T2A1 - Workbook

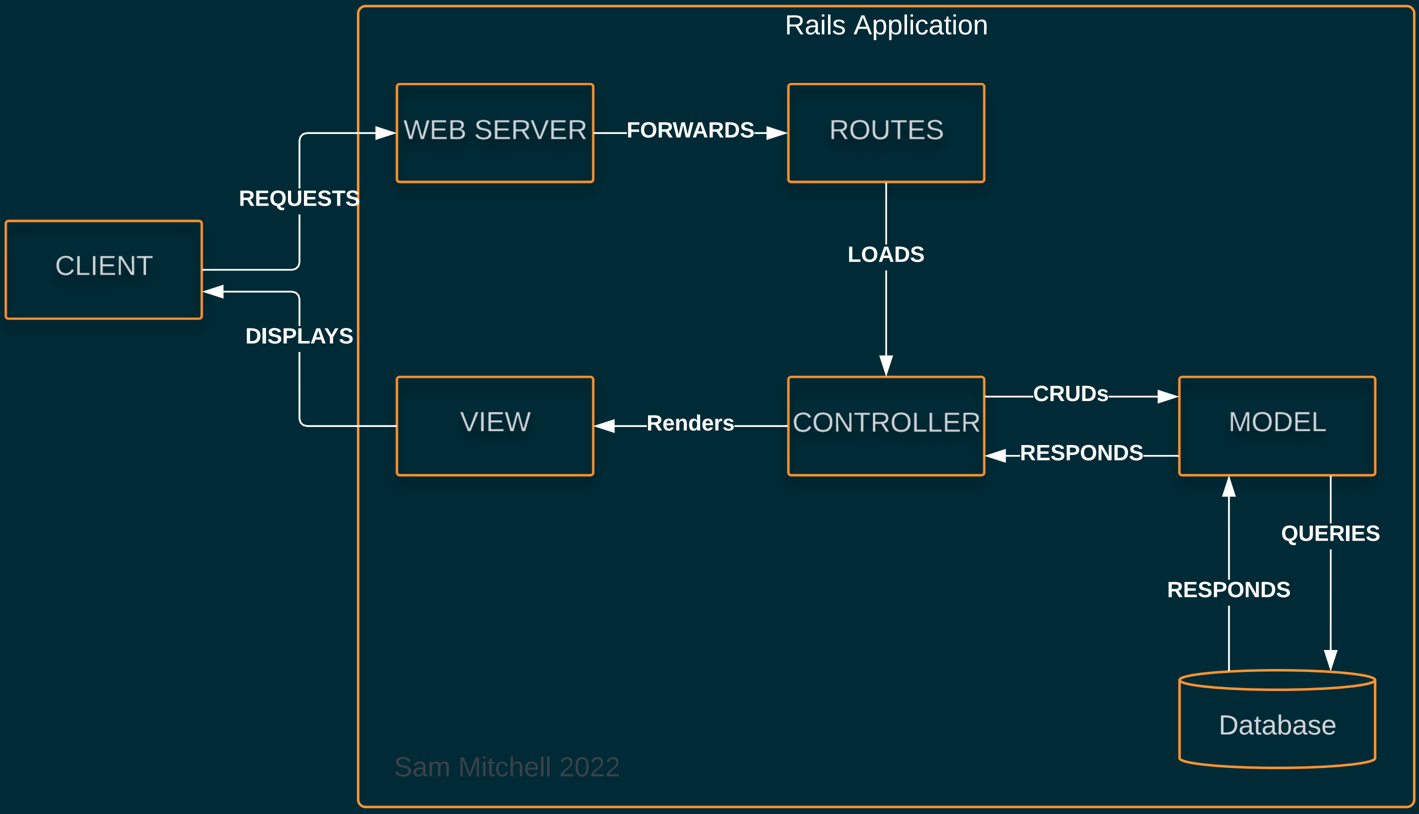
Sam Mitchell

Table of Contents

        - Unit Testing
        - Integration Testing
        - System Testing
        - Acceptance Testing
    - STLC and agile
- Security and Privacy
  - Q6 - Discuss and analyse requirements related to information system security and how they relate to the project
  - Q7 - Discuss common methods of protecting information and data and how you would apply them to the project
  - Q8 - Research what your legal obligations are in relation to handling user data and how they can be met for the project
- Relational Databases
  - Q9 - Describe the structural aspects of the relational database model. Your description should include information about the structure in which data is stored and how relations are represented in that structure.
  - Q10 - Describe the integrity aspects of the relational database model. Your description should include information about the types of data integrity and how they can be enforced in a relational database.
  - Q11 - Describe the manipulative aspects of the relational database model. Your description should include information about the ways in which data is manipulated (added, removed, changed, and retrieved) in a relational database.
- Marketplace Case Study - Ebay
  - Q12 - Conduct research into a marketplace website (app) and answer the following parts:
    - a. List and describe the software used by the app.
    - b. Describe the hardware used to host the app.
    - c. Describe the interaction of technologies within the app
    - d. Describe the way data is structured within the app
      - Relational Databases
      - Graph Network Databases
      - Document Databases
    - e. Identify entities which must be tracked by the app
    - f. Identify the relationships and associations between the entities you have identified in part (e)
      - Accounts
      - Listings
      - Items
      - Categories
      - Transaction
      - Bids
      - Review
    - g. Design a schema using an Entity Relationship Diagram (ERD) appropriate for the database of this website (assuming a relational database model)

Ruby on Rails

Q1 - Describe the architecture of a typical Rails application



MCV

The typical rails applications follows the 'Model-View-Controller' pattern which can be loosely defined as follows

- Model
  - The applications dynamic data structure
  - Responsible for data management
  - Receives user input through the controller
- View
  - Any information displayed
- Controller
