Spring 2023

Databases and Deliberation

Britany Orozco

Quentin Jensen

Databases and Deliberation

Britany Orozco and Quentin Jensen

Introduction

GeoEngineers, a Washington-based consulting firm with a focus on engineering and earth science, is in the process of transitioning between paper-based documentation to digital copies. Because of this, GeoEngineers wants to prevent future accumulation of paper documents by using webforms instead of continuing to collect hard copies. However, certain documentation, such as the GeoEngineers Vendor Information Form, requires security factors to be taken into account.

GeoEngineers wants an electronic version of the current system, where they can send an email link to contractors and have them fill out and submit the webform electronically. An online portal for employees to check on the status of the forms and find the contact information of the contractors is integrated with the email form. We, a team of computer science students at Western Washington University consisting of Madeline Carter, Ethan Crow, Evan Johnson, and myself, were contracted to make this software with the help of Andy Zeigert and Jon Stopchick from GeoEngineers.

Summary

Contractors send packets to gather information on their company, like who to contact to negotiate a contract. This is known as the vendor information form. To be filled out electronically, this form would need to be emailed to the company's contact through a link, filled out at said link, and then submitted. The form would then be saved into a database where employees at GeoEngineers can search to find companies to contract and send out form links through email if a company doesn't exist or needs to update their information.

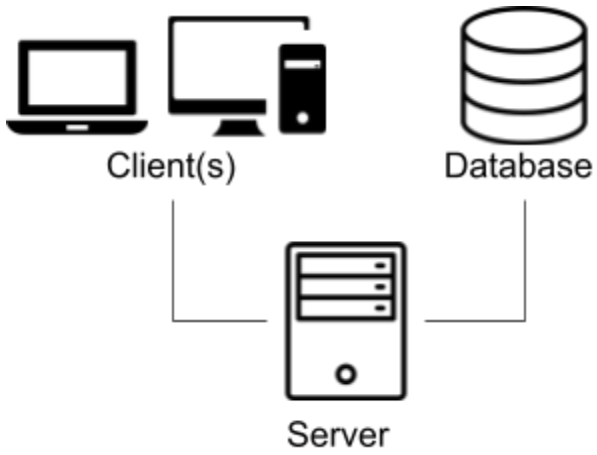
Background

Form-based connections between clients, servers, and databases can be seen in daily internet interactions. One example is online shopping. After filling up an online shopping cart with goods, the customer is then brought to the checkout page. This checkout page is an online form where the customer must input their shipping address and credit card information. After the form is filled out and submitted, the customer has then successfully purchased their items. The rest of the work necessary to process this purchase is then in the hands of the company selling the goods.

The resulting line of work is as follows: the shipping address and information about which and how many items were purchased gets extracted from the form by the client that is hosting the form. The client then sends that information to the company's servers, which saves it in a database. From here, the company can process the data and pull the information out of the

database when filling out the order. Our work follows a similar workflow where there needs to be an efficient connection between the client, server, and database.

Client-Server-Database Architecture



A diagram of a laptop and desktop computer with the label “clients” connecting to a server while the server connects to a database.

What is a Client?

A client is a program on the user’s side that communicates with the server to receive resources and perform services. The client program sends requests to the server, and the server responds back with the requested data or the result of a performed service. The most common examples of client programs are email clients, instant messaging applications, web browsers, etc.

A popular system used in web development is the client-server system. In this system, the server exists on the developer’s side, and it works with the client to provide a specific service or resource. An example of a client-server system is a Google web server providing web pages to a web browser client on a user’s computer when they go to Google.com.

What is a Web Server?

A web server is a program on the developer’s side that communicates with the user’s client. The web server receives requests from clients and sends back the requested content, which in the context of this project is a web page that includes HTML, CSS, and JavaScript files. Servers are responsible for processing requests from clients, routing them to the appropriate resources, and sending back the requested content. Web servers can also communicate with databases to provide clients with dynamic content.

What is a Database?

A database is the storage container that holds and organizes data. From the database, the data can be accessed, updated, and deleted. Databases are typically managed by a database management system (DBMS), which is software that provides tools and interfaces for creating, accessing, and manipulating data in the database. In this project, the DBMS used is PostgreSQL.

Databases can generally be classified as either relational databases or non-relational databases. Relational databases have tables with predefined relationships between them, while non-relational databases use document-based, graph-based, or key-value data models to allow for more flexibility when storing and organizing data.

