M7011E Project
Fish Finder

Maxim Khamrakulov (maxkha-3@student.ltu.se)
Axel Vallin (axeval-3@student.ltu.se)

23 januari 2017

Innehåll

1 Introduction

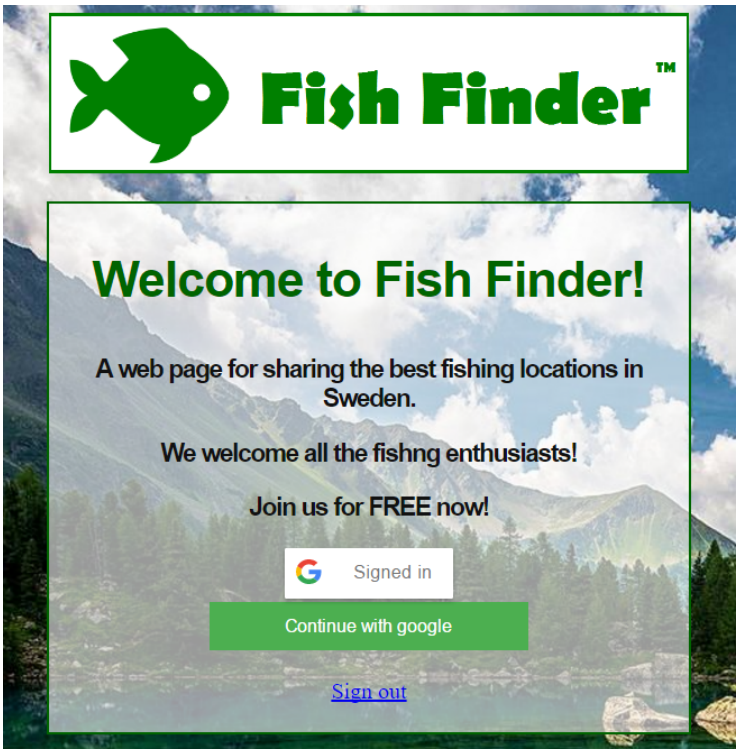
This a report on the project, called Fish Finder ™, which was developed in the second learning period of 2016/2017 as an examination assignment for the course M7011E (Design of Dynamic Web Systems). The purpose of the project was to develop a dynamic web page with certain requirements. The list of those is listed below.

- Backend (server-part) that stores data persistently and efficiently.
- A clearly defined and documented API for communication with the backend.
- Easy to use frontend that allows for upload of data via the web and allows the user to see all data stored in an efficient and "nice" way.
- All communication with the backend should be done via the API.
- History should be handled in the browser!
- A simplified but well documented example application that communicates with the backend to allow for new users to create new types of applications using the backend.
- Security on the right level". It is recommended that a modern security scheme is used where users can login using alternative services like e.g. Google or FaceBook.
- The system should utilize modern HTML5-technologies.
- The final application has to handle binary data uploaded to the server!

The page should also solve a certain user problem, which, for our project, will be described in the section 2. The solution for the problem can be discovered in the section 3. This report was mostly written after development, and describes the methology and technology, used for creating the web system.

1.1 Web page

Fish Finder ™ is a web page for fishing enthusiasts, where they are able to mark good and bad fishing spots on the map for all the other users to see. Users are able to make fishing trips to those fishing spots, by clicking on them and leaving the feedback, such as the total weight of the catch, comment and a grade.



Figur 1: Home page of Fish Finder ™.

We came up with the idea of the page together, because both of us like fishing quiet a lot. And we would like to make a page to improve the life of people with the same hobby as us, namely fishing. It was not that hard to start working on the page, because we had some experiences with web pages before and we roughly knew what features we wanted to have added onto the page.

