1. **High Level Architecture, Database Organization**

This section describes the system architecture and technology stack of fuldaflats.de. It contains the most important software components, frameworks, libraries and development tools that are used in the development process.

**General Architecture Purpose**

The general purpose for our system architecture is to create a loose coupling between the web client in a user’s browser and our server sided program code. We aim to achieve this by using a RESTful web service architectural style. Our server sided code focuses purely on business logic, authentication, authorization and database connectivity. It does not process any kind of HTML templates. Our whole HTML, CSS and JavaScript code will be delivered as static files to the client. Interactivity is created by using client sided JavaScript code.

Client sided JavaScript is used to manipulate the user interface, react to user input and to load data from the server asynchronously using AJAX and JSON technology. Thus, we aim to create a 3-Tier architecture with a database, a thin server layer and a more powerful client.

Through this we get the advantage that we can easily split our team in frontend and backend developers that interfere less with each other’s work and connect to each other via a predefined, standardized HTTP interface. This also makes it very easy to provide test data in JSON format and makes the application more stable and user friendly against server errors (HTTP 500), since this makes it more unlikely that a user sees them.

**Server Foundation Software**

The overall architecture goal for fuldaflats.de is to create a multi-tier web application that uses a REST-Webservice and AJAX technology as main data exchange interfaces. We use a **Linux/Debian** virtual machine that is hosted on the **Microsoft Azure Cloud** as a server for our project.

The data-tier of the application is handled by a **MySQL Database** which can be administrated by the phpMyAdmin user interface.

**Server Sided Programming**

The application / logic tier uses **Node.js 7.0[[1]](#footnote-1)** as technology platform to enable **server-sided JavaScript** coding. Node.js itself is quite a bare software platform, but it can be extended through the package-manager **npm[[2]](#footnote-2)**. Node.js modules that are mandatory for the fuldaflats.de project are **CaminteJS[[3]](#footnote-3)** as an ORM library for database access and **Express[[4]](#footnote-4)** to create RESTful web service endpoints and start a web server within node.js (Those two are comparable to JPA and JAX-RS in Java EE).

**File Upload**

Uploaded image files will be stored in the .png image format directly on the server’s file system (contrary to storing them as BLOBS in the database), because that makes it much easier to backup text-based data in the database and to provide demo data. When providing demo images, they do not need to be stored in the database first to be used. To store the images on the file system, the Express module **multer** will be used, which is capable of handling *multipart/form-data* HTTP requests that contain uploaded images. To prevent the server from a user’s spam, an image size limit of 5MB per file will be programmed. Furthermore, a single flat offer is only allowed to contain up to 7 images.

There are also plans to allow users to upload one video file per flat offer which interested users can then stream from the server. However, we classified this feature as prio2 and it would eventually be developed as a separate web service function.

**Search Algorithm**

The search functionality is one of the key features of fuldaflats.de. It is very prominent on the home page of the application and is refined in the result overview. Since our application only focusses on the area around Fulda, a full-text search of different search criteria is not that necessary (for example to enter a city to search in). Instead, we have a wide variety of predefined search criteria for the user to choose. This includes the apartment type, the distance from the Hochschule Fulda, the price, the size of the apartment and many more (see story boards for more details). While the search function on the home page only lists the most important criteria, a detailed search mask is also available on the results page.

On the backend, this search uses query functions that are included in CaminteJS. The algorithm will first search for the criteria that are chosen by the user and will put the results into a list. This list is then ordered by the creation date of the offer, but the user can also choose another sort criteria in the detailed search mask.

**Client Sided Programming**

For the web client, we use **jQuery[[5]](#footnote-5)** as a standard library to enhance browser APIs, as well as some small JavaScript libraries for handling user input and server connectivity (for example **KnockoutJS[[6]](#footnote-6)** for lightweight DOM-Databinding). We also use **Twitter Bootstrap[[7]](#footnote-7)** as a presentation framework (mainly for its CSS) for responsive web design.

**Map Data**

For showing Map data to the users, we want to use the library **Leaflet.js** which enables interactive maps from different map providers in the browser. We will use **Open Street Map (OSM)** as a free, open source map provider. Geolocation data is stored as float-valued latitude and longitude coordinates. To get the coordinates for a given street name and house number, we want to use the **Open Street Map** lookup service **Nominatim**. We will query Nominatim server sided upon the creation of a flat offer.

