Assignment 3: Design a 12 Factor App for an ecommerce website using Heroku as the tech stack

Description: You are tasked with designing a scalable and reliable ecommerce website using the 12 Factor App principles in the Heroku tech stack. The website should be able to handle a high volume of traffic, be easily deployable, and allow for easy collaboration among developers.

Tasks:

1. Design the application using the 12 Factor App principles, including configuration, dependencies, logs, and processes.
2. Choose a suitable datastore for the application, such as Heroku Postgres, and design the database schema.
3. Implement a continuous integration and deployment (CI/CD) pipeline using Heroku's Git-based workflow.
4. Use containerization with Docker to create a portable and consistent environment for the application.
5. Implement monitoring and alerting using Heroku's add-ons, such as New Relic and Logentries.
6. Implement caching to improve performance, using Heroku's Redis add-on.
7. Implement authentication and authorization using Heroku's add-ons, such as Auth0 and Okta.
8. Implement security measures, such as encryption of sensitive data and protection against common web attacks.
9. Use Heroku's add-ons, such as Cloudinary and AWS S3, to handle image and file uploads.
10. Implement a search functionality using Heroku's add-ons, such as Algolia and Elasticsearch.
11. Implement email and messaging functionality using Heroku's add-ons, such as SendGrid and Twilio.
12. Use Heroku's add-ons, such as Heroku Connect and Salesforce, to integrate with external systems, such as CRM and ERP systems.

## Deliverables:

1. A detailed architecture diagram of the application, including the various components and their interactions.
2. A detailed database schema and data access layer design.
3. A working application that meets the requirements specified in the tasks.
4. Documentation on how to deploy, maintain, and scale the application.
5. A presentation on the design choices made and the rationale behind them.
6. A demonstration of the application's scalability and reliability under high traffic.
7. A discussion on the trade-offs made between different design choices and their impact on the application's performance, scalability, and maintainability.
8. A reflection on the lessons learned and the improvements that can be made in future iterations of the application.

## Solution:

1. Codebase: Use a version control system (VCS) like Git to manage the codebase for the ecommerce website. Use Heroku Git to deploy the codebase to the Heroku platform.
2. Dependencies: Use a dependency management tool like Bundler for Ruby or NPM for Node.js to declare and manage the dependencies required by the application.
3. Config: Store configuration information in environment variables using Heroku's config vars feature. Avoid storing configuration information in code or files.
4. Backing services: Use Heroku add-ons to provide backing services such as databases, messaging queues, and caching services. Use the 12 factor principle of treating backing services as attached resources that can be swapped out easily.
5. Build, release, run: Use a build system like Heroku's buildpacks to build the application and package it for release. Use the Heroku CLI or Heroku Dashboard to manage releases and deployments.
6. Processes: Use Heroku's process model to run the application as one or more lightweight processes. Use the Procfile to declare the processes and their commands.
7. Port binding: Use the PORT environment variable to specify the port that the application should listen on. Use the Heroku platform's routing system to route incoming requests to the appropriate process.
8. Concurrency: Use Heroku's dyno scaling to adjust the number of processes running based on the application's load. Use a background job processing system like Heroku's worker dynos to handle asynchronous processing.
9. Disposability: Design the application to be disposable by minimizing startup time and gracefully handling shutdown signals. Use Heroku's dyno manager to handle process crashes and restarts.
10. Dev/prod parity: Use Heroku's pipelines feature to ensure that the development, staging, and production environments are as similar as possible.
11. Logs: Use Heroku's logging system to store and manage application logs. Use a log management system like Logentries or Papertrail to analyze and search logs.
12. Admin processes: Use Heroku's run command to run ad-hoc administrative tasks or one-off scripts. Use Heroku's API or Heroku CLI to automate administrative tasks.