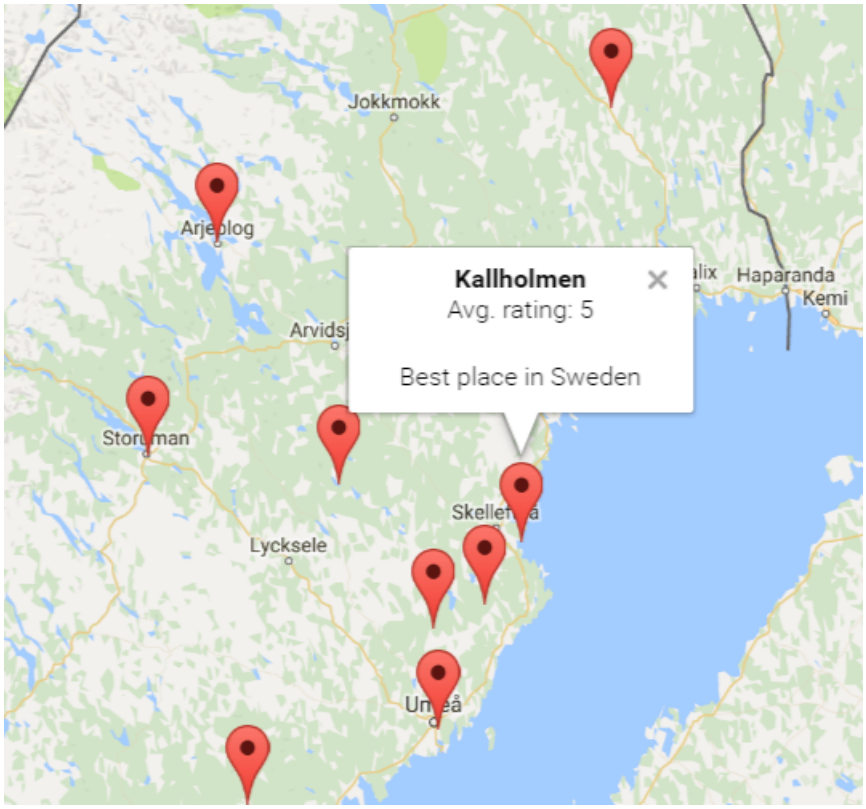
2 User problem

A problem that many fishers face is that they spend an awful lot of time to find good fishing spots. As experienced fishers we know how frustrating it can be to fish in a bad spot. Imagine just standing there by the riverbank for endless hours and not getting anything in return. We would like to illustrate a user problem by introducing an imaginary person called Billy.

Billy is a nice guy, who likes fishing. Billy has a lot of friends, but they, unfortunately, do not like fishing as Billy does. That means that Billy can not ask any of his friends about good fishing locations. He is on his own. Even though Billy is a nice guy, he has some problems with his short temper. He tends to get really angry and frustrated, when he does not get any fish. He would like to avoid that feeling of getting no catch from fishing. He is looking for a solution to a problem of finding good fishing locations, but it turns out to be not that easy to find it.

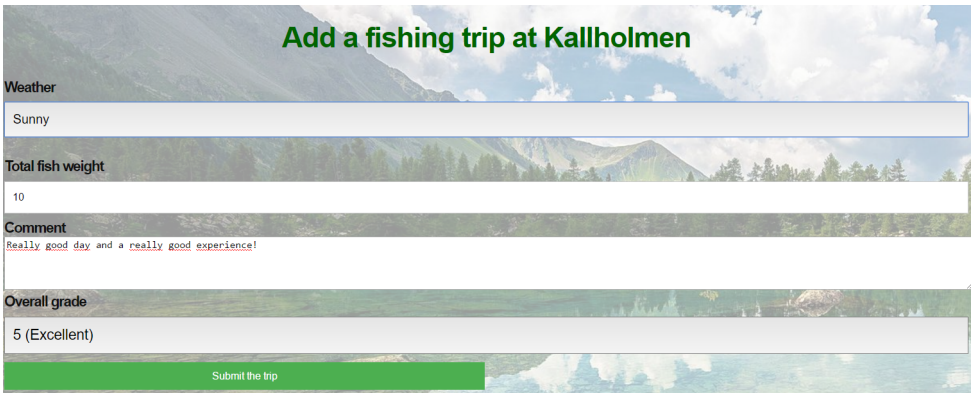
3 Solution

We kept that user case from the previous section in mind when creating our software. Fish Finder ™ is a perfect place for users like Billy. It is a page where Billy can find good fishing spots, based on the user reviews. Once Billy finds something interesting on a map (shown in figure 4), he can click on the location to find more information about the place. He can browse through comments and see if this is the right fishing place for him.