  - Receives input from the user and passes commands to the view or model

In practice a Ruby Rails application has more components then just this.
The following is a breakdown of the major components in a Rails Application

Web Server

A rails application uses a 'Puma' web server by default.
Puma provides a simple, fast, multi-threaded HTTP Server for ruby applications

Routes

The purpose of the Rails router is to interpret URLs and load the appropriate controller to process the HTTP request.
The router also manages the generation of paths and URLs so that these do not need to be hardcoded in views.
The /config/routes.rb file is used to define the both the controllers and actions for HTTP requests received from the server and the path/ URL generation methods.

CONTROLLER

The controller is loaded by the router and the appropriate action or method is called. The most common actions follow the CRUD pattern using POST, GET, UPDATE, DELETE methods.
The controller acts as a middle man between the View and Model components.

MODEL

The model contains the 'Business Logic', or the code and logic that controls how data is manipulated and validated. It responds to CRUD actions called by the controller and returns information to the controller regarding the success of the operation and any data requested. It is the only component that directly interacts with the applications database.

Database

In a rails environment, the default Database Management System is SQLITE3, which is a file based DBMS. This works well as a lightweight system for development and testing however is not appropriate for deployment.

VIEW

In rails the 'View' component of the MVC pattern is managed by a combination of 'layouts', 'templates' and 'partials'.

Partials

Partials are the smallest unit of a Rails View. They can be rendered by templates or layouts and are used to simplify larger renders by breaking them down into smaller re-usable units. Partials have access to the variables of the template that called them by also can be passed additional local parameters that allow further control to the partial, like hiding an option in a context menu if the partial was called by a specific template.

Templates

Templates render the information output from a controller action. Generally each of the different actions will have their own template used to render information or forms (#index, #show, #new, #update, #delete)
Templates use a combination of simple conditionals and loops to render the supplied data. This may be done with the ues of partials.

