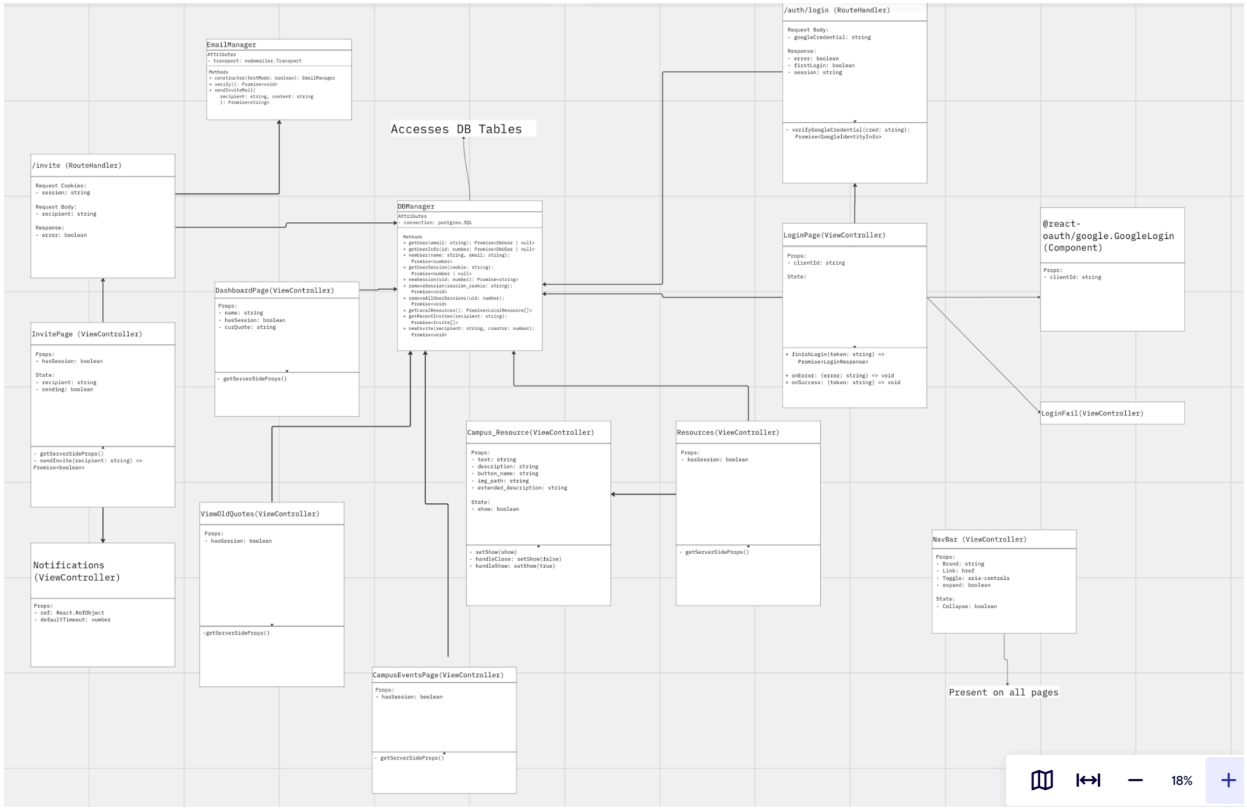
Design “Class” Diagram

See *diagram here on Miro*

Major Components



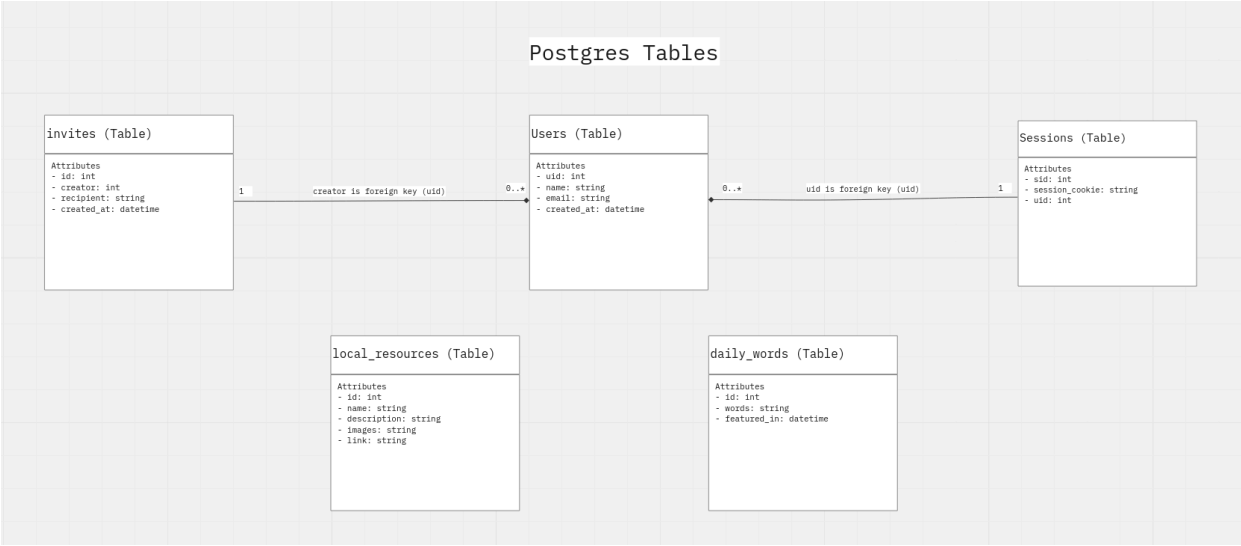
(View an enlarged version on Miro)