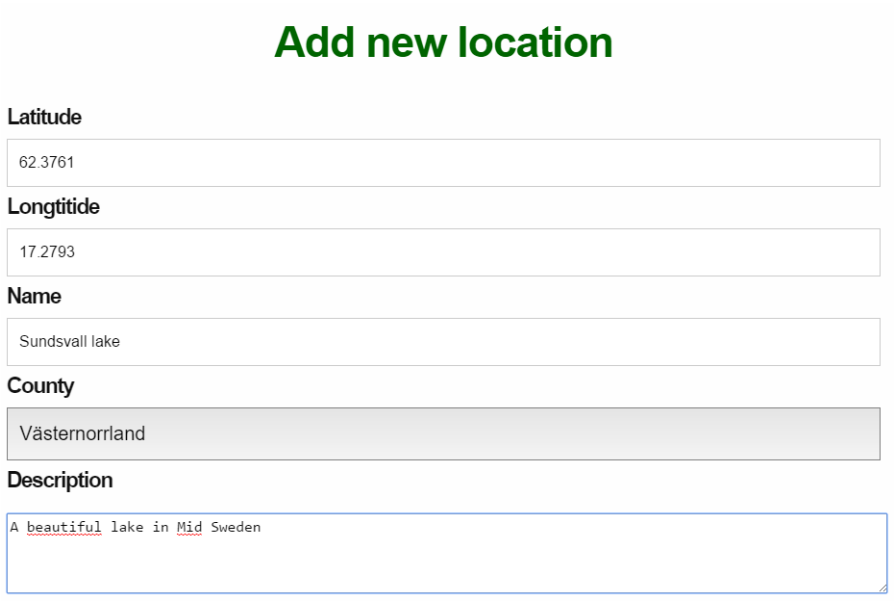
Figur 2: Fishing map.

As mentioned before, Billy is a nice guy. He is very kind and helpful person, who likes to contribute to fishing community. By using Fish Finder ™, Billy can help other fishers by leaving feedback on already existing fishing spots. The user feedback is based on users making, so called, fishing trips. After doing a few hours of fishing at a location, users are free to leave some information about the trip, such as weather, grade, total amount of catch and a comment.



Figur 3: Adding a comment to a fishing location.

Another way to contribute is to do some exploring into the unknown, namely visiting a location that is not shown on the map, doing some fishing there and then sharing the experience from that new spot. Once visiting the spot and doing a bit of fishing, Billy can right-click on a map to add a new location to the database. If the place turned out to be bad, Billy can leave a bad grade and a comment that will prevent other users from making the same mistake, by going there on a fishing trip. If the place is a good one, Billy can share it to everyone, become a popular fishing spot critic and make some more fishing friends.



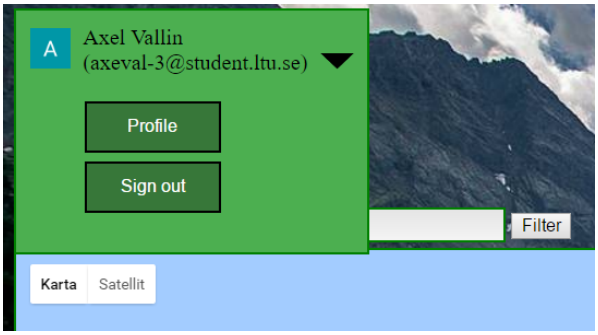
Figur 4: Adding a new location.

4 Design and Architecture

4.1 Front-end

The front-end consist of a login page, main page and location page. The login page is very simple with a Google login button and a button which takes you to the main page if you're logged in. This makes the site very accessible if you already have a Google account. It also has a short information about the page so that the user gets a feel what the site is about and also a sign out button.

Only a logged in user can access the main page. The user can access the profile information as well as a sign out button in the drop down menu in the top left corner. The profile information can easily be changed using the popup window that appears when clicking on profile.



Figur 5: The drop down menu showing profile and sign out.

A big part of the main page is covered by a map with fishing spots represented by markers on the map. Hovering over a marker on the map shows the name of the fishing spot, a short description and the average rating given by other users. Just by looking at the map and quickly checking out some locations a user can get a good idea on what fishing spots are interesting and which to avoid. Right clicking on the map adds a new fishing spot which after a few fields in a popup window is filled in is stored in the database. This includes the first fishing trip done to this spot. This makes it easy to add new locations to the map. If the user wishes to know more about a location before deciding whether or not to go there he or she can click on a marker to visit the location page for that fishing spot.