Layouts

Layouts are used to wrap the outputs from the various controller actions in a common HTML view template. The HTML and HEAD elements of the outputted web page are generally found in the layout. It might also include headers or footers.
While the template itself is rendered through a `<%= yield %>` block in the HTML body element, additional information can be passed to the layout by specifying `content_for`. For example:

```
<!-- index.html.erb -->
<%= content_for :head do %>
<title>A page title</title>
<% end %>
<p>Page content</p>

<!-- layout.html.erb -->
<head>
  <%= yield :head %>
</head>
<body>
  <%= yield %>
</body>

<!-- output.html -->
<head>
  <title>A page title</title>
</head>
<body>
  <p>Page content</p>
</body>
```

Q2 - Identify a database management system (DBMS) commonly used in web applications (including Rails) and discuss the pros and cons of this database

NEO4J

Neo4j is a DBMS built around a graph database. A graph database is one that uses a graph structure to store data. Data is stored in the form of either an 'node', 'edge' or 'property'. A Neo4j database can be accessed using 'Cypher' (an open-source query language developed by Neo4j) over HTTP or using the binary 'Bolt' protocol.
Both the DBMS and Cypher are considered mature systems and are widely used by developers. Cypher is used by other DBMS as their query language and Neo4j can be integrated with other services such as Google's Anthos.
A Neo4J database can be hosted locally or on the cloud using the AuraDB service provided by Neo4j.

Pros

- Good for unstructured data
- Faster when searching more than one level deep
- Useful UI tools for visualising query outputs and graph structures
- Cypher is widely used in other Graph Network DBMS
- Can select large amounts of data from multiple types of nodes in one query vs SQL requiring you to join the results of multiple queries
- Useful when the relationships between objects are just as important as the data the objects contain.

Cons

- Not as adept and performing the same operation on large numbers of data elements.
- Less efficient at simple tasks over simple structured data because the data structure of an object needs to be inspected to view its structure vs a traditional relational database where the structure of a table is known before an operation.
- Not automatically supported by Ruby on Rails (however there are gems available to extend Neo4j to Rails)
- The benefits gained in interpreting data are lost if a Graph Network Database is used to simply store data.

Graph Databases

Nodes

Nodes represent an object in the database. In the example to the right

ORANGE

circles represent 'Person' nodes and

PURPLE

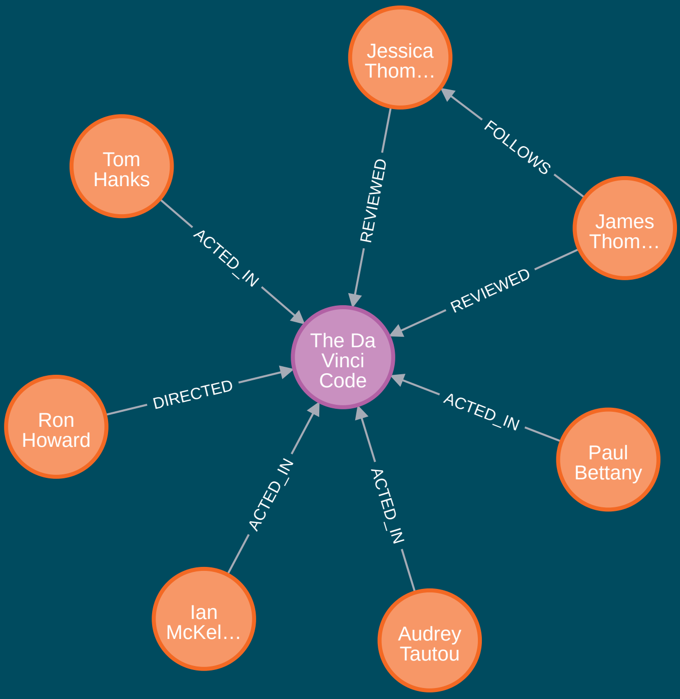
circles represent 'Movie' nodes.

Edges

Edges represent the relationships between nodes. Edges are labeled with types, in the example the right these types are [DIRECTED, ACTED_IN, REVIEWED, FOLLOWS].

Properties

Properties are optionally used to describe both nodes (objects) and edges (relationships). They are stored as a key: value pair.



