Work Tracking - GitHub Issues with GitKraken

- Will use branching to avoid merge conflicts

Database Design/ Schema - MySQL Workbench

Create schema trivia_night

- Create table category
    - category_id primary key auto_increment
- Create table trivia_question
    - trivia_id primary key auto_increment
    - question varchar(300) not null
    - category_id foreign key not null
- Create table answer
    - answer_id primary key auto_increment
    - question_id int foreign key not null
    - answer varchar(100) not null
    - isCorrect int not null
    - Category_id foreign key
- Create table user
    - user_id primary key auto_increment
    - username varchar(30) not null
    - password varchar(45) not null
    - questions_answered int
    - questions_correct int

Create script to populate trivia_night - VS Code

- Use API from https://opentdb.com/
- Call API to receive questions and answers in JSON format
- Add questions to database using addQuestion
- Add answers to database using addAnswer with matching questionId as foreign key

Create schema trivia_night_test

- Create table category
    - category_id primary key auto_increment
- Create table question
    - trivia_id primary key auto_increment
    - question varchar(300) not null
    - category_id foreign key not null
- Create table answer
    - answer_id primary key auto_increment
    - question_id int foreign key not null
    - answer varchar(100) not null
    - isCorrect int not null
    - Category_id foreign key

- Create table user
    - user_id primary key auto_increment
    - username varchar(30) not null
    - password varchar(45) not null
    - questions_answered int
    - questions_correct int
- Create known good state
    - Delete from category
    - Alter table category auto_increment = 1
    - Delete from question
    - Alter table question auto_increment = 1
    - Delete from answer
    - Alter table answer auto_increment = 1
    - Delete from user
    - Alter table user auto_increment = 1
    - Populate with sample data

Implement Models - IntelliJ

- Create User Model
    - int userId
    - String username
    - String password
    - int numAnswered
    - int numCorrect
    - Getters and Setters for the above data
- Create Answer Model
    - int answerId
    - boolean isCorrect
    - Getters and Setters for the above data
- Create Question Model
    - int questionId
    - String question
    - List <Answer> answers
    - String category
    - Getters and Setters for the above data

Data Layer - IntelliJ

- Create QuestionJdbcTemplateRepository
    - Create QuestionMapper
        - mapRow method

        - findByCategory (questionId : int) : List<Question>
        - selectQuestions (questions : List<Question>) : List<Question>
            - will randomizes questions list order
- Create KnownGoodState class in test
- Create QuestionJdbcTemplateRepositoryTest
    - @Autowired QuestionJdbcTemplateRepository repository
    - @Autowired KnownGoodState knownGoodState
    - @BeforeEach setup()
    - Test shouldFindByCategory
- Create UserJdbcTemplateRepository
    - Create UserMapper
        - mapRow method
    - findById(userId : int) : User
    - findAll() : List<User>
    - create (user : User) : User
    - update (user : User) : boolean
    - delete (userId : int) : boolean
- Create UserJdbcTemplateRepositoryTest
    - @Autowired UserJdbcTemplateRepository repository
    - @Autowired KnownGoodState knownGoodState
    - @BeforeEach setup()
    - Test CRUD should scenarios
- Create AnswerJdbcTemplateRepository
    - CreateAnswerMapper
        - mapRow method
    - findByQuestionId (questionId : int) : List <Answer>
    - findByAnswerId (answerId : int) : Answer
    - addAnswer (answer : Answer) : Answer
- Create AnswerJdbcTemplateRepositoryTest
    - @Autowired AnswerJdbcTemplateRepositoryrepository
    - @Autowired KnownGoodState knownGoodState
    - @BeforeEach setup()
    - Test shouldFindByQuestion
    - Test shouldFindByAnswerId

Domain Layer - IntelliJ

- Create UserService
    - repository UserJdbcTemplateRepository
    - findById (int userId)
        - Returns User
    - findAll ()
        - Returns List <User>
    - create (User user)

        - Returns User
    - update (User user)
        - Returns boolean
    - delete (int userId)
        - Returns boolean
    - private validateUser (User user)
        - Returns Result
- Create AnswerService
    - Repository AnswerJdbcTemplateRepository
    - findByQuestion (int questionId)
        - Returns List <Answer>
    - findById (int answerId)
        - Returns Answer
- Create QuestionService
    - QuestionJdbcTemplateRepository repository
    - findByCategory (int categoryId)
        - Returns List <Question>
- Create Result
    - List <String> messages
    - User payload
    - Getters and setters for above data
    - isSuccess ()
        - Returns boolean

UI Layer - IntelliJ

- Create ApiController
    - QuestionService questionService
    - AnswerService answerService
    - UserService userService
    - findUserById (int userId)
        - Returns User
        - Mapping: GET ‘/user/{userId}’
    - findAllUsers ()
        - Returns List <User>
        - Mapping: GET ‘/user’
    - createUser (User user)
        - Returns ResponseEntity <Object>
        - Mapping: POST ‘/user’
    - updateUser (User user)
        - Returns ResponseEntity <Object>
        - Mapping: PUT ‘/user/{userId}’
    - deleteUser (int userId)
        - Returns ResponseEntity <void>

        - Mapping: DELETE ‘/user/{userId}’
    - findAnswerByQuestion (int questionId)
        - Returns List<Answer>
        - Mapping: GET ‘/answer/question/{questionId}’
    - findAnswerById (int answerId)
        - Returns Answer
        - Mapping: GET ‘/answer/{answerId}’
    - findQuestionsByCategory (int categoryId)
        - Returns List <Question>
        - Mapping: GET ‘/question/{categoryId}’
- Create ErrorResponse
    - String message
    - getMessage ()
        - Returns String
    - Build (Result<User> result)
        - Returns ResponseEntity<Object>
- Create GlobalExceptionHandler
    - handleException (DataAccessException ex)
        - Returns ResponseEntity<ErrorResponse>
    - handleException (Exception ex)
        - Returns ResponseEntity<ErrorResponse>

Implement React Components - VS Code

- Create User page
    - Simple form to capture new user information
- Create New Game page
    - Populated by list of categories for user to select
- Create Question page
    - Single page but the content will be rendered based on the question ID
- Create Leaderboard page
    - Populated with top 5 players and their number of correct answers
- Create Home page
    - Logo in the center screen with navbar on top to drive the rest of the application
- Create Login page
    - Simple form to capture user information
- Create Summary page
    - Use state to tell when all of the questions are answered, which will trigger this display