Beanstalk
Milestone 1

Terry Yang, Annie Lin,
Hiroka Tamura, Kaelan Mikowicz

Beanstalk - Share your adventure

- Bring people together around trendy spots and hidden gems!

Outline

Requirements Analysis

ER Model

Technology Stack

Live Demo

Requirements Analysis

Functionality and Features

User Authorization

User Profile Management

Post Creation

Activity Feed Maintenance

Search Feature

Instagram Map

User Authorization

- Users sign up for an account with an username, email and password
  - Each username and email must be unique
  - Passwords must be at least 6 characters with a combination of letters and numbers
  - Passwords will be encrypted server-side for security
- Users login with their username and password

Session Management

- ➔ Users should stay logged in once they log in
- ➔ Server will generate an authentication token for the user to send with every request
- ➔ AUTH_TOKENS(id: PRIMARY_KEY, selector: char(12), hash_validator: char(64), userid: FOREIGN KEY ‘Users’, expires: dateTime)
  - ◆ Index on selector
  - ◆ Client stores selector and validator in token, database finds selector and checks hash of validator is the same as the hash in the database

Account Recovery

- Users can reset their password if they have forgotten it
  - User requests password reset
  - Temporary authentication code sent to email
  - Code hashed into database
  - Link opens in app through MIME type
  - App verifies authentication code with database
  - Shows password reset screen and user can change password in database

User Profile Management

- Users can set their profile to Private or Public
- For personal feed, users can choose to put their profile as:
  - Feed of pictures in reverse chronological order
  - Feed of categories that will filter pictures by interest once clicked
- Profile displays bio, number of photos shared, number of followers and following
- Users can edit or delete information

Post Creation

- ➔ Users will only be able to upload pictures, videos from their photo gallery
- ➔ Users will have the option to add a caption and tag other users
- ➔ User can add hashtags and location (feature specific to our app)
  - ◆ Hashtags and location will curate posts under the same tag
- ➔ Options to categorize their uploads (food, travel, homes etc.) to organize profile feed

Activity Feed Maintenance

- Any user can like or comment on a post as long as they can see them
  - There will be options to unlike or delete comments
- User may follow or unfollow any other user
- User’s feed will show all posts newest to oldest

Search Feature

- ➔ Search by user, places, hashtags
  - ◆ Results will show suggestions that already exist
  - ◆ Hashtag relevance: heuristic like tf-idf -> interaction frequency / hashtags in post
- ➔ Allow cross filtering to refine search

Instagram Map

- ➔ On a user's profile, there will be a map tab to show a map of the user's tagged locations
- ➔ When clicking on a location, you can see a feed of all photos tagged in that location from the owner of the profile