A `Person` node may contain properties such as `name` , `date_of_birth` or `gender` , a `Movie` node may contain properties such as `title` or `release_date` , a `REVIEWED` edge may contain properties for `rating` or `reviewed date` .

Project Management

Q3 - Discuss the implementation of Agile project management methodology

The implementation of the Agile methodology requires more then just putting in place frameworks and using the correct terminology. It requires a cultural shift.

The Manifesto for Agile Software Development

The values and principles defined by the "Manifesto for Agile Software Development" are as follows

Agile Software Development Values

We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

> - **Individual and interactions** *over* processes and tools
> - **Working software** *over* comprehensive documentation
> - **Customer collaboration** *over* contract negotiation
> - **Responding to change** *over* following a plan

"That is to say, while both sides have value and the items on the right should be considered, the authors of the manifesto chose to tip the balance in favor of the items on the left."

Agile Software Development Principles

The Manifesto for Agile Software Development is based on twelve principles

> 1. Customer satisfaction by early and continuous delivery of valuable software.
> 2. Welcome changing requirements, even in late development.
> 3. Deliver working software frequently (weeks rather than months)
> 4. Close, daily cooperation between business people and developers
> 5. Projects are built around motivated individuals, who should be trusted
> 6. Face-to-face conversation is the best form of communication (co-location)
> 7. Working software is the primary measure of progress
> 8. Sustainable development, able to maintain a constant pace
> 9. Continuous attention to technical excellence and good design
> 10. Simplicity (the art of maximizing the amount of work not done) is essential
> 11. Best architectures, requirements, and designs emerge from self-organizing teams
> 12. Regularly, the team reflects on how to become more effective, and adjusts accordingly

Implementation of Agile Values and Principles

People over processes

The Agile methodology values the team and it's members of planning processes or tools.
The developers' knowledge, experience, thought processes, and how an individual works with other individuals in the team is more important then any framework put in place to manage workflow. Without competent and motivated individuals working in a cohesive team, work output and quality will not be to the level it could be.
By engaging the team members in the planing process a manager can draw on technical expertise from senior developers, allow a junior developer not indoctrinated into one way of thinking to question the way things are done or draw on group knowledge to estimate durations for tasks.

Documentation "Just barely good enough"

A 1000 page manual on how to service a car is useless without the car. Just as complete documentation on a incomplete program is useless. This does not mean that a development team should forgo documenting their software but instead it should be "just barely good enough". Just good enough to describe use cases, model and methods. Just good enough to allow a new member to the team to understand the software.

Being adaptable

There's a saying in the Army that no plan survives contact with the enemy, but this doesn't mean you don't plan at all, it means you need to be prepared to make changes to your plan on the fly based of intelligence gathered.

Project management is the same. You can spend weeks planning an entire project from start to finish before a single line of code is written. Assigning individuals to teams, and teams to tasks. Detailing milestones and deadlines only for a pandemic to spread, causing lockdowns and requirements to work from home. Now not all that planning is wasted but a significant part of it will no longer be relevant or require adjusting.
This isn't to say that an adept project manager would not be able to handle this shift in circumstances but by planning in shorter cycles and changing circumstances and requirements being the normal means a team working under Agile methodology will be able to stop and change direction faster because they are carrying less administrative momentum.

Short Cycles

By working in an iterative process that ends with a working piece of software at the end of each cycle a team can maintain customer satisfaction by enabling them to see the development and growth of the software. It also enables the customer to provide feedback which can be fed backing into the planning cycle to prioritise work or change requirements.

Q4 Provide an overview and description of a standard source control workflow

One of the most widely used source control systems today is Git, which is a Distributed Revision Control System (DRCS). A DRCS allows multiple users to maintain a full working copy of the repository locally to work on and any changes can be disseminated to their peers to use in their own local repository.
There is no one 'standard' source or version control workflow because different workflows may be more appropriate for different sized projects or different software being used to manage the workflow. It's not a one size fits all situation. However, because there are only so many 'commands' or actions available there is a lot of common ground between major workflow models and a few best practices from years of experience.

