1. (5 points) Explain the backend for frontend (BFF) architectural pattern. Why might you want to use it?

   The backend for frontend architectural pattern is a methodology where you create a separate backend for each frontend; this is opposed to a single back end servicing all front-ends (web, apps, etc) [1]. This is useful because it allows you to create a backend that is specifically tailored to the needs of a particular frontend, typically increasing efficiency and reducing complexity. There is also the added benefit of removing competing requirements/needs between the different frontend.
   Overall, this pattern is useful when you have multiple frontend that have different requirements and needs. Each front end is allowed to have its own backend and such can be tailored and developed in isolation from the other frontend, eliminating the competing requirements.

2. (5 points) Explain microservices, why they're useful, and when an organization might consider introducing them?

   Microservices is a software architectural style that uses small independent services which communicate over APIs. Each service is owned (developed) by a small team and is more easily scaled and quicker to develop [2]. This is useful because it allows for the development of an application to be broken down into smaller, more manageable pieces. This allows for the development of each service to be done in isolation, and can be developed and deployed independently of the other services. This can lead to increased agility and speed of development, as well as the ability to scale and maintain the application more easily.

3. (5 points) What is gRPC and when might you use it?

   gRPC is an open-source remote procedure call (RPC) system developed by Google, designed for efficient communication between distributed systems. It utilizes Protocol Buffers (protobuf) as its Interface Definition Language (IDL) and HTTP/2 as its transport protocol, offering features like bidirectional streaming and code generation for various programming languages. gRPC provides support for authentication, authorization, and provides a modern and efficient framework for building distributed applications [3] [4].

4. (10 points) Compare and contrast GraphQL and REST and why you might design an API to use one versus the other.

GraphQL and REST are both popular API design paradigms, with REST being the current "standard" for web APIs. REST is a stateless client-server architecture that uses HTTP methods to perform CRUD operations on resources. It is simple, easy to understand, and is widely supported by various tools and libraries. However, REST APIs can suffer from over-fetching and under-fetching of data, as the client has no control over the shape of the data it receives. This can lead to performance issues and increased network traffic.
GraphQL is a query language for APIs that allows clients to request only the data they need. It provides a single endpoint for all data operations and allows clients to specify the shape of the data they require. This can lead to reduced network traffic and improved performance, as well as a more flexible and efficient API. However, GraphQL can be more complex to implement and may require additional tooling and infrastructure.
When designing an API, you might choose to use REST if you need a simple, easy-to-understand API that is widely supported and can be quickly implemented. On the other hand, you might choose to use GraphQL if you need a more flexible and efficient API that allows clients to request only the data they need, reducing network traffic and improving performance.

5. (6 points) Compare and contrast Express with another web framework of your choosing.

Express.js vs Flask: Express.js and Flask are both popular web frameworks, differing in their language, routing, and scalability approaches. Express.js, based on JavaScript and Node.js, utilizes middleware-based routing and offers multiple asynchronous handling methods. It excels in performance but has a moderate learning curve. In contrast, Flask, built on Python, uses flexible routing with decorators and supports asynchronous request handling. It provides session management and authentication by default but may need extra configuration for scalability. Flask is beginner-friendly and performs well for smaller projects. Ultimately, the choice depends on language preference, project needs, and developer familiarity.

6. (8 points) Choose two common security concerns with web applications and explain them in some detail along with their common mitigations (i.e. injection attacks, cross-site scripting, etc.)