At the location page you can see all the reviews that other users have written about this location and an image uploaded by the creator of the fishing spot. The reviews include the rating, comment, total weight of the catch and the weather during the trip. Of course this is also the place where you add your own review of the fishing spot. When a review is added to a location it will be displayed on the bottom of the location page and the average rating for that fishing spot is updated.

4.2 API

4.2.1 Describtion

The API is made up of PHP functions that return, add and modify data in the database. When creating a frontend web page these functions are called to access the backend. Below is a list of the functions in the API and a short description on how they work.

userExistent(id)
Checks if a new user should be created.

addNewUser(id, userName, userEmail)
Adds a new user to the database. The arguments are fetched when logging in with a google account before being passed to the function.

getUserInfo(id)
Returns an array containing information about a user corresponding to a certain *id*. The associative array looks like follows. ['email', 'address', 'city', 'biography', 'county_id']

getUserName(id)
Returns the user name corresponding to a certain *id*.

setUserInfo(id, address, city, bio, county_id)
Updates the *address*, *city*, *biography* and *county* fields in the user table where the user with the id *id* is stored.

getLocationRating(county_id, rating)
Returns an array containing the name, latitude, longitude, rating and description of a location that is in the county corresponding to the *count_id* and has a higher rating than *rating*. If county_id is 0 then locations from all counties are returned (it still needs to meet the rating criteria though). Each associative array returned looks like follows: ['name', 'lat', 'lng', 'rating', 'description'].

getCounties()
Returns the rows of all counties.

getCenterMap(id)
Returns the latitude and longitude of the county with id *id*. Use the 'lat' and 'lng' entries in the returned associative array to center the map on that county.

addLocation(lat, lng, rating, county_id, name, desc)
Adds a new location to the database.

addFishingTrip(id, loc_id, comment, catch, weather, rating)
Adds a new fishing trip to the database.

getComments(location)
Returns the user id, weather, catch, rating and comment for all user reviews

about a certain *location* (the location id). Each returned associative array looks like follows: ['user_id', 'weather', 'catch', 'rating', 'comment'].

getImage(loc)
Returns the blob image of the location with name *loc*.

getLocationID(loc)
Returns the id of the location with name *loc*.

getLocationLatLng(lat, lng)
Returns the name of a location based on its *latitude* and *longitude*.

updateGrade(loc)
Updates the grade for a certain location.

4.2.2 Usage of API in the application

Logging in/user management

In order to perform login operations, the *getUserInfo()* function is used, to fetch information from Google login API. If the user logs in into the service for the first time (which is checked by the *userExistent()* function) the user is added to the database by the *addNewUser()* function.

Functions *getUserName()* and *getUserInfo()* are used to display information about the user on the locations-page, as well as in the comment section for the locations.

User info can be changed with *setUserInfo()* function (such as county, address, name etc).

Location related

In order to display the location rating, the *getLocationRating()* is called. The locations can be added with *addLocation()* function. When adding a location, user can attach an image, that shows the location. This image is fetched by the *getImage()* function. API functions *getLocationID()* and *getLocationLatLng()* are used by the Google Maps API to vizualize their position on the surface of our planet Earth.

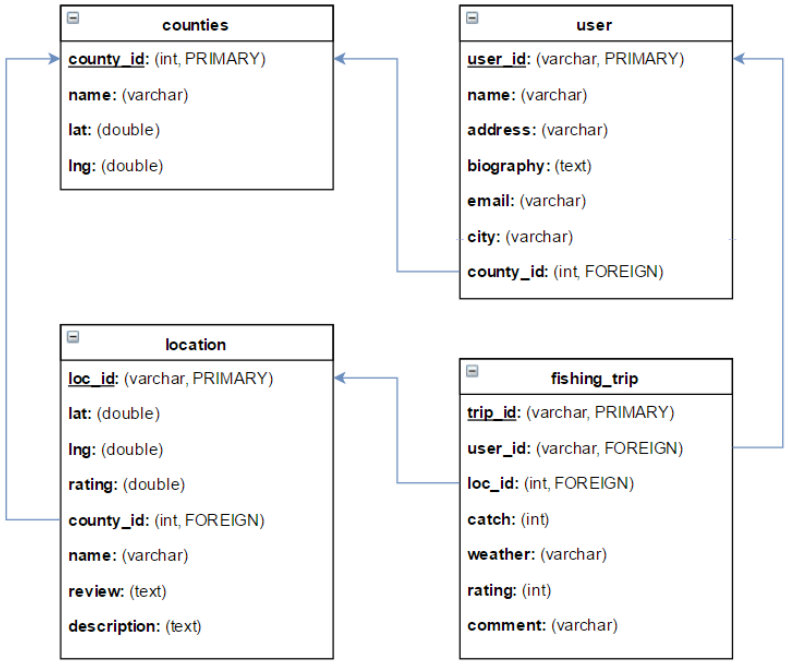
The users are making fishing trips continuously. The API function *addFishingTrip()* is used to add new trips to the database. To fetch the comments from the database, the function *getComments()* is used. When a new fishing trip is added, the overall average grade needs to be changed. This is done by calling the *updateGrade()* function.

