Deploy Application to Cloud & Research

Kaya Nelson

Grand Canyon University: CST-323-O500

Nov 24, 2024

Test Application

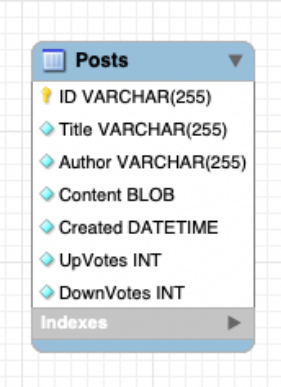
GitHub URL: https://github.com/Kdeshun/CST-323-activitys

The test application for this course activity is still under development, but it has made notable progress since the last milestone. It will function as a multi-page lite-blogging platform, more akin to Twitter than WordPress, and will be linked to a MySQL database. The application is being built using NodeJS and the React framework, with Bootstrap employed to maintain a cohesive design.

Users will have the option to "claim" a username while they are active on the app. Once they log out, that username will be released for others to use for posting or managing their content. The main feed includes a simple text editor for creating posts and a scrolling display of the most recent entries. Additionally, users can click on their username next to the Logout button to access a section where they can manage all their posts.

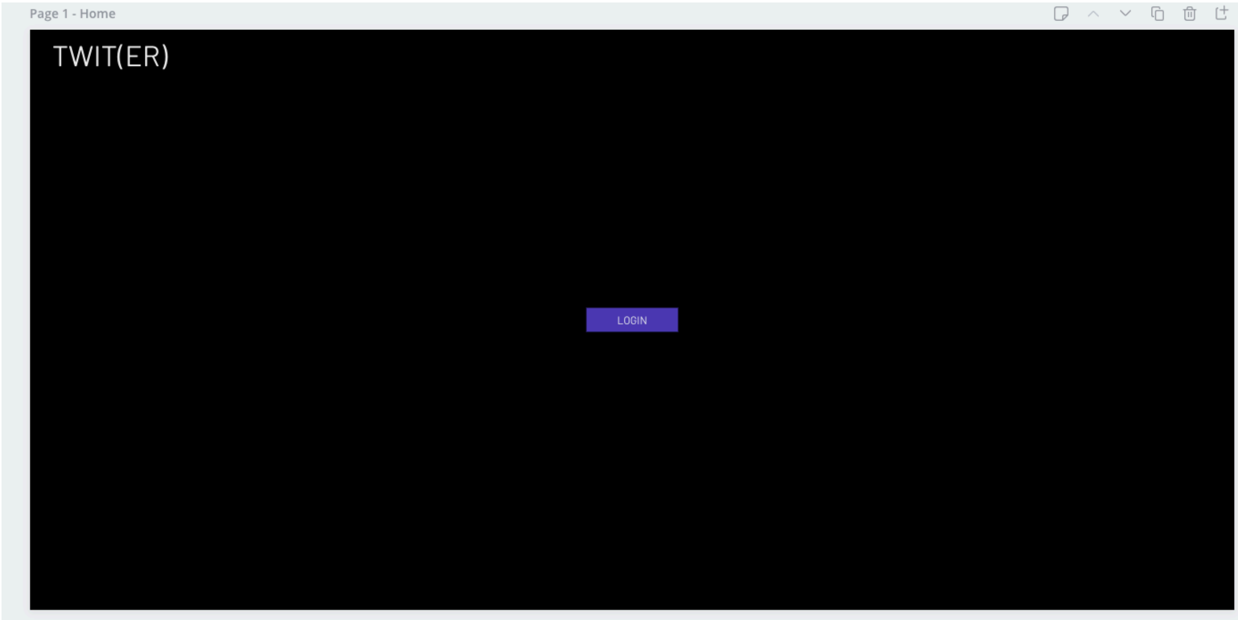
ER Diagram

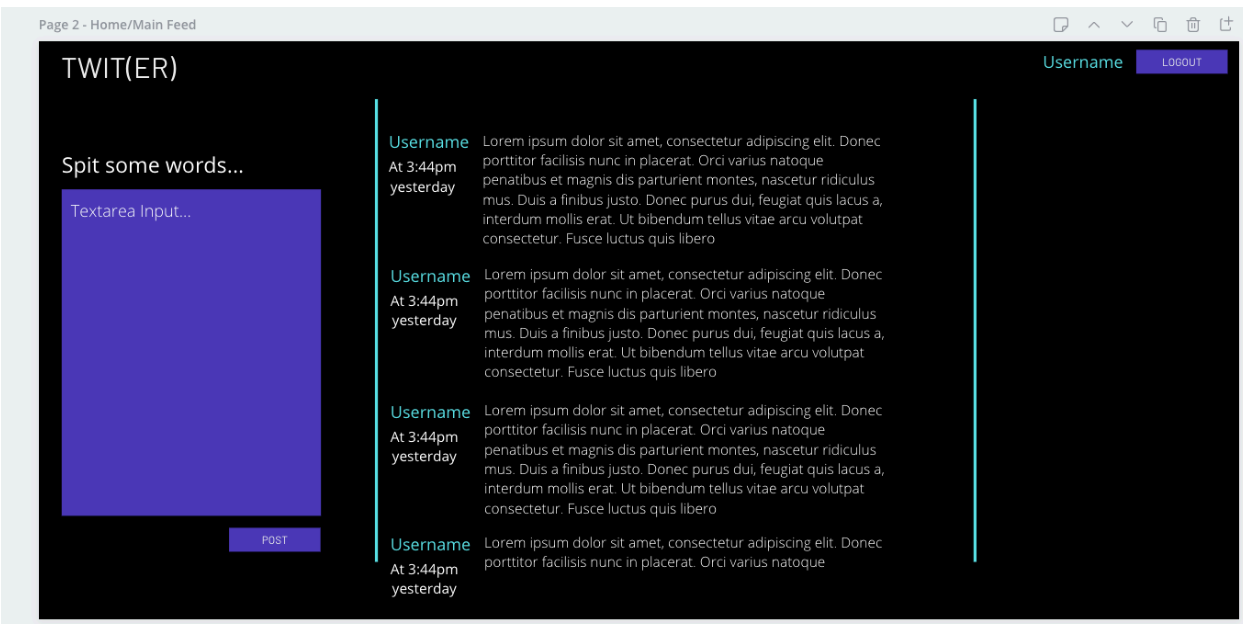
As of this milestone, the test application requires only one database table to function.



UI Designs

Some basic preliminary designs made in Canva.


Page 1 - Home
TWIT(ER)
LOGIN


Page 2 - Home/Main Feed
TWIT(ER)
Username
LOGOUT
Spit some words...
Textarea Input...
POST
Username
At 3:44pm yesterday
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec porttitor facilisis nunc in placerat. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Duis a finibus justo. Donec purus dui, feugiat quis lacus a, interdum mollis erat. Ut bibendum tellus vitae arcu volutpat consectetur. Fusce luctus quis libero
Username
At 3:44pm yesterday
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec porttitor facilisis nunc in placerat. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Duis a finibus justo. Donec purus dui, feugiat quis lacus a, interdum mollis erat. Ut bibendum tellus vitae arcu volutpat consectetur. Fusce luctus quis libero