The Main Branch

Nearly all source control workflows utilise a main branch as part of their model however different models disagree as to whether or not work should be directly committed to it.

Development Branch

Some workflows use a development branch, a branch which is used for the development of new features. When the code is ready to be deployed it will either be merged back into the master branch or into a release branch.

Feature Branches

Most model recommend the use of a feature branch. This is a branch from the main or development branch of code used to work on new features. Once the feature is complete it is merged back into the originating branch. Feature branches should be short lived in order to make the merging back into their originating branches as simple as possible.

Release Branch

Some workflows use a release branch to separate a version of software for production. In some cases however the main branch is used for deployment and tagged with a version.

Commits & pushes

All workflows involve the use of a local repository. Changes are made to this local repository. When enough changes have been made, files are committed together in logical groupings with informative messages about the changes made. In order for the changes to be shared they need to be 'pushed' to the master branch.

Examples

Flow Variants	Trunk Based Development
The original 'Gitflow' (has fallen out of favour in recent years due to complexity)	TBD for small teams. (incomplete features are disabled for release through 'feature flags')
'Github Flow'. Used by Github for a continuos development environment	Scaled TBD (Useful for managing code review in larger teams)

Q5 - Provide an overview and description of a standard software testing process (e.g. manual testing)

The Software Testing Life Cycle (STLC)

There are four stages to testing in the Software Testing Life Cycle. Unit Testing, Integration Testing, System Testing and Acceptance Testing. These can be further separated into two categories.

Verification Stages	Validation Stages
Unit Testing	Acceptance Testing
System Testing	
Integration Testing	

Unit Testing

In this stage individual components are tested. In OOP the units being tested may be a single class but may be individual functions if procedural programming is being used.
All units are tested to check functionality and this stage can occur as changes are made to code to ensure that any errors are caught early to reduce the debugging overhead later in the development process.
Testers require a detailed understanding of functions and classes to carry out this stage of testing. This type of testing is called 'White Box Testing'.

Integration Testing

In this stage of testing, individual components are tested again individually and then tested together as a group. The aim of this stage is to assess how well all the components work together and identify any issues in the behaviours of interacting components.
Groups of working components should be combined into progressively larger groups until the entire system is being tested.

System Testing

System Testing is hte final stage in the verification stage of testing. The aim of this stage is to verify the system meets all requirements and specifications.

Acceptance Testing

Acceptance Testing is carried out in the final stages before the release of a system. It may include:

- End user testing
- Contractual acceptance testing
- Alpha and Beta testing

This stage of testing is often carried out using 'Black Box Testing', which is where the tester has no knowledge of the inner workings of the system.

STLC and agile

In agile the STLC is broken down into 6 steps.

1. **Requirement Analysis**
   Understand the specifications for the system, what are the expected outputs, what are the priorities?
2. **Test Planning**
   Prepare documentation, carry out time analysis and assign tasks to teams.
3. **Test Case Design and Development**
   Create test cases and prepare scripts for the automation of testing.
4. **Test Environment Setup**
   Understand minimum system requirements. Setup and smoke test the testing environments (Smoke testing is a type of testing designed to find basic and critical bugs. The term originates from an older hardware test where the device passed the test if it didn't catch fire when turned on.)
5. **Test Execution**
   Run test cases, identify and log failures with details, create bug fixes and test again
6. **Test Closure**
   Verify that all tests are completed, document the testing results and prepare the test closure report.

Security and Privacy

Q6 - Discuss and analyse requirements related to information system security and how they relate to the project

Q7 - Discuss common methods of protecting information and data and how you would apply them to the project

Q8 - Research what your legal obligations are in relation to handling user data and how they can be met for the project

Relational Databases

Q9 - Describe the structural aspects of the relational database model. Your description should include information about the structure in which data is stored and how relations are represented in that structure.

Q10 - Describe the integrity aspects of the relational database model. Your description should include information about the types of data integrity and how they can be enforced in a relational database.

Q11 - Describe the manipulative aspects of the relational database model. Your description should include information about the ways in which data is manipulated (added, removed, changed, and retrieved) in a relational database.