Map and filtering

In order to set the center of the map, which is needed for when a user decides to filter the locations to a certain county, the function *getCenterMap()* is used. The county can be selected by *getCounties()* function. As mentioned earlier, the marker positions are retrieved by calling the *getLocationLatLng()* function.

4.3 Database

The database consists of four tables: *counties*, *fishing_trip*, *location* and *user*. The schematics are shown in a figure below.



Figur 6: Database structure.

Counties

This database table is used to store the information about counties in Sweden. It is used by the Google Maps and the sorting functions to sort the fishing locations and show them on the map.

county_id - Primary key of the table. Holds the unique identification value of the

name - Holds the name of the county.

lat - Holds the latitude coordinate of the geographical center of the county.

lng - Holds the longitude coordinate of the geographical center of the county.

Fishing_trip

This database table is used to store the information about the fishing trips, made by the users. It is used to calculate the average grade of the fishing locations

and by the location page, that views the comments, made by users.

user_id - Foreign key of the table, that points to the *user_id* column in the *user* table. Holds the identification number of the user, who went on the fishing trip.

loc_id - Foreign key of the table, that points to the *loc_id* column in the *location* table. Holds the identification number of the location, where the fishing trip took place.

catch - Holds the total weight of the catch from the trip.

weather - Holds the weather from when the fishing trip took place.

trip_id - Primary key of the table. Holds the unique identification number of the fishing trip.

rating - Holds the grade of the fishing trip (from 1 to 5, the higher - the better).

comment - Holds the comment, left by the user, to describe the experiences from the fishing trip.

Location

This database table stores the information about the different fishing locations. It is used mainly by the Google Map object to mark the locations on the map and show the information about them.

lat - Holds the latitude coordinate of the fishing location.

lng - Holds the longitude coordinate of the fishing location.

loc_id - Holds the unique identification number of the fishing location.

rating - Holds the average rating value of the fishing spot.

county_id - Foreign key of the table. which points to the *county_id* column in *counties* table. Holds the identification number of the county, where the fishing spot is located.

name - Holds the name of the fishing location.

description - Holds the brief description of the fishing location.

User

This database table stores the information about the users of the software. It is mainly used for logging in/out.

user_id - Primary key of the table. Holds the unique identification number of the user.

name - Holds the name of the user.

address - Holds the address of the user's home.

biography - Holds the biography of the user.

email - Holds the e-mail address of the user (Google).

city - Holds the city, where the user lives.

county_id - Foreign key of the table. which points to the *county_id* column in *counties* table. Holds the identification number of the county, where the user is residented.

