

Phillip Ball  
Prof. Jazzar  
CST-323  
11/13/24

# Activity 3

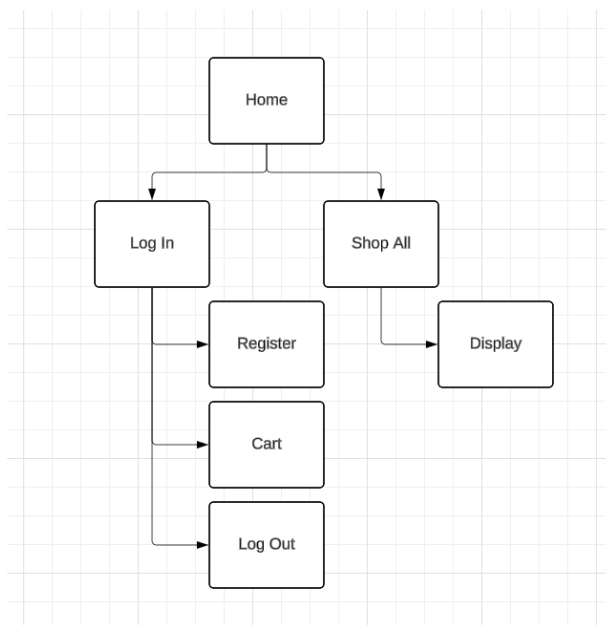
Screencast: <https://www.youtube.com/watch?v=BjkHLcSUq9g>

## Heroku Step-by-Step

- 1: Made Heroku account
- 2: Downloaded Heroku CLI on VS Code
- 3: Connected the API Repository to Heroku
- 4: Deployed the API to Heroku using 'git push heroku master'
- 5: Connected the procfile to the web dyno
- 6: Server is now live

API Git Repository: <https://github.com/PhillipBall1/LeafLoungeAPI>

Application Design:



## Database Design:

Plants	
_id	ObjectId
plant_name	string
family_name	string
scientific_name	string
plant_description	string
image	string
price	string
difficulty	string
featured	boolean
indoor	boolean
edible	boolean

Users	
_id	ObjectId
username	string
password	string
admin	boolean
cart	array

## Cloud Computing Research

### Eight "Worst Practices" in Cloud Migration

#### Ignoring Cloud-Specific Architectures

Migrating applications as-is without leveraging cloud-native architectures can lead to performance, scalability, and reliability issues. *Prevention:* To avoid this, invest in re-architecting the application to use cloud-native features like microservices, serverless functions, or containerization, which optimize resource usage and scalability. Involve architects and developers early to align application design with cloud best practices, ensuring the migration brings tangible benefits in cost, agility, and performance.

#### Failing to Conduct a Thorough Cost Analysis

Migrating without a comprehensive understanding of cloud costs can lead to unexpected expenses and budget overruns. *Prevention:* Implement a detailed cost analysis to project expenses, considering factors like data transfer, storage, and compute instances. Use cloud providers' pricing calculators and budgeting tools to estimate costs accurately, and consider cost-optimization practices such as rightsizing resources, using reserved instances, and setting budget alerts.

## **Neglecting Security and Compliance Requirements**

Assuming that all security concerns are covered by the cloud provider can result in data breaches or compliance violations. *Prevention:* Identify and document your application's specific security and compliance needs, including data protection, encryption, and access control requirements. Use the cloud provider's security tools, such as Azure Security Center or AWS Security Hub, to enforce policies and continuously monitor for vulnerabilities. Regularly review and update security practices to align with industry standards and regulatory requirements.

## **Inadequate Planning for Data Migration**

Poor planning for data transfer can lead to data loss, extended downtimes, and service interruptions. *Prevention:* Develop a data migration strategy that includes data integrity checks, bandwidth considerations, and downtime minimization tactics. Use robust data transfer tools, like AWS Data Migration Service or Azure Data Box, to streamline the migration process, and test data migration on a subset before full-scale transfer to identify any issues early.

## **Not Testing the Cloud Environment**

Skipping testing can lead to unanticipated errors and performance bottlenecks in the new environment. *Prevention:* Conduct comprehensive testing, including performance, load, and integration tests, in the cloud environment to detect potential issues. Build a testing framework that mirrors real-world usage scenarios to validate that the application performs as expected and meets SLAs. Regular testing ensures smoother transition and helps prevent unexpected issues post-migration.

## **Overlooking Disaster Recovery Planning**

Not having a disaster recovery (DR) plan can result in prolonged outages and data loss during critical incidents. *Prevention:* Create a DR plan that includes data backups, failover configurations, and recovery procedures. Use cloud-native DR solutions such as Azure Site Recovery or AWS Backup, and perform regular DR drills to ensure preparedness. Document the DR processes, assign responsibilities, and ensure that the team is trained to execute the plan effectively.