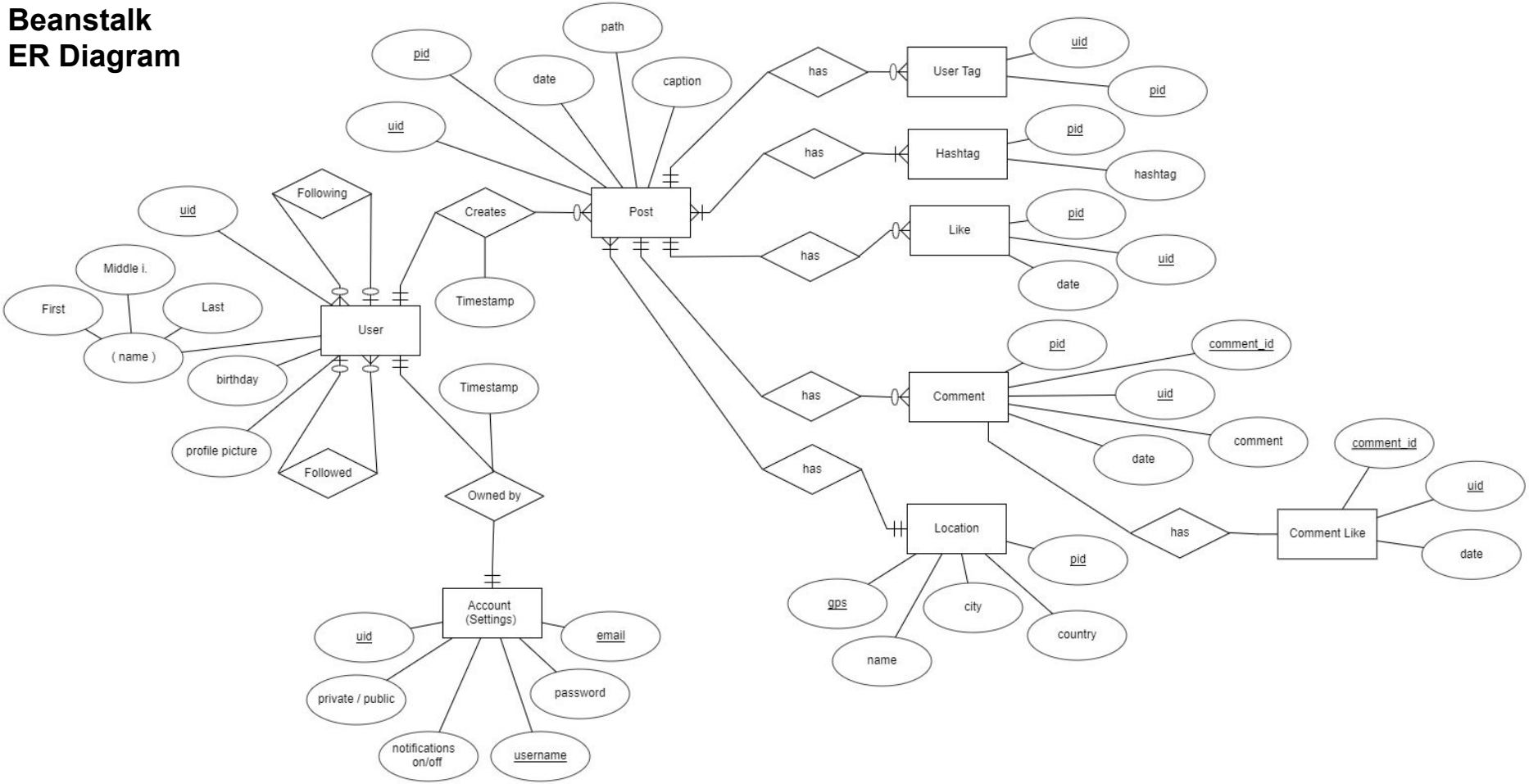
ER Model

Design Choices and Constraints

ER Diagram

ER Model Constraints

Beanstalk
ER Diagram



ER Model Constraints

- User
  - One to one: Account (Setting)
  - One to many: Post
  - Following / Followed
    Many-to-Many relationship with itself
- Post
  - One to many: User tag, Like, Comment
  - Many to many: Hashtag
  - Many to one: Location

Technology Stack

System Overview

Frontend

Backend

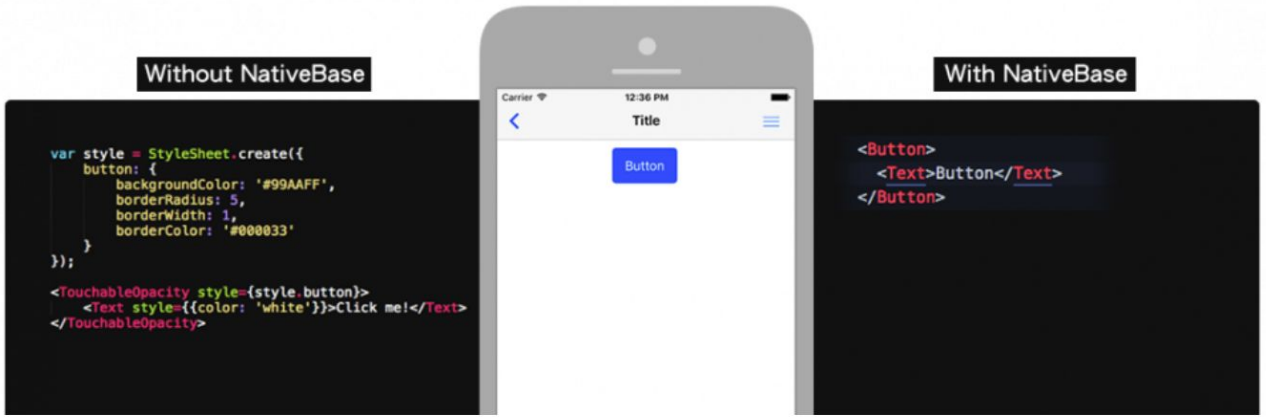
Database

Frontend

- ➔ React Native: a framework that lets you build native mobile apps using Javascript and ReactJS
- ➔ Useful features:
  - ◆ Cross-platform Development
  - ◆ Hot Reloading
  - ◆ Apps built are genuinely native
- ➔ Use Fetch, GET request to retrieve data from our database

React UI Component Libraries

- ➔ NativeBase
- ➔ Onsen UI
- ➔ Snowflake



Backend

- ➔ Flask: a Python microframework that provides simplicity, flexibility and fine-grained control
- ➔ Flask uses extensions to add functionality. A few useful extensions:
  - ◆ Flask-SQLAlchemy: SQL toolkit and Object Relational Mapper
  - ◆ Flask-Marshmallow: object serialization/deserialization
  - ◆ Flask-RESTful: quickly build REST APIs
- ➔ Using Flask we will build a RESTful API for the frontend to utilize

Backend

- ➔ Today: demonstrate basic create, read, update and delete (CRUD) functionality for users
- ➔ Currently available API endpoints
  - ◆ GET - /api/User - Retrieve all users
  - ◆ POST - /api/User - Add a new user
  - ◆ PUT - /api/User - Update an user
  - ◆ DELETE - /api/User - Delete an user

Database

- ➔ Data model is relational -> **PostgresSQL**: ACID compliant, high read and write throughput, client/server clustering RDBMS
  - ◆ High-Read Aggregation on comments, likes, followers, following
  - ◆ Will need aggregation tables
  - ◆ Pagination on results like posts and comments

Database

- ➔ Data is also graphical: How many ways / how “quickly” can I reach another user?
  - Through posts/comments/likes/tags
  - Followed/Following is a directed graph
  - Queries not based on “owns”, “has” relations, but more interested in graph connectivity
    - Graph traversals. Shortest path queries
  - Graph DBs like neo4j and ArangoDB

Live Demo

Questions?