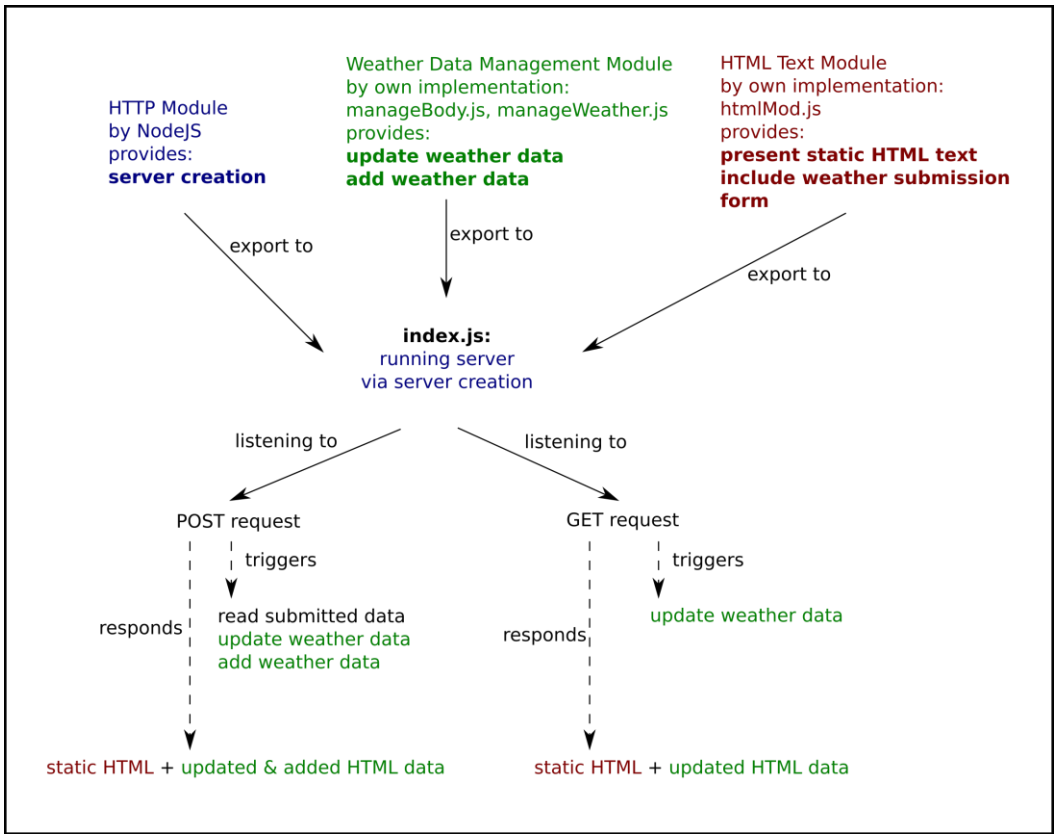
Technical Approach / Solution with respect to content and concept

In the concept phase it was described that a web server is used to exchange data with a single page application. The single page application is represented by a module called htmlMod.js. All necessary HTML text and a weather submission form is included there. Another module was created by own implementation to provide the functionality to update and postprocess weather data on the server side. Furthermore the http module as part of the standard NodeJS package is used to create the server. All three modules were exported to an index.js file which represents the initial point of the application. The following picture shows the approach for reference.



The running server initialized in the index.js file listens to POST and GET requests. A GET request is initiated by any sort of client, in this case by the mobile devices of pilots via browser. The weather data will be updated and the response consists of updated HTML weather data embedded in static HTML text, which is then rendered on the web page. The POST request is triggered by a pilots weather submission. Beneath the weather update which is also performed in the GET request the submitted data needs to be read and formatted. Afterwards the weather data can be added to the existing weather data including averaging of weather properties for different weather areas. The functions for

adding and updating of weather data are provided from the data management modules in manageBody.js and manageWeather.js. The respective functions coming from other modules are shown in the picture as well. Apart from the event driven update triggered by accessing the page or posting weather data it is demanded that pilots always look on up to date weather tables in the app. Therefore an automatic client based update is included in the single page app implementation by using reload functions via setting time outs. In the end it triggers every two minutes a further GET request which is handled in the same way as described above.

Lessons Learned

By using pure NodeJS for handling requests it was experienced how to read data directly from request bodies. This generic way of capturing data from the request body without the usage of any further middleware packages, gave a good understanding of how client server communication is working. However an increasing functionality of the application will result in more complexity for handling all different request scenarios. For future extension of the app the usage of the express library or similar frameworks should be considered.

The implementation also showed an efficient way of directly use data structures of JavaScript without the need of any external database to store weather data. As the weather data submitted by pilots will only by useful for real time usage there is no need for a long time storage. However the data would be gone in case of a server shutdown. An increased functionality based on long term server data would require to include other database frameworks.

The automatic update of weather data is triggered on a client side via a reload function. Here it could be questioned if a server based data update would be more efficient and the client only requests up-to-date data instead of initiating the update itself. Basically there are a lot of possibilities to adapt the functionality on client and server side for different use cases. Experiencing these possibilities and knowing that the choice of architecture is crucial for efficient usage of the application it is necessary to have a detailed focus on early design stages in software development.