Username
At 3:44pm yesterday
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec porttitor facilisis nunc in placerat. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Duis a finibus justo. Donec purus dui, feugiat quis lacus a, interdum mollis erat. Ut bibendum tellus vitae arcu volutpat consectetur. Fusce luctus quis libero
Username
At 3:44pm yesterday
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec porttitor facilisis nunc in placerat. Orci varius natoque

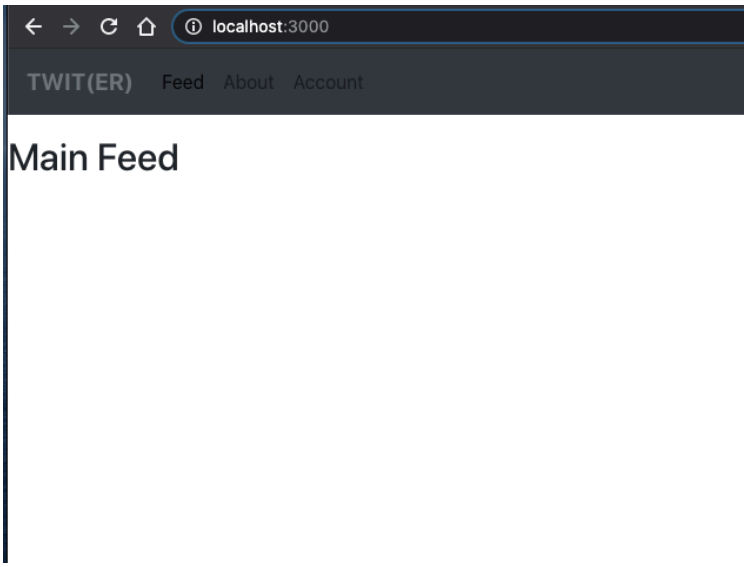
There will be an “About” page, including a feedback form that will create a comment in a different database table.

Service API Design

The API servicing this small application will have only a handful of routes for manipulating and querying the Post records from the connected database.

- /api/posts (GET)
- /api/post/:id (GET)
- /api/post/create (POST)
- /api/post/update (POST)

Screenshots



Cloud Computing Research

API Consumption Considerations

Here are some technical questions to keep in mind when leveraging a new API service through an external provider.

1. Does this API conform completely to the constraints imposed by the REST standard?
2. Does this API offer integration with my service's language/framework, or does it at least support a data format that I need for data parsing/manipulation (XML/JSON/etc.)?
3. Are my needs for consuming this API going to be met by the API's usage limits and restrictions? If not, will I need to perform caching myself or restrict my own users' activity limits?
4. Does the API offer authentication methods which allow users to give their explicit consent for actions taken on their behalf? (Think of a GitHub API which allows 3rd party applications to perform actions on user's repositories for them).
5. Does this API have adequate developer documentation and support articles and communities where a team could learn from other customers' use of the platform? Does

the documentation set include a developer sandbox for getting better acquainted with its features?

