Design Specification
Note-ify

Group 12: Darin Barth, Matt Martin, and Priamwad Poudel

Date	Changes	Version
10/6/17	Initial Document	1.0

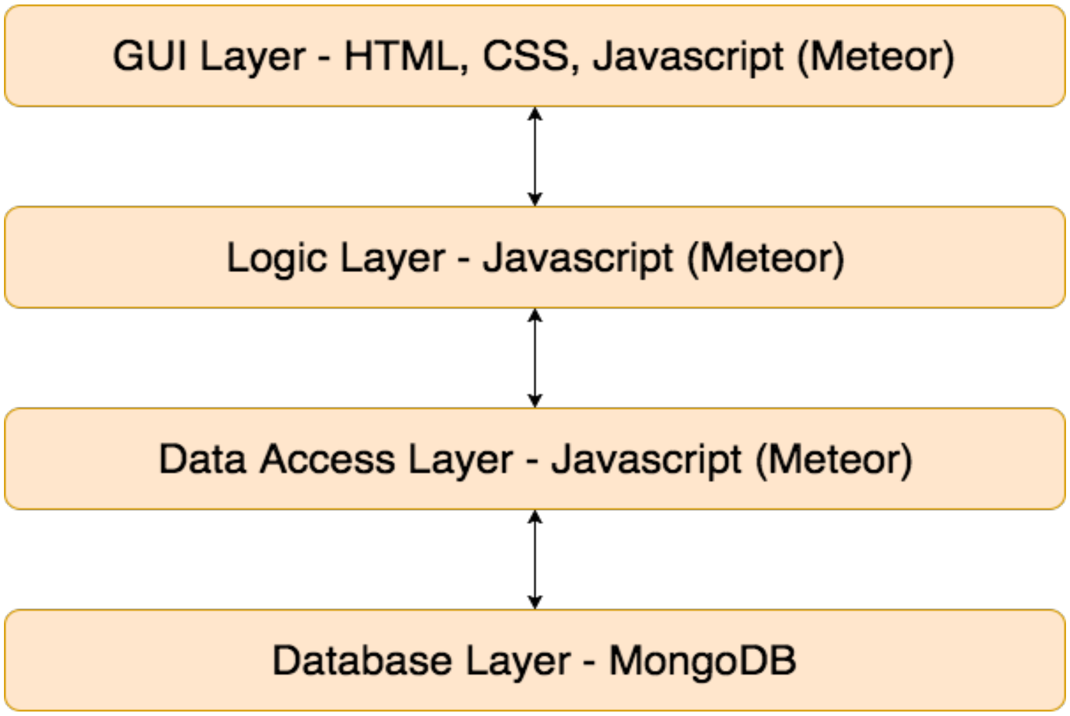
Table of Contents

1. Introduction

The purpose of this document is to demonstrate and explain the high-level architecture and entity relations for the Note-ify web application. This document will include architecture and entity relation diagrams showing the relationship between the different elements of the system in addition to the relationships between corresponding tables in the database. The intended audience of this design document is software engineers and software architects who will be implementing the project described in the following sections.

2. Architecture

2.1 Introduction


GUI Layer - HTML, CSS, Javascript (Meteor)
Logic Layer - Javascript (Meteor)
Data Access Layer - Javascript (Meteor)
Database Layer - MongoDB

The high level architectural design of the system is a layered model and the lowest level is a MongoDB database curated by Meteor and used to store permanent information. Due to Meteor's native integration with MongoDB, most database interactions through the data access layer can easily occur with a simple Meteor function. In a Meteor webapp, the Logic, GUI, and Data Access layers are tied tightly together with the majority of the code being written in Meteor-style Javascript, the exact implementation of which will be described in further detail in later sections.

2.2 Modules

2.2.1 Database Layer

The responsibilities of the database layer include maintaining all persistent data for the system as well as defining the relationships between this stored data. The system will use a MongoDB database due to its native integration with Meteor.js. The data relationships are described in more detail below in section 3.1.

2.2.2 Data Access Layer

This module is responsible for managing data as it goes back and forth from the database layer. The primary responsibilities of this layer include querying the database and converting database data into Meteor objects that can be manipulated by the Logic and GUI Layers, with any permanent changes to objects being saved in the database. Due to Meteor's use of Distributed Data Protocal (DDP or "data on the wire") via websockets, the HTML displayed in the GUI is updated in real-time whenever a relevant server-side MongoDB collection is altered in the database.

2.2.3 Logic Layer

The primary responsibility of the Logic Layer is to perform all necessary business logic involving the Meteor models in the system. This module will supply the desired information to the GUI layer to display the desired information and appropriate entries to the user. As with the data access layer, this will also be implemented with Meteor in Javascript.