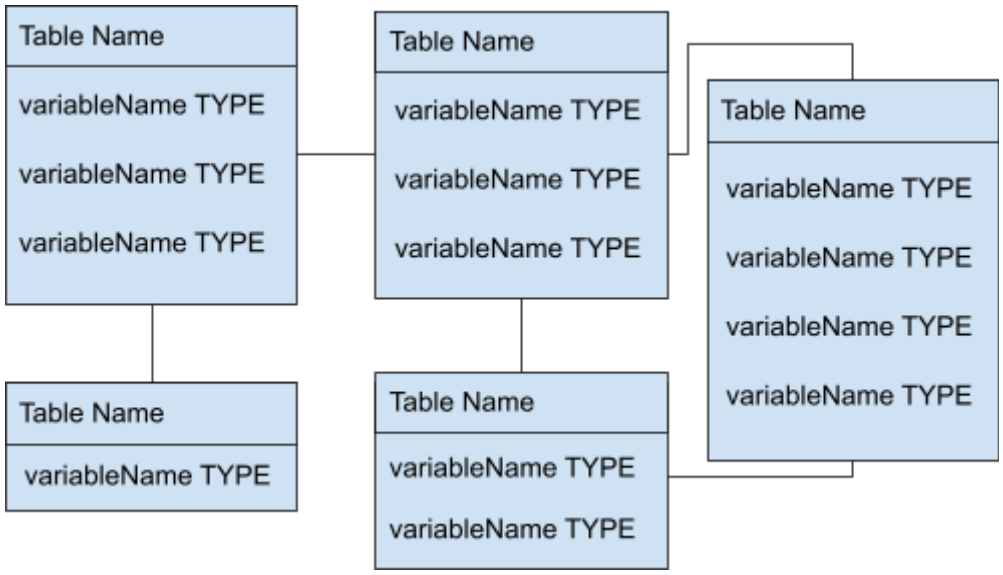
Adding a database to a client-server system creates the client-server-database system. This system is actively used in most web applications that collect data and store it for future use.

Database Design Fundamentals

Because this project was highly dependent on the usage of a database, we spent the beginning stages of our project on database design. This project utilizes a relational database because this project cares more about the data validity rather than flexibility. We started by creating a schema and then focused on constructing an Entity Relationship Diagram from the schema we made, also known colloquially as an ERD, to have a solid conceptual foundation on which to build our database from.

What is a Schema?

A schema is like a blueprint used to explain how to organize data in a database. It provides a framework for the components that make up the database. The components that make up a schema include: uniquely-named tables to store objects, columns to separate and store specific data attributes or properties, relationships between tables to describe how data between tables are related to each other, and constraints that set rules for how to store the data, what types of data can be stored, and where to store that data. Overall, a schema is a structural method of conceptualizing data storage for a more efficient and consistent way of organizing and accessing data.

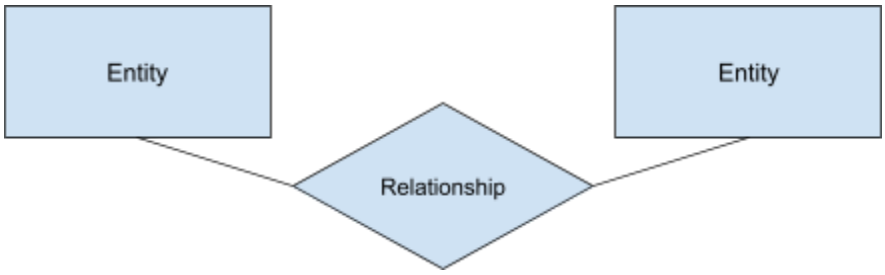


A diagram of a schema full of five tables, each with their own table name and variables.

What is an ERD?

An Entity Relationship Diagram (ERD) is an illustration used to show how data connects to each other in a database. The data is separated into tables called “entities” and connected

with “relationships.” Relationships share values from both entities to be able to connect them. An example would be a student taking a course, where the student and course are entities and the action of enrollment is the relationship between the two.



A diagram of two boxes labeled as “entity” connected by a diamond-shaped box labeled “relationship.”

What’s the Difference Between a Schema and an ERD?

While the two seem similar, a schema and an ERD share some key differences. Most notably, a schema is a representation or plan to build a database off of while an ERD is a visual tool to help explain the ways that the data connects to each other through relationships in the database. Both are used to help describe how to build a database, but a schema is more detailed than an ERD. Furthermore, a schema is the primary document for the implementation of a database while an ERD serves as a supplementary model to assist in connecting the data tables after creating the base of the database with the schema. For the purposes of this project, we created a schema to develop the foundation for the database and designed an ERD to better explain the connections between the data.

