niederschlag, Wettercode, Windgeschwindigkeit, Windrichtung, Oberflächenluftdruck) von der Open-Meteo API abgerufen und mit dem Fang in der Datenbank gespeichert. Leider gibt es für Seen keinen zuverlässigen Forecast von Open-Meteo. Der vorhandene Marine Forecast beschränkt sich vor allem auf Ozeane/Meere.

Für die Entwicklung verwendete Dokumentation: https://open-meteo.com/en/docs

Relevanz: Bereichert die Fangdaten um wertvolle Umweltinformationen. Die Daten sind für Nutzer im Frontend nicht sichtbar, können aber für die Fischereibewirtschaftung nützlich sein, um das Verhalten der Fische, wie zum Beispiel Jagdfenster, genauer zu erforschen.

Dateien/Routen:

- src/routes/fang-eintragen/+page.server.js (Form Action, fetch-Aufruf an Open-Meteo)

4.6. Hierarische Navigation für Statistiken

Beschreibung: Die Fischfangstatistiken sind nicht nur als eine grosse Liste dargestellt, sondern können hierarchisch durch Auswahl von See -> Jahr -> Typ bis hin zur Detailansicht exploriert werden. Dies wird durch dynamische Routen und spezifische load-Funktionen für jede Ebene realisiert, die gezielte Datenbankabfragen auslösen. Zudem sind die aufgerufenen Daten nicht bereits so zur Verfügung gestellt, wie sie auch angezeigt werden, sondern werden via db.js dynamisch aufbereitet.

Dateien/Routen:

- src/routes/fangstatistik/+page.svelte (Seen-Links)
- src/routes/fangstatistik/[see_id]/+page.svelte(Jahres-Links)
- src/routes/fangstatistik/[see_id]/[jahr]/+page.svelte (Typen-Links)
- lib/server/db.js

4.7. Dynamische Optionen für Dropdowns aus Datenbank

Beschreibung: Auf der Seite "Neuen Fang Eintragen" werden die Optionen für die Auswahl des Sees (getUniqueLakeIds aus der fangstatistik-Collection) und die Vorschläge für die Fischart (getAllFischartenNamen aus der fischlexikon-Collection) dynamisch aus der Datenbank geladen. Die Daten werden also von der Datenbank gelesen und durch den User dynamisch bestimmt, anstatt diese im Frontend manuell direkt auf +page.svelte zu Schreiben.

Dateien/Routen:

- src/routes/fang-eintragen/+page.server.js (load-Funktion)
- src/routes/fang-eintragen/+page.svelte(Verwendung der geladenen Daten in <select>, <datalist>)
- src/lib/server/db.js (Funktionen getUniqueLakeIds, getAllFischartenNamen)

4.8. Robuste Fehlerbehandlung

Beschreibung: Die Anwendung implementiert Fehlerbehandlung sowohl auf der Server-Seite (in load-Funktionen und Form Actions, die error() und fail() von SvelteKit nutzen) als auch auf der Client-Seite (Anzeige von data.loadError und form.message). Formulare behalten bei Validierungsfehlern die eingegebenen Werte.

Dateien/Routen:

- Alle +page.server.js, +page.js Dateien.

- src/routes/fang-eintragen/+page.svelte (Anzeige form.message)
- Alle +page.svelte Dateien, die data.loadError anzeigen.