**Development Tools**

The fuldaflats.de project uses **git[[8]](#footnote-8)** as a source code management system, the code is hosted as a **private repository on** **GitHub[[9]](#footnote-9)**. GitHub issues and milestones are also used for project management and team communication. We use **Visual Studio Code[[10]](#footnote-10)** as development environment for web applications and node.js (which is not Visual Studio 2015, but rather just a free, enhanced text editor tool from Microsoft).

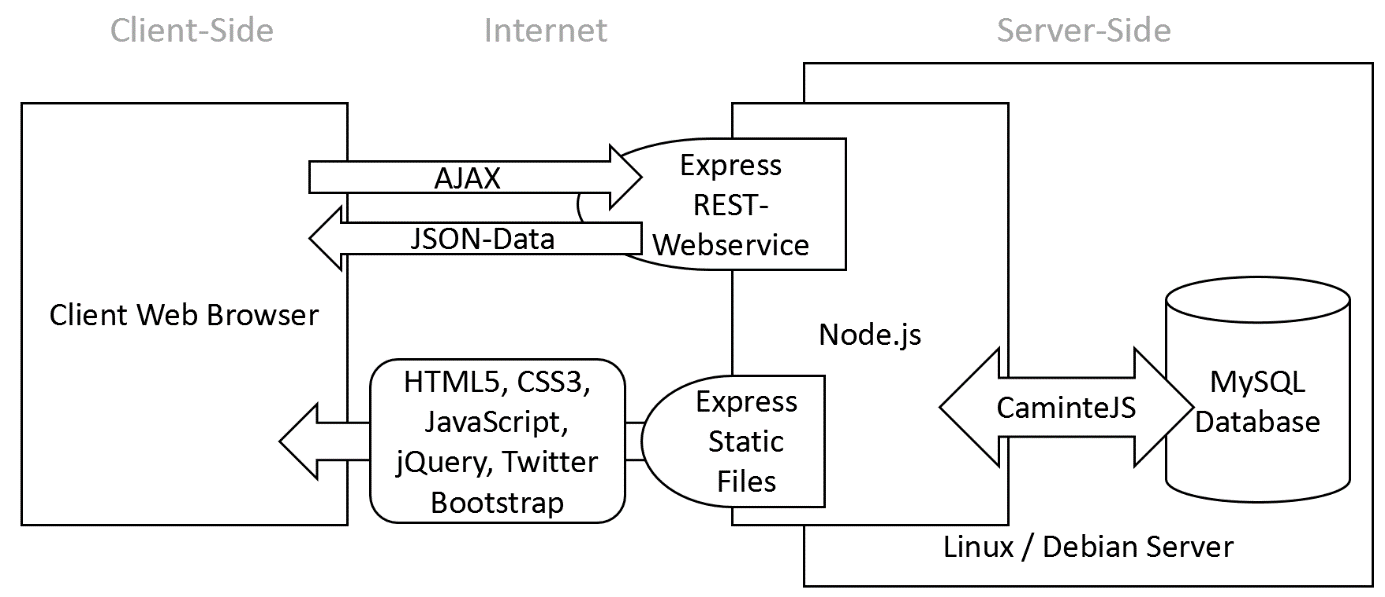
**Compatibility**

The final product will support and be tested on the following Browsers:

* **Google Chrome (Version 54.0)**
* **Mozilla Firefox (Version 42.0)**
* **Apple Safari (Version 10.12)**