The diagram shows the relationship between components, page components, API route handlers, and the DBManager singleton within UMaine Together. Standard components are functional React components which typically display application data that has been passed as props. Page components are special React components used by the NextJS framework to define individual pages within the broader web application. Page Components are able to fetch data using the getServerSideProps method. This method will be executed during server rendering on initial page load, or it will be executed by an API request (and serialized) during client-side page transitions. These different use cases are handled by NextJS automatically. API route handlers are also run by the NextJS framework in response to HTTP requests under the /api/ path. All API routes make requests to the database using DBManager, receive properties using an HTTP post with a JSON-encoded body, and return data also using JSON serialization.

The DBManager is visible in the center of the diagram. It implements a number of asynchronous methods which wrap around PostgreSQL queries (see the Database Tables section below). Only one instance of the DBManager will be created per instance of the UMaine Together server. It is lazily initialized by the first user of the getDbContext function, which will typically be a getServerSideProps method on one of the main pages. One important DBManager method is getUserSession, which checks for a session cookie present in requests to either an API handler or a getServerSideProps on a page. All page components or API handlers which make database requests have an arrow on the diagram connecting to the DBManager.

The other major server-side component is the EmailManager class. This class is also lazily instantiated as a singleton. It is in charge of sending email via SMTP for the invite and check-in functionality of UMaine Together. It implements a number of asynchronous methods for sending mail. So far, this component is only accessed by the handler for the /sendinvite route. However, the class can also be instantiated in “test mode”, in which case the backend will only write messages to an in-memory queue instead of over an SMTP connection. This allows automated testing of components that use the manager.

Database Tables



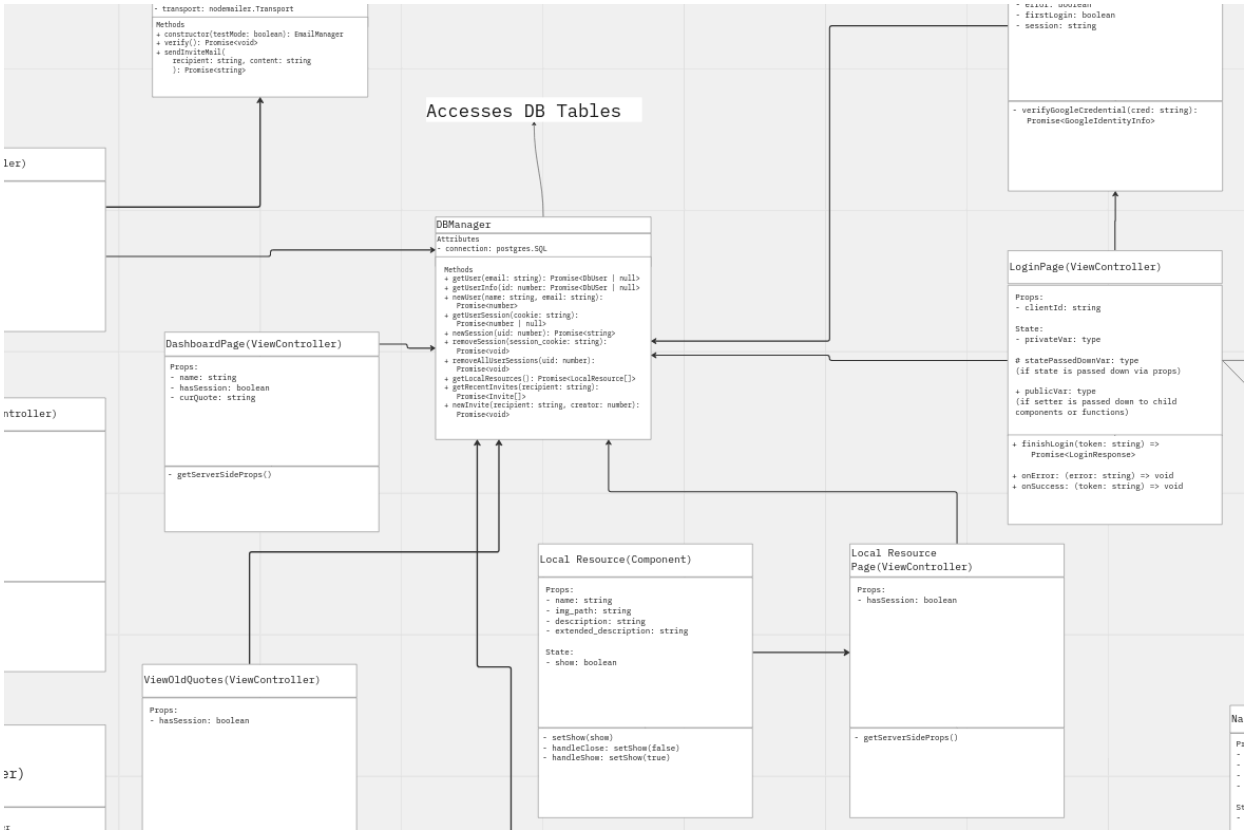
The state of UMaine Together is stored in a PostgreSQL database. This schema of this database marks the ultimate interface between the server code and the actual data. The individual tables are represented as the components in this part of the diagram. The attributes represent columns in the database, with a simplified version of the SQL types (in particular, varchar and text fields have both been labeled “string”). In the actual application, most of the code calls methods on the “DBManager” object, a wrapper over the database. This wrapper allows the database to potentially be given a new schema in the future, or even changed out for a different provider.