Methodology

Because this project was derived from a previous project, the development environment that we adopted consisted of JavaScript as the primary coding language and PostgreSQL for the database management with the help of Knex to create the database schema. Our frontend User Interface (UI) was created with the ReactJS library, and the backend runtime environment used to create the routers and server was NodeJS with the help of Express. Our Integrated Development Environment (IDE) of choice was Visual Studio Code with Amazon Web Services (AWS) as our server host and authenticator. For version control and to host our repository, we used Git via GitHub.

Designing the Database

We built off of the pre-existing database, so the tables that already existed were attachments, companies, osha_logs, and reviews. The schema that we built added business_classifications, naics_codes, organization_types, vendor_business_classifications, vendor_contact, vendor_identification, vendor_information, and vendor_request_status tables. From here, we connect the tables with foreign keys and primary keys. After the schema is created, we can build the database.

Building the Database

To build the database, we used migrations. Migrations are files that create new schemas or data components to update a database or move data to an updated database. The migrations that we created were JavaScript files that used Knex's schema property to create each table and populate the table's data types and variable names. Most of our table variables were text types to provide flexibility as to what input would be collected from the form.

Creating the Client

The client consists of components and screens. The components include the company table, request table, and vendor information form. The screens include the web pages that will be seen by users of the portal, such as the home page and request creation page. The vendor information form was created using React to design the input fields and collect input with form group components. The request page maps requests to a table to display the information for each request including the company name, contact name, email, phone, and submission status.

Creating the Server

An Application Programming Interface (API) endpoint for the vendor onboarding router is made. This API is the root for all endpoints made in the router. The endpoint is a unique route that interacts with the API and server. The vendor onboarding router uses Express and Knex to create endpoints for all functionality of the client web pages. Knex also works to communicate with the database through queries. The server consists of get, post, and delete endpoints.

Connecting the Client to the Server

The form submits with a button that sends a post request to the server when it is clicked. The form also fetches the business classifications and organization types from server api endpoints to populate each respective field for future scalability. The requests table receives the requests from a server api and uses it to populate the requests table.

Connecting the Server to the Database

The server uses api endpoints to extract data from the database and send it to the client. In the endpoints, Knex queries receive the requested data and provide it to the server for the server to then send the data to the client. From here, the client can then display updated data on the web pages while also using the server to send new data back to the database. The separation between the client and database ensures that unvalidated data doesn't enter the database, so the server can organize the data before sending it to the database.

Extra Features

Our project also required a feature that would send out an email through the creation of a new form request. This feature was created with Amazon SES to allow us to send an email to a company's contact with a server-side endpoint that is called by a client-side post request. We also were able to add resend and delete request features as bonus features since we

completed the bulk of the project before our expected end date. These features have their own buttons on the client side and endpoints on the server side to resend an email without creating a duplicate request and to delete a request itself from the database while not deleting a company or any other important data.

Discussion

While this paper is a general guideline for building a form-collection project that includes a database, it is important to mention that this project was built off of a pre-existing company portal and was created and managed through a business setting. All of the features created for this project cater to the needs of GeoEngineers rather than those present in a simple form collector, such as a survey. As a disclaimer, this paper does not mention any security concerns or protocols that should be taken and implemented to ensure that all collected data is secure. Furthermore, this project can be expanded to include more features that would make the portal easier to use or more useful for the end users.

Future Work

Some possible features and updates discussed during a shareholder meeting include the ability to forward forms, the ability to upload documents, an annual automatic re-sent form request to update old form submissions, an NAICS code lookup embedded into the form, and adding a required signature field before the form can be submitted. The signature feature would be useful when adding the form-forwarding ability because the vendor contact who received the form might not be the best person to fill out the form. In this case, the person who received the form can then forward the form to someone else and have them add their signature before submitting. This ensures that the contact listed for the company is not assumed to be the person who filled out the form, so the proper person can be contacted should there be any issues with the form submitted.

Conclusion

This paper describes the process taken to build a digital form-collection system in a business context. By starting with the motivation for the project, we can gather the requirements necessary to complete before we can consider the project complete. From there, we start discussing the client-server-database architecture principles that must be reviewed before we can design our database. Database design fundamentals are reviewed, such as what a schema or entity relationship diagram is, and those fundamentals are used to design the database. Next, we show how a development environment is chosen. With the environment set up, we then discuss how to code the client and server. The client displays the web pages while the server helps the client communicate with the database through endpoints and requests. With the client, server, and database connected and communicating, the implementation of features can then be considered.