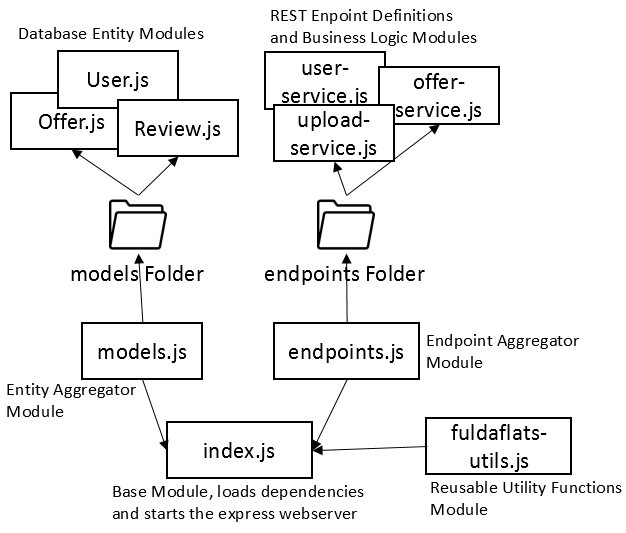
1. **High Level UML Diagrams**

**System Overview Graphic**

****

**Node.js Module Overview**

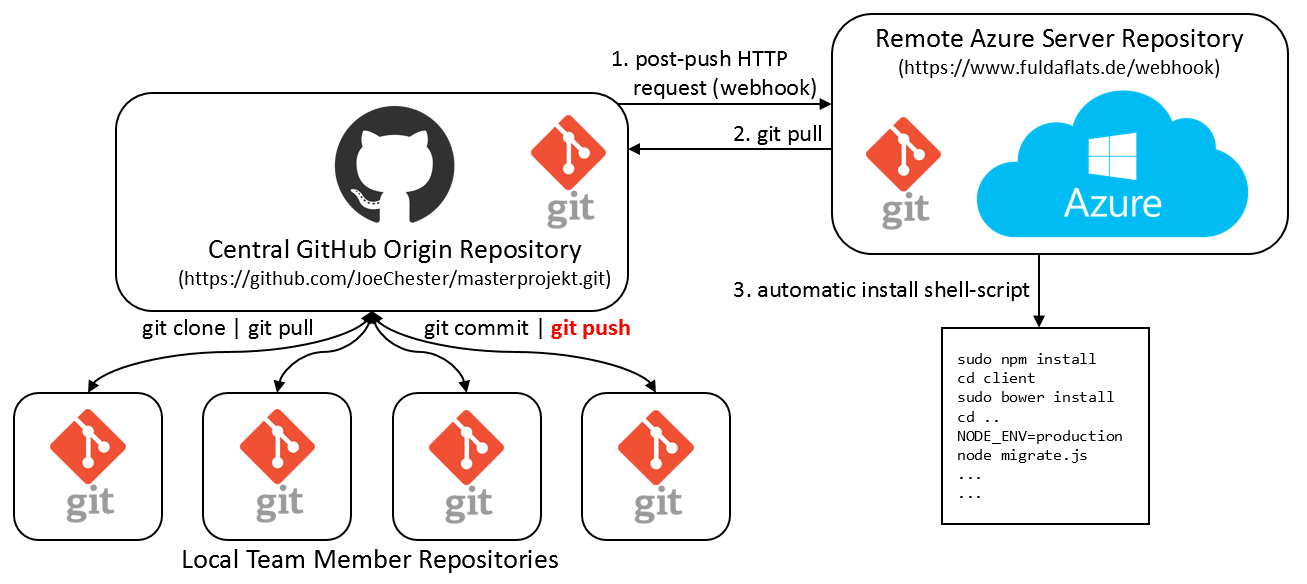
Instead of classes in conventional object oriented programming, Node.js mainly uses modules to divide program code into self-contained parts. The following diagram gives an overview of how the codebase of our server side will be organized.



**Deployment**

The deployment process to get our application running on the Azure cloud server is fully automated by incorporating GitHub’s webhook feature. We configured our GitHub repository in that way that every time a developer pushes something into the GitHub origin repository, GitHub automatically sends a specific POST http request to our server that runs on Microsoft Azure. This POST request is handled by a little node.js script that checks the content of the request for a secret passphrase that we configured on GitHub and runs a shell installation script afterwards if the passphrase is correct.

This shell script pulls the latest commits from the GitHub, installs all dependencies that are configured in npm, migrates the database, inserts demo data into the database and restarts the server application.

****

1. <https://nodejs.org> [↑](#footnote-ref-1)
2. <https://www.npmjs.com> [↑](#footnote-ref-2)
3. <http://www.camintejs.com> [↑](#footnote-ref-3)
4. <http://expressjs.com> [↑](#footnote-ref-4)
5. <https://jquery.com> [↑](#footnote-ref-5)
6. [http://knockoutjs.com](http://knockoutjs.com/) [↑](#footnote-ref-6)
7. <http://getbootstrap.com> [↑](#footnote-ref-7)
8. <https://git-scm.com> [↑](#footnote-ref-8)
9. [https://github.com](https://github.com/) [↑](#footnote-ref-9)
10. <https://code.visualstudio.com> [↑](#footnote-ref-10)