I believe that these questions can help a team sort out what exactly they need to be searching for in an ideal 3rd-party API service. A good, well-documented API is a rarity worth seeking out.

Cloud Platform Comparison

AWS and Google Cloud will be compared on ten different points.

1. Regions/Availability: Both AWS and GCP offer various regions where users can host their solutions. AWS has over 40 Availability zones spread throughout 16 regions globally, and GCP has around 24 locations from which users can spin up new services.
2. Languages/Frameworks: Both companies offer PaaS solutions which cater to a smattering of more popular and common programming tools/langauges. Lambda and GCP Functions support Node.js, Python, Go, and Java. However, AWS supports a handleful of other languages too, including PowerShell, Ruby, and C#. The IaaS-level services can support whatever tools and languages developers can configure, of course.
3. Database Offerings: Both providers offer hosting or nominal support for most popular database systems (Oracle SQL, Redis, MySQL, PostgreSQL). AWS and Google offer their own proprietary hosting solutions for long-term storage or archiving: Google BigQuery & AWS RedShift. AWS also offers RDS, which can host any of the main database engines along with their own proprietary relational database, Aurora.
4. Web Client/GUI: The AWS web client is certainly more mature than Google's cloud service. While GCP has a more modern-appearing, Material design interface, AWS seems to pack more configuration menus and options into their site overall.

5. Developer integration/SDKs/CLIs: Both companies sport a large number of SDKs to work with each of their more mature products. Both have official developer CLIs, which have the ability to manipulate most cloud resources just from the console. From personal experience, AWS appears to have a higher volume of developer documentation and tutorials for dealing with their code interfaces. GCP has a very streamlined documentation set, but it can leave developers searching elsewhere for more complex examples or expanations.
6. Pricing: This all depends on the needs of the individual customer, but overall these companies both offer comparable services for around the same cost since they are direct competitors. They both offer a free trial tier that gives roughly the same value (~$200 service credit) to new users.
7. Security: GCP's security interface is less straight-forward than the one AWS uses for managing its IAM permissions, in my opinion. AWS, as the more mature service, boasts a vast scope of granular security options. Its complexity can be a high barrier for new developers or teams to get used to. However, this makes AWS the go-to provider when security is a primary concern. However, a user's security is only as good as they've configured it, as we all saw with the recent Capital One hack.
8. Scalability: Both companies offer instantly-scaling NoSQL databases and serverless computing. There are differences in setting configurations and limits for containerized services and their PaaS solutions, but there aren't any harsh disparities between what both can offer.
9. IaaS (Private Cloud): Both providers offer robust private cloud offerings. GCP has less, for sure, given that it's Compute Engine is the only product their offer that's purely IaaS.

GCP largely got its start as a PaaS provider, whereas AWS began as an in-house IaaS provider for Amazon itself.

10. Support: AWS has the more mature support team and article set, but GCP also offers comparable tiers of near-instant support help, all the way up to getting aid with architecting services on their platform. AWS is likely slightly cheaper at scale, but both of them offer roughly the same features to each similar-tier support plan.

Cloud Deployment Limitations

Following is a list of some limitations businesses should keep in consideration when pursuing cloud deployment options for their services and products. Some of these are constrained only to the skill of their staff, while others can be highly circumstantial depending on their industry tools and internal environment.

1. In-house experience with cloud technologies is a huge determining factor in the feasibility of moving an application from on-premise or between cloud providers. In the case of AWS, a lot of enterprise support is available for a cost, but the high complexity of the platform might warrant hiring outside consultants or contractors to complete a successful migration.
2. Integration with on-premise systems. At my company, some scripts and application services require secure access to on-premise servers and systems behind boundaries set by strict domain controllers. Moving these to the cloud required some specific settings and workarounds in order to get the same scripts functional again. Sometimes that hassle is just too much work during a service's expected lifespan.
3. Internal adversity to risk/new costs. Every business's people have a different common comfort level for risk or exploration. Companies with a high tolerance for these will be

more willing to explore these cloud options fully, whereas the more adverse companies will drag their feet and potentially commit half-way, resulting in different sets of issues.

4. The kind of platforms being deployed. If a company is deploying a customer-facing platform or service, they need to consider the reliability for the sake of their customer base. They'll require training for their support team and informed analysis on how the move to the cloud could impact customer experience.
5. The technical needs of the system. If the system deployed is an emulator requiring heavy processing power and low latency, then it may not be feasible to offload it to the cloud. A strange example of this is Google Stadia, which attempted to offload the need for users to own game consoles in order to play the latest AAA titles. Unfortunately, the user experience was trashy, as many users didn't have Wifi fast enough to have a playable experience at all. In these cases, it's ultimately better to use an "on-premise" system.

References

Kavis, M. (2014). *Architecting the cloud: Design decisions for cloud computing service models (SaaS, PaaS, and IaaS)*. Hoboken, NJ: Wiley.