Marketplace Case Study - Ebay

Q12 - Conduct research into a marketplace website (app) and answer the following parts:

a. List and describe the software used by the app.

Ebay currently use a wide variety of in house open source software for almost all of their operations.
Their account on github.com has over 160 repositories.
The website itself runs on JavaScript (MarkoJS), CSS and HTML.
Oracle, MongoDB and its in house NuGraph built on the open source JanusGraph are used as DBMS.

b. Describe the hardware used to host the app.

Ebay uses in house designed servers to distribute 500PT of storage and process 300 billion queries every day.
Each pf Ebay's search servers are comprised of 2 Intel 6138 processors, each with 64GB of RAM.
Each server has multiple 2TB hardrives in RAID 10 and is capable of storing the database index.
Each server is connected to Ebay's network infrastructure via 2 x 10GBps network ports.
Each server fits into a 1U space in a standard 19" rack.
It is not specified how many server nodes Ebay uses however their system is designed to be able to add or remove server nodes as need to the number is probably always growing however given a standard 19" rack has 48U of space and Ebay is likely to have multiple of these in each server cluster and server clusters distributed all over the US and globe it is possible to imagine just how many of these server nodes there are.
This is just the storage server, in addition Ebay would require web servers and data servers for their operations.

c. Describe the interaction of technologies within the app

Ebay uses a fairly standard front end stack of HTML, CSS and JavaScript in the form of MarkoJS, behind the scenes ebay utilises a a number of different technologies for their operations. They use a combination of traditional Relational Databases, Graph Network Databases and image databases. These are combined with powerful Machine Learning algorithms to provide users recommendations, enable fraud control and the ability to search for listings based on an image from their device.

d. Describe the way data is structured within the app

Ebay uses multiple data structures depending on what the data is used for.

Relational Databases

Used to store hashed indexes to enable faster lookups in the Graph Network

Used to store customer/ seller coupons

Graph Network Databases

Used to store the bulk of information regarding customers, listings and sales.

Enables Machine Learning algorithms to group records more easily.

Document Databases

Ebay uses dedicated databases to store images

e. Identify entities which must be tracked by the app

Account	Item	Images	Category
Address	Price	Bids	Review
Listings	Transaction		

f. Identify the relationships and associations between the entities you have identified in part (e)

Accounts

Can have listings

Can have addresses (delivery address or item location)

Can a list of transactions (buy or sell)

Can have reviews (buyer or seller)