2.2.4 GUI Layer

This is the primary layer that the user will interact with. It is responsible for generating the user interface, and providing the user with the ability to interact with the system. It is in this layer that the user will be able to login to the system to access their entries.

A combination of HTML, CSS, and Javascript (including MeteorJS and Blaze) will be used in this layer to reactively generate user pages and allow for smooth user interaction for both desktop and mobile users.

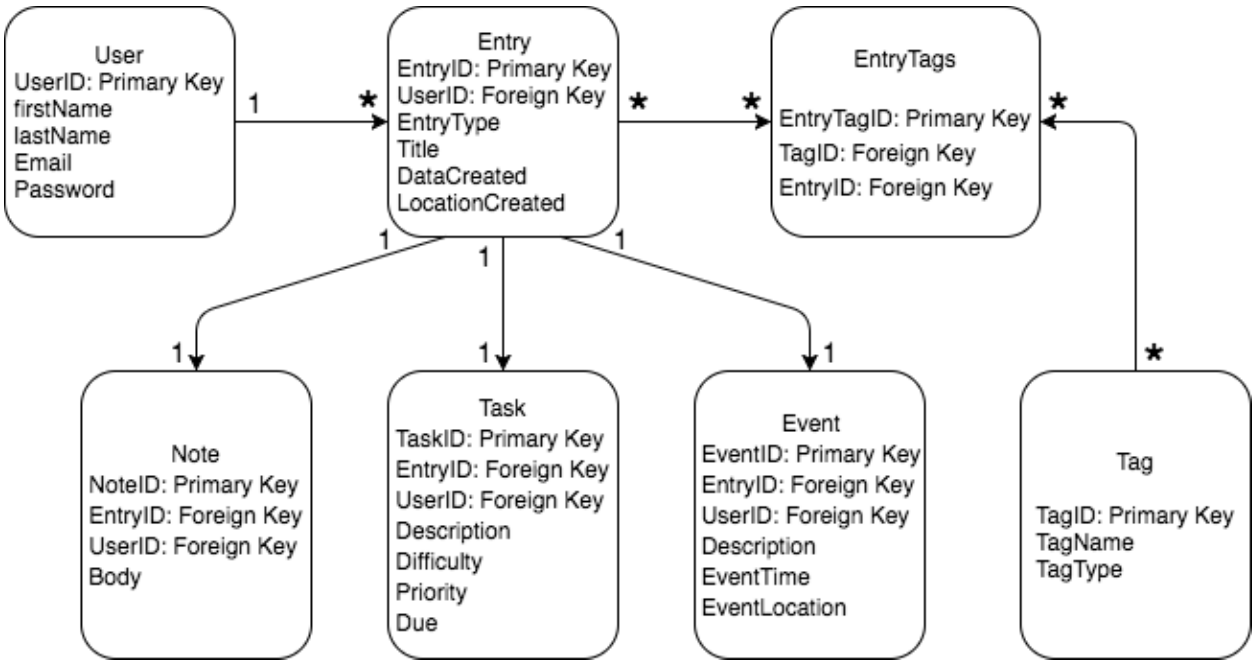
For the first phase, the GUI will consist of a simple login page, followed by a user's home page where they can view and search a chronological list of their entries, as well as create, edit, and delete entries via a menu. In the future phases, a more advanced GUI will be developed allowing for more complex filtering and searching as well as a calendar view mode for the user to see their entries overlaid on a calendar.

3. Class Diagrams

3.1 Data Table Classes

The system will be using MongoDB to hold any persistent data. Persistent data includes all information on users, classes, and the relations between classes. Below shows the database in tables with lines dictating the relationships between the different tables.

3.1.1 Schema



3.1.2 Schema Information

The database is organized in tables and columns as described below.

User: Holds the data for a single user of the system including information such as first name, last name, email, and password.

Entry: This table holds the data for every entry that a user creates. One user can have many entries and the entry class shares a one to one relation with a choice of three specific entries. These three are Note, Task, and Event. The entry table also shares a relation with Tag through a relation table named EntryTags. This table includes the data EntryType, Title, DateCreated, and LocationCreated.

Tag: This table holds the data for the tagID, TagName, and TagType. This is connected to the EntryTags relation table with a many to many relationship to EntryTags

EntryTags: This table holds the relationship data to relate the tables Entry and Tag.

Note: This table holds the data used for every entry class specific to the Note section. This holds a one to one relation with the Entry class. This table includes the data Body.

