The backend

Saving and retrieving data in an app

A UNC Makeathon workshop

Tuesday, February 12th, 2019

Jeff Terrell Ph.D., computer science professor of the practice

The problem

- All apps need data
- Where does the data live?
  - On-device
  - In a centralized location (e.g. “in the cloud”)
- The centralized location is the authoritative “source of truth”
- This is called “the backend” or “server”
- (”The frontend” or “client” is the app code running on the user’s device)

Backend components

- **The database** stores the data durably
  - “Durably” means “surviving a reboot”, i.e. using disks
- **The “API server”** interfaces between clients and the database
  - Why? Separation of concerns.
  - Handling HTTP requests
  - Doing authentication and authorization checks
  - Limit the types of interactions (untrusted) clients can have with data
  - Easier to develop as a separate program than as part of database
- Backend programmers *use* a database but *create* an API server

The API: application programming interface

- Like a “user interface” (the screens that users interact with), but for programs
- The frontend is a program that uses the API
- The API defines ways that the frontend code can save and retrieve data
- These ways are called “endpoints”; an API is a set of endpoints
- You must define an API for your app and your data
- Modern APIs usually use HTTP

HTTP: the hypertext transfer protocol

- Browsers use HTTP almost exclusively
- Two types of HTTP messages: requests and responses
- Requests have a method, a URL, and maybe parameters or a body
- Responses have a status code (success? error?) and usually a body
- All messages have headers with extra information, e.g. cookies and content types
- Request methods might be GET, POST, PUT, DELETE, etc.
- You can inspect HTTP messages in your browser

App development process

- Design screens that users will see
- Define an API
- In parallel:
  - Develop the backend
  - Develop the frontend
- Deploy
- Profit

Outline

1. ~~Introduce backend concepts~~
2. Decide what to build
3. Define an API
4. Write backend code
5. Write frontend code
6. Use a database
7. Deploy to Heroku
8. Lab: building a simple API

What we'll build

- A rudimentary social networking backend, inspired by Twitter
- Supported actions:
  - Create tweets
  - See tweets for a user
  - Like a tweet
  - See likes for a tweet

Outline

1. ~~Introduce backend concepts~~
2. ~~Decide what to build~~
3. Define an API
4. Write backend code
5. Write frontend code
6. Use a database
7. Deploy to Heroku
8. Lab: building a simple API

Defining an API • creating a tweet

- Our job: define expectations for HTTP requests and HTTP responses
- The request:
  - should have a method of POST
  - should have a URL path of `/tweet`
  - should have a content type of JSON
  - should have a body with 2 properties: `handle` and `body`, both strings
- The response:
  - If the expectations aren't met:
    - should have a code of 400 ("bad request") and a body that explains why
  - If the expectations are met:
    - should have a code of 201 ("created") and a JSON body of the tweet including an ID and a timestamp

Defining an API • seeing tweets for a user

- The request:
  - should have a method of GET
  - should have a URL path of `/t/<handle>`
- If there are no tweets for that user, the response:
  - should have a code of 404 (“not found”) and an empty body
- Otherwise, the response:
  - should have a code of 200 (“OK”)
  - should have a JSON body of tweets by that user
  - should include the IDs in the tweets
  - should sort the tweets most-recent first

Defining an API • liking a tweet

- The request:
  - should have a method of PUT
  - should have a URL path of `/like`
  - should have a JSON body with 2 string properties: `handle` and `tweetId`
- The response:
  - should have a code of 400 (“bad request”) if the given request was invalid
  - should have a code of 404 (“not found”) if the specified tweet was not found
  - otherwise, should have a code of 201 (“created”)
  - should have a JSON body with the like details echoed back

Defining an API • seeing tweet details

- The request:
  - should have a method of GET
  - should have a URL path of `/tweet/<tweetId>`
- The response:
  - should have a code of 404 (“not found”) if the specified tweet was not found; otherwise:
  - should have a code of 200 (“OK”)
  - should have a JSON body with two properties: `tweet` and `likes`

API Summary

Method	Path	Params	Status code(s)
POST	/tweet	• handle (string) • body (string)	400 (“bad request”) 201 (“created”)
GET	/t/<handle>	-	404 (“not found”) 200 (“OK”)
PUT	/like	• handle (string) • tweetId (identifier)	400 (“bad request”) 404 (“not found”) 201 (“created”)
GET	/tweet/<tweetId>	-	404 (“not found”) 200 (“OK”)

Outline

1. ~~Introduce backend concepts~~
2. ~~Decide what to build~~
3. ~~Define an API~~
4. Write backend code (see code repository on GitLab)
5. Write frontend code
6. Deploy to Heroku
7. Use a database
8. Lab: building a simple API

Outline

1. ~~Introduce backend concepts~~
2. ~~Decide what to build~~
3. ~~Define an API~~
4. ~~Write backend code~~
5. Write frontend code (see source code file on GitLab)
6. Deploy to Heroku
7. Use a database
8. Lab: building a simple API

Outline

1. ~~Introduce backend concepts~~
2. ~~Decide what to build~~
3. ~~Define an API~~
4. ~~Write backend code~~
5. ~~Write frontend code~~
6. Deploy to Heroku
7. Use a database
8. Lab: building a simple API

Deploying to Heroku

- Your backend needs to be running somewhere
- Heroku is an easy-to-use backend hosting service
- Steps
  - Create an app in Heroku
  - Add a Procfile telling Heroku how to run your backend
  - Tell git about Heroku
  - To deploy: `git push heroku master`

Outline

1. ~~Introduce backend concepts~~
2. ~~Decide what to build~~
3. ~~Define an API~~
4. ~~Write backend code~~
5. ~~Write frontend code~~
6. ~~Deploy to Heroku~~
7. Use a database
8. Lab: building a simple API

Using a database

- Remember: a database stores data durably (surviving restarts)
- We'll use MongoDB for this app because it's pretty simple
- Other choices: PostgreSQL, MySQL, Datomic
- Heroku has an "addon" service for MongoDB with a free tier
- One new concept: asynchronous code and async/await

Outline

1. ~~Introduce backend concepts~~
2. ~~Decide what to build~~
3. ~~Define an API~~
4. ~~Write backend code~~
5. ~~Write frontend code~~
6. ~~Deploy to Heroku~~
7. ~~Use a database~~
8. Lab: building a simple API

Lab: building a simple API

- Your turn! Apply this knowledge.
- Goal: build a backend that reports how many times its only endpoint has been accessed (since the last restart)
- 1. Define the API
- 2. Create the API server
- 3. Prove that it works (with “Postman” app or curl/http on the CLI)
- Pair programming recommended
- Use my code: https://gitlab.com/unc-app-lab/backend-tutorial
- Bonus goal: add an endpoint to reset the counter
- Also, fill out survey ($100 raffle!): http://www.surveymonkey.com/r/uncmakeathon