(Personal) Data Exposure: Personal data exposure (and really any data exposure) is a common security concern with web applications, as it can lead to identity theft, fraud, and other malicious activities. In the context of web applications, this can occur when sensitive information such as usernames, passwords, credit card numbers, and personally identifying information are stored or transmitted insecurely. To mitigate this risk, web applications should use encryption to protect sensitive data, implement secure authentication and authorization mechanisms, and follow best practices for secure data storage and transmission. This includes using HTTPS (There is literally no reason to use HTTP!), hashing and salting passwords, and implementing access controls to limit the exposure of sensitive data.
Data exposure also exist beyond database breaches and poor data storage/transmission practices. Data can also be exposed through simply web-scraping/crawling. Data like, emails, phone numbers, addresses, etc are commonly collected from webpages via this method and used in phishing and spam attacks. This can be mitigated by implementing rate limiting, CAPTCHAs, and other bot detection mechanisms.
Outdated/vulnerable packages and libraries: Web applications often rely on third-party packages and libraries to provide functionality and features. However, these packages and libraries can contain vulnerabilities that can be exploited by attackers. Particularly, the use of outdated packages and libraries can be a significant security risk, as they may contain known vulnerabilities that have been patched in newer versions. To mitigate this risk, web applications should regularly update their dependencies to the latest versions. (There is an argument to staying a couple of versions behind to avoid zero day vulnerabilities). Additionally, web applications should use tools and services to monitor for vulnerabilities in their dependencies and take action to remediate any issues that are identified.
Studies have found that "95 % of the analyzed websites use at least one product for which a vulnerability existed" [5]. As recently as "December 5, 2023, The CISA (Cybersecurity and Infrastructure Security Agency) announced that a cyber-attack had occurred against the United States government. It is stated that con artists hacked it through vulnerabilities of outdated software." [6]

7. (5 points) What is progressive enhancement and how might metaframeworks like Next.js and Remix help us achieve this?

Progressive enhancement is a strategy where you provide the most basic features to all users, while only providing higher level, more advanced features, to users with more modern/capable browsers and internet speed [7]. This is done to ensure that all users can access the content and functionality of a web application, regardless of their device or network conditions. But also to provide a better experience to those which have the capability to use it.
Metaframeworks like Next.js and Remix help us achieve this by providing tools and features that make it easier to build web applications that are progressively enhanced. For example, Next.js and Remix both provide support for server-side rendering, which can improve the performance and accessibility of web applications. They also provide tools for code splitting, which can reduce the initial load time of web applications and improve the user experience. Additionally, Next.js and Remix provide support for modern web technologies like React and TypeScript, which can be used to build web applications that are more accessible and performant.

8. (6 points) Compare and contrast relational databases (i.e. Postgres) and document databases (MongoDB). You may choose to consider how you access data, how data is stored, and scaling characteristics of these types of databases.

Relational databases like Postgres are based on the relational model and use tables to store data. They use SQL to access and manipulate data, and support complex queries and transactions. Relational databases are well-suited for applications that require complex data relationships and need to support ACID transactions. However, they can be less flexible and scalable than document databases, and may require more effort to manage and maintain.
Document databases like MongoDB are based on the document model and use collections to store data. They use JSON-like documents to represent data, and support flexible schemas and nested data structures. Document databases are well-suited for applications that require flexible data models and need to support horizontal scaling. However, they may be less efficient for complex queries and transactions, and may require more effort to manage and maintain.
In terms of scaling, relational databases typically use vertical scaling, which involves adding more resources to a single server. This can be expensive and may not be sustainable in the long term. Document databases, on the other hand, typically use horizontal scaling, which involves adding more servers to distribute the load. This can be more cost-effective and sustainable, especially for applications that need to support large volumes of data and traffic.

References

[1] Microsoft, "Backends for frontends," n.d., accessed: March 13, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/architecture/patterns/backends-for-frontends

[2] Amazon Web Services, "Microservices," n.d., accessed: March 13, 2024. [Online]. Available: https://aws.amazon.com/microservices/

[3] "gRPC," gRPC, n.d., accessed: March 13, 2024. [Online]. Available: https://grpc.io

[4] "gRPC introduction," gRPC, n.d., accessed: March 13, 2024. [Online]. Available: https://grpc.io/docs/what-is-grpc/introduction/

[5] N. Demir, T. Urban, K. Wittek, and N. Pohlmann, "Our (in)secure web: Understanding update behavior of websites and its impact on security," 02 2021.

[6] Infiniwiz. (2023) Government breach exposes risks of outdated software. Accessed: March 13, 2024. [Online]. Available: https://www.infiniwiz.com/government-breach-exposes-risks-of-outdated-software/

[7] R. Team. (Year the page was last updated, e.g., 2024) Remix documentation: Progressive enhancement. Accessed on Date, e.g., March 17, 2024. [Online]. Available: https://remix.run/docs/en/main/discussion/progressive-enhancement