Listings

Must have a seller

Must be a type of item

Must have images

Can have a buy price

Can have bids

Can be reviewed

Items

Must belong to at least one category

Categories

Can have parent or children categories

Can have items associated

Transaction

Must have a listing

Must have a buyer

Must have a price

Must have a delivery address

Bids

Must belong to a listing

Must belong to a buyer

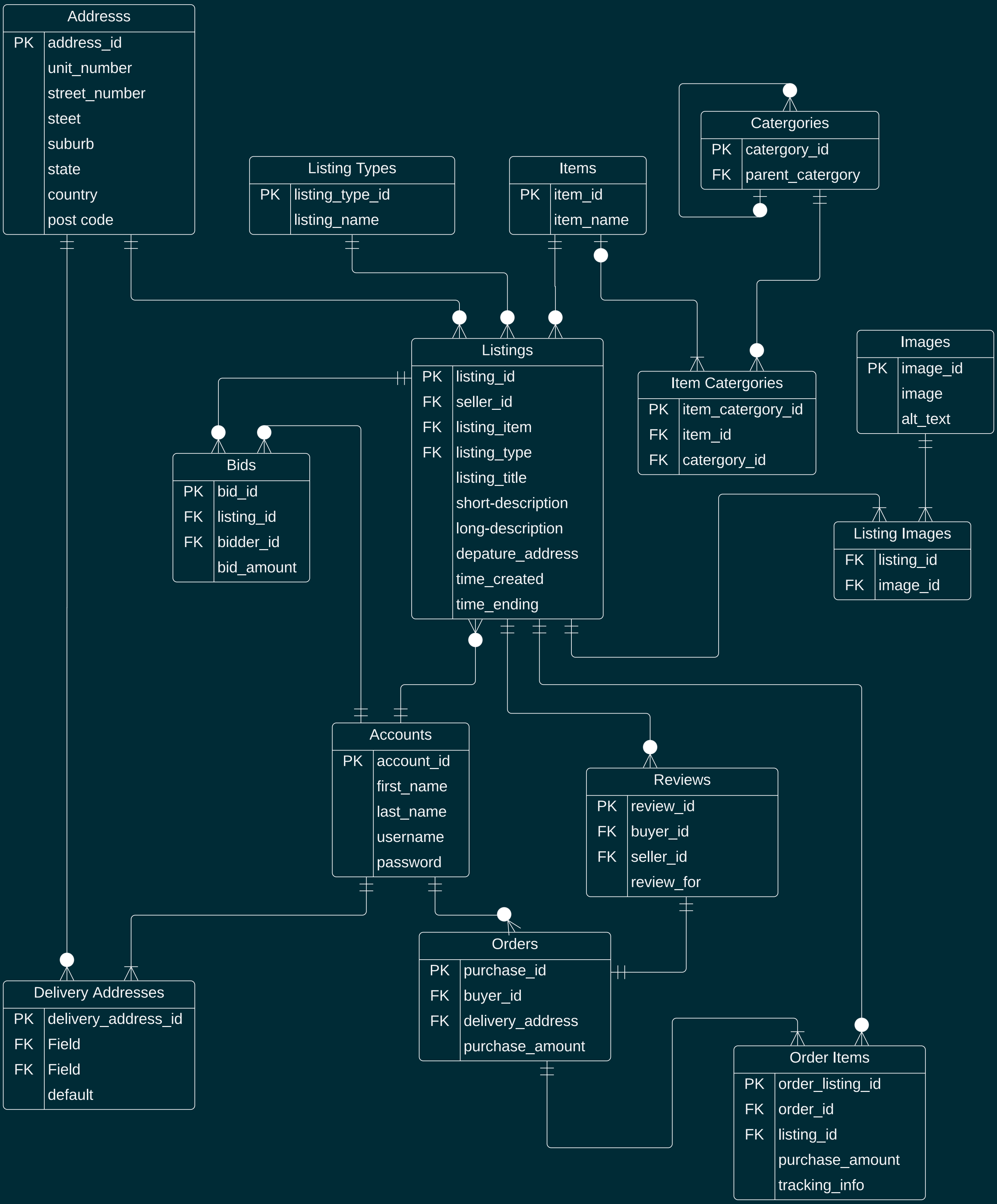
Must have a price

Review

Can be about a buyer, seller, or listing

Can be from a buyer or seller

g. Design a schema using an Entity Relationship Diagram (ERD) appropriate for the database of this website (assuming a relational database model)


Addresss
PK address_id
unit_number
street_number
steet
suburb
state
country
post code
Listing Types
PK listing_type_id
listing_name
Items
PK item_id
item_name
Catergories
PK catergory_id
FK parent_catergory
Listings
PK listing_id
FK seller_id
FK listing_item
FK listing_type
listing_title
short-description
long-description
depature_address
time_created
time_ending
Item Catergories
PK item_catergory_id
FK item_id
FK catergory_id
Images
PK image_id
image
alt_text
Bids
PK bid_id
FK listing_id
FK bidder_id
bid_amount
Listing Images
FK listing_id
FK image_id
Accounts
PK account_id
first_name
last_name
username
password
Reviews
PK review_id
FK buyer_id
FK seller_id
review_for
Orders
PK purchase_id
FK buyer_id
FK delivery_address
purchase_amount
Delivery Addresses
PK delivery_address_id
FK Field
FK Field
default
Order Items
PK order_listing_id
FK order_id
FK listing_id
purchase_amount
tracking_info