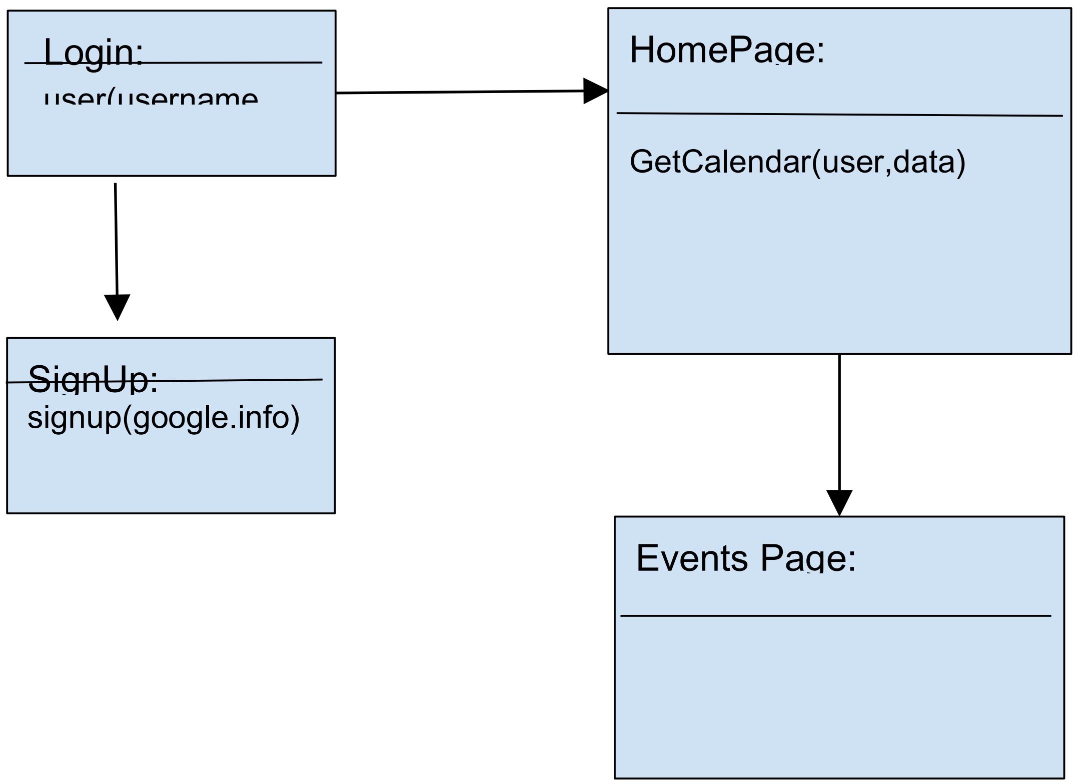
Task: This table holds the data used for a task specific entry class. This holds a one to one relation with the Entry class and includes the data Description, Difficulty, Priority, and Due.

Event: This table holds the data used for an event specific entry class. This holds a one to one relation with the Entry class and includes the data Description, EventTime, and EventLocation.

3.2 Class Information

Classes will be implemented in the form of collections, which in Meteor automatically correspond directly with the appropriate tables of the MongoDB Database. These collections will replicate the database schema as defined above in section 3.1, for example a user collection will be implemented with variables for UserID, FirstName, LastName, Email, and Password. These collections are duplicated in a MiniMongo database serving as a temporary cache on the client-side of the application. This side of the application communicates with the server-side via publication and subscription calls that synchronize the database with the GUI.

3.3 GUI Layer



The GUI layer consists of four pages which the user will be able to interact with. The first page, Login, is shown when the user access our website. Here the user will have two main options, login or create an account. If the user is logging in, his credentials (username and password) will be verified with our database. However, if the user is signing up, then he is prompted to a new screen.

The SignUp page will allow the user create an account. Using an API, the system will allow users to use their social networking accounts instead of creating a new account for accessing the system. However, if a person chooses not to use a social networking account, then he or she can use the default sign up module. Once a person signs up , their information is then saved to the database for logging in next time.

After a user has successfully logged in, he or she is greeted by the home page. The home page is split into three sections: entries display (either calendar or list view), profile, and search/filter. All three sections will be talking with the data layer in order to ensure user data is retrieved and updated appropriately in the GUI as soon as it is modified. For example, the calendar and entry lists will immediately display the corresponding entries associated with each setting as specified in the search/filter as soon as the filter is changed. There is also an edit button which allows the user to create or alter titles, entries, and tags. If the user clicks the new entry, he or she will be taken to the new entry page. The new entry page first requires an entry type to be input (either Note, Task, or Event for Phase 1), and then the form will auto-populate with the appropriate fields to be filled in. The edit button triggers a page similar to the new entry page, except with the previously entered fields auto-populated.