## **Insufficient Staff Training on Cloud Technologies**

Failing to train staff on cloud technologies can lead to configuration errors and underutilization of cloud resources. *Prevention:* Provide structured training programs for your team to gain skills in cloud services, monitoring, and security best practices. Encourage certification in relevant cloud services and conduct workshops or lab sessions to build hands-on experience. Ensuring that staff are skilled and confident in using the cloud environment mitigates risks associated with knowledge gaps.

### **Neglecting Post-Migration Monitoring and Optimization**

Once the migration is complete, failing to monitor and optimize can lead to degraded performance and increased costs over time. *Prevention:* Set up automated monitoring tools to track application performance and resource utilization. Use cloud-native tools like Azure Monitor or AWS CloudWatch for real-time insights, and regularly analyze the data to make informed adjustments. Periodically review and adjust resources to keep costs down and ensure optimal performance, addressing potential issues before they impact users.

## **Comparison of Microsoft Azure and Heroku Cloud Features**

### **Platform Focus**

1. *Azure:* A versatile cloud platform for IaaS, PaaS, and SaaS offerings, supporting complex enterprise applications.
2. *Heroku:* Primarily PaaS, tailored for developers to deploy applications quickly without managing infrastructure.
3. *Comparison:* Azure supports a broader range of cloud models, whereas Heroku's simplicity suits smaller, more agile projects without extensive infrastructure demands.

### **Supported Programming Languages**

*Azure:* Supports .NET, Java, Python, JavaScript, Go, Ruby, PHP, and more.

*Heroku:* Supports most popular languages like Java, Node.js, Ruby, PHP, Python, Scala, and Go through buildpacks.

*Comparison:* Both offer wide language support, though Azure's extensive range may cater better to enterprises with legacy applications.

## **Deployment Models**

*Azure:* Offers flexibility with on-premises, hybrid, and multi-cloud deployment models.

*Heroku:* Primarily cloud-only, lacking hybrid deployment options.

*Comparison:* Azure provides more deployment flexibility for companies with specific regulatory or operational requirements, whereas Heroku's cloud-only focus simplifies deployment.

## **Container Support**

*Azure:* Offers extensive container support via Azure Kubernetes Service (AKS) and Docker integration.

*Heroku:* Has limited container support, mainly through Heroku's own "Dynos."

*Comparison:* Azure is ideal for containerized applications, while Heroku is better suited for simpler deployments with minimal container orchestration.

## **Cost Structure**

*Azure:* Pay-as-you-go, reserved instances, and spot pricing for cost optimization.

*Heroku:* Predictable monthly pricing tiers based on resource allocation.

*Comparison:* Azure offers more complex, flexible pricing options suited to varied usage, while Heroku's simpler pricing tiers are more predictable for small to medium-sized applications.

## **Data Storage and Management**

*Azure:* Provides multiple storage options (Blob, SQL Database, Cosmos DB).

*Heroku:* Limited to PostgreSQL and Redis, though third-party add-ons offer extended options.

*Comparison:* Azure offers a more comprehensive suite of managed storage solutions, while Heroku focuses on ease of use with fewer options.

## **Scaling and Load Balancing**

*Azure:* Auto-scaling and load balancing across VM instances, databases, and more.

*Heroku:* Offers horizontal scaling and load balancing through Dynos.

*Comparison:* Azure's scaling capabilities are more granular and suited to complex applications, while Heroku's simpler Dyno-based scaling is easy to manage but limited in complexity.

## **Marketplace and Add-Ons**

*Azure:* Extensive marketplace with thousands of services and third-party integrations.

*Heroku:* Add-ons marketplace provides useful tools but is smaller than Azure's.

*Comparison:* Azure's marketplace offers more extensive integration options for enterprise-grade applications, while Heroku's add-ons focus on streamlined, developer-friendly services.

## **Security and Compliance**

*Azure:* Advanced security features (Azure Security Center) and comprehensive compliance certifications for industries like healthcare, finance, and government.

*Heroku:* Basic security with compliance standards, but fewer industry-specific certifications.

*Comparison:* Azure provides a more robust security and compliance ecosystem, especially important for regulated industries, while Heroku's security focuses more on developer ease-of-use.

## **Developer Experience**

*Azure:* Offers developer tools like Visual Studio, GitHub integration, and DevOps solutions for complex CI/CD pipelines.

*Heroku:* Known for its simplicity, using Git-based workflows for deployment with quick set-up.

*Comparison:* Heroku's developer experience is straightforward and beginner-friendly, while Azure's tooling is designed for more complex DevOps practices, making it more suitable for large, collaborative projects with advanced requirements.

