Initial Architecture Document

Team Number: 14
Team Members:

- Daniel Butler
- Charley Findling
- Skylar Franz
- Jack Gerety
- Beckett Malinowski

Project Name: Coach App
Project Synopsis: A task management app with gamification and an interactive coach.

Architecture Description:

Framework:

Using the Laravel framework (language: PHP) for apps and websites, the project will be split up into five main components:

Migrations: Files used to create SQL tables.
Example: \database\migrations\0001_01_01_000003_create_tasks_table.php

These will define the SQL tables that the application will store data in.

(Eloquent) Models: Objects linked to the SQL tables and application.
Example: Tasks, Tags, Coaches, Users

These are objects linked to their associated SQL table, where the object variables are associated with columns/entries in the SQL database. For example, the Task Model is linked to the SQL table "tasks" and has the variables "name" and "description," which are columns in that table. Each SQL entry/record in that table is a different task following the Task Model.

Models can have parent-child or many-to-many relationships with other Models (i.e tasks can have multiple tags, a tag can be associated with multiple tasks, done through pivot tables and foreign keys.

Views: Web pages the user accesses.
Example: taskpage.blade.php, completed.blade.php

These are web pages in the application that will display the Models (SQL entries) and variables using Blade formatting. They will also allow the user to modify Models through HTTP form submission requests (POST, PATCH, DELETE).

Routes: Link web pages to URLs, functions, and Controllers.
Example: web.php

When a user clicks on a link (whether to a view or a form submission), it's defined as a route with a URL and what view (and Models) it returns (if it's a GET request). Routes have functions (that will mostly come from Controllers) that will modify the given Model in the SQL database (if the route is a POST/PATCH/DELETE request).

Controllers: Connect views and form requests to SQL database and routes.
Example: TaskController, TagController, CoachController

These store functions like creating, deleting, and updating models; and are connected to Routes.

Now that the framework components are defined, it's time to define the Coach App itself.

Application Outline:

Overall, the application will be about users creating and completing tasks: half of the main screen being tasks and the other half being an interactive "coach" that will react to what the user is doing. When the user completes tasks they get coins they can use to buy customization options for the application.

Tasks:
The main Model the user will interact with. Tasks can be created, deleted, updated, completed, and have Tags assigned to them. The user can see incomplete tasks and past, completed tasks.

Later development will include Tasks being assigned to days, i.e. reoccurring and repeatable Tasks (e.g. "Go for a walk" every two days, or every Saturday, etcetera).

- ID
- Name
- Description (Notes)
- Completion Status (Whether the task is completed or not)
- Color (Customization)
- User_ID: Which User does the task belong to (User is parent, task is child)
- (Pivot table/Many-to-Many relationship): Tags, Days
- Repeated Days (When the task will repeat again)
- Time Stamps: Date Created, Date Updated, Date Completed

Tags: (Skills/Journeys)
Tags are a model designed to organize and group tasks together. For example the User could create an “Exercise” tag that can be assigned to the “Do 20 push-ups” and “Do 20 Jumping Jacks” tasks. A task can be assigned multiple Tags (Like “Exercise” and “Go Outside”), making the Tag to Task model a many-to-many relationship.

While they are called Tags (for now), think of them as more as “Projects” – Tags will have their own page where you