Several of the tables reference each other using foreign keys. For instance, the invites table, which tracks email invites sent to potential users, uses the “creator” field and a foreign key to reference the user who originally created the invite. The foreign key relationships are shown by the arrows in the diagram. Many of the tables also use a “created_at” field, which is automatically given a value of the current time by Postgres.

Design Patterns

UMaine Together uses the Adapter pattern to facilitate loading data into the views. The DBManager singleton handles access to a variety of database tables. In order to query a table, the particulate structure, including the column names, must be known. However, from Typescript classes, it is more convenient to work with defined types. The DBManager handles the translation by providing a series of methods that wrap SQL queries.

The class diagram excerpt below shows a number of connections from page components and api handlers to the DBManager. The manager makes at least one corresponding SQL request for each method call, although this is not shown in the diagram. See the Miro board for a more complete version.


- transport: nodemailer.Transport
Methods
+ constructor(testMode: boolean): EmailManager
+ verify(): Promise<void>
+ sendInviteMail(
recipient: string, content: string
): Promise<string>
- firstLogin: boolean
- session: string
- verifyGoogleCredential(cred: string):
Promise<GoogleIdentityInfo>
Accesses DB Tables
DBManager
Attributes
- connection: postgres.SQL
Methods
+ getUser(email: string): Promise<DbUser | null>
+ getUserInfo(id: number: Promise<DbUSer | null>
+ newUser(name: string, email: string):
Promise<number>
+ getUserSession(cookie: string):
Promise<number | null>
+ newSession(uid: number): Promise<string>
+ removeSession(session_cookie: string):
Promise<void>
+ removeAllUserSessions(uid: number):
Promise<void>
+ getLocalResources(): Promise<LocalResource[]>
+ getRecentInvites(recipient: string):
Promise<Invite[]>
+ newInvite(recipient: string, creator: number):
Promise<void>
LoginPage(ViewController)
Props:
- clientId: string
State:
- privateVar: type
statePassedDownVar: type
(if state is passed down via props)
+ publicVar: type
(if setter is passed down to child
components or functions)
+ finishLogin(token: string) =>
Promise<LoginResponse>
+ onError: (error: string) => void
+ onSuccess: (token: string) => void
DashboardPage(ViewController)
Props:
- name: string
- hasSession: boolean
- curQuote: string
- getServerSideProps()
Local Resource(Component)
Props:
- name: string
- img_path: string
- description: string
- extended_description: string
State:
- show: boolean
- setShow(show)
- handleClose: setShow(false)
- handleShow: setShow(true)
Local Resource
Page(ViewController)
Props:
- hasSession: boolean
- getServerSideProps()
ViewOldQuotes(ViewController)
Props:
- hasSession: boolean