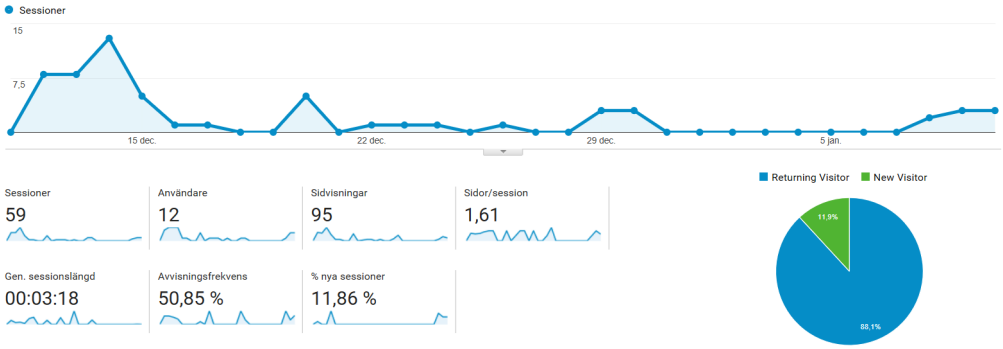
4.4 Technical overview

The web page is written in html, javascript, css and php. This seemed like a standard setup to us and we decided not to dig too deep into alternative ways since we didn't want to risk failiures because of our inexperience. When creating the frontend a few new functionalities introduced in html5 was used. The 'nav' was used to create a drop down menu to access the profile information and the signout button. This makes information that does not need to be displayed all the time not take up too much space on the screen while still being easily accessible. To make sure input fields in the forms were filled in a 'required' attribute was set on those to make it more user friendly by preventing users from forgetting some fields.

A php API was made to communicate between the frontend and the backend. This is done to avoid writing calls to the database in the code for the frontend. The backend consists of a MySQL database that stores the all data used by the web page. The API sends SQL queries to the database and returns information if necessary which can then be used by the frontend. The MySQL database was chosen because it seemed like a good solution for storing and retrieving the data we were going to deal with.

The project is deployed on *lamp.skola.skelleftea.se* where we both earlier have made some simple web pages. We used the editor *Notepad++* with the plugin *NppFTP* to connect to the server and later uploaded everything on *GitHub*. This was also something we had used earlier and we decided to use it again so that we could get started as soon as possible.

Google has a bunch of useful API:s that can be used when creating a web page. In this project the *Google login* and *Google maps* API:s are used. The initial design of the web page consisted of a map where fishing spots were going to be shown and *Google maps* seemed like the most logical choice for achieving that. Therefore it was also decided that Google was going to be used when creating the login function of the site. It is also possible to see the traffic on the site using *Google analytics*



Figur 7: Information and graphs that can be seen on the Google analytics page.

4.5 Security

To provide som safety during the login the Google API for Google login was used. This provides the same safety as any other web site where you can login using your Google account which seemed sufficient for this project.

A user with some knowledge how an SQL database works can attempt to send parts of an SQL query through the input fields on the website to access or remove information in the database. If for example this input is just added at the end of an SQL query it can then become a query which affect the database very differently than intended by the developer. To protect the database from these SQL injections the API uses prepared statements. This creates a template that is first sent to the database. The final SQL query then needs to match this template which prevents any parameters sent to the API function from manipulating the query since the template doesn't depend on these parameters.

5 Workflow

The development was divided into four steps, also called sprints. The group tried to follow Agile Software Development strategy, during the project. That said, the tasks were split between us, two group members, so that several different modules and functions were developed simultaneously, independent from eachother. Some larger tasks were completed with some cooperation. It was important to follow the deadlines, because as the development goes on, the tasks start to become more complicated and depend on eachother. Some tasks require certain modules to be completed, in order to properly be able to start the development. Both of us worked with Agile Software Development before, during both D0020E (*Project in Computer Science*) and D0018E (*Database Technology*). Back then we used an online service called *GravityDev*, to handle all the stories and tasks for sprints. Once we started working on Fish Finder, we wanted to access the service but, unfortunately, it was gone. That made us go all old-fashioned and use a pen and a paper instead. As mentioned before, we divided the development into four steps. The complete description of the sprints is provided in subsections 5.1 to 5.4.

5.1 Sprint 1

The first sprints purpose was planning and preparations for sprint 2. This sprint lasted a rather short time, namely less then a week. The tasks for sprint 1 can be seen in a bullet list below:

- **Architecture design** - decide which database we should use and what programming languages will be used.
- **Recall from previous courses** - recall information about HTML, CSS, SQL and PHP.
- **Learn JavaScript** - get a basic understanding in how JavaScript works and how it's syntax is like.
- **Set up server and a basic web page** - Set up a web server and a MySQL database with phpMyAdmin.
- **Git** - Set up a repository at GitHub.

5.2 Sprint 2

The second sprint was all about getting started with the project. Things such as a basic design was created along with integration of Google API's. This sprint lasted until the short presentation we had in November. The complete list of tasks for sprint 2 can be seen in the bullet list below:

- **Basic front page** - create a home page for the web site, that will hold the login form and some information.
- **Google login** - integrate an API for Google login.
- **Basic location page** - create a location page, which will list all the fishing locations in the future.
- **Google maps** - integrate an API for Google Maps, which will be used to visualize the fishing locations.
- **Basic design** - make a simple CSS design for the web pages.
- **Database Design** - design of the database structure.
- **Presentation** - make a power point and get other stories ready to be able to present the to our classmates.

5.3 Sprint 3

The third sprint was by far the longest one and lasted until the final review in December. During sprint 3, the group made the most of the progress. During that sprint, the login page was completed, along with the locations page and the page, describing each fishing spot. An API was created along with improved web design.

- **Database storage for Google users** - new users should be able to be added to a database and have some personal info added to them.

- **API** - add API functions for handling the database queries (for insertion, deletion and modification of data), as well as the functions for maintaining the database connection.
- **Marker retrieval for Google Maps API from Database** - the markers should be retrieved from the database via an XML file and the shown on the map.
- **Fishing spot sorting** - add functionality for sorting the fishing locations by grade.
- **Improve the design** - mainly add some HTML5 functionality to the design and improve the visuals.
- **Add location page** - add a page where users can browse through reviews and grades on each and every fishing spot, as well as leave their own comments.
- **Adding new locations** - add possibility to add new locations by right-clicking on the large map.

5.4 Sprint 4

The last sprint lasted over Christmas holidays and was used to further improve the app, as well as address the issues, appointed at the final review in December.

- **SQL injection** - to prevent SQL injections the API now uses prepared statements.
- **Images** - a user can upload an image when creating a new location which is displayed on the location page for that location.
- **Bug fixes** - a couple of bugs on the frontend as well as the backend were fixed.

6 Reflection

We both have done some simple web pages earlier but at that time we only used very basic html, css and php. Working with this project we had to improve our skills in all those as well as learning how to use javascript. All this took some time we'd rather have spent improving the design and functionality of the project.

When working on a web project next time we have a lot more knowledge and know some things we should avoid which saves a lot of time. A couple of times during the project we came up with some solutions which seemed good at the time but later realized they maybe weren't that great. Some of them were easy to change but sometimes other parts of the project depended heavily on these things.

Some things added late in the development does not have a great representation on the web page. For example since uploading images was not possible when most of the entries to the database was created, only a couple of locations actually has an image to show. The fishing spot in Gällivare is one of those that was added later and have an image.

7 Future Work

Once the project was ready for hand-in, we took a brief moment to discuss what existing features could be improved and what new features could be added to the software. After the discussion, we came up with the list of improvements, that could be done to the software:

- Better webpage design.
- Improved error handling, such as improved handling for wrong file format (for location image), empty file and checking the "total catch weight" input field.
- Support for more browsers. Right now the web site is only tested for Google Chrome which means that some functionalities probably won't work on other browsers.
- An admin user should be able to delete locations and reviews to keep the site clean.

Referenser

[1] FishFinder,
`http://lamp.skola.skelleftea.se/~afo1009/FishFinder/`

[2] Project folder on GitHub,
`https://github.com/maxkha-3/FishFinder`

[3] W3Schools,
`http://www.w3schools.com/`

[4] PHP Manual,
`http://php.net/manual/en/index.php`

[5] Google login API,
`https://developers.google.com/identity/sign-in/web/devconsole-project`

Appendices

A Deployment

In order to run your own version of the software, or contribute to it, one has to follow those steps, described down below.

1. Go to the project's Git (`https://github.com/maxkha-3/FishFinder`) and download the source code.
2. Upload the code to a compatible server of your choice.
3. We used MySQL relational database, so we strongly recommend to run the backend on a MySQL database platform.

4. Copy the code from *create_dbstructure.php* into the SQL query window and run it, which will create the database structure, specifically tailored to run well with the API. To see more specific information about the database, we advice to examine section 4.3 carefully.

5. Edit the file, called *db_info.php*, by inserting your database name, username and the password.

6. Examine the documentation carefully and go through the code.

7. Final step: Make changes to the code as you like.

Detailed instructions on how to use the API to create a similar application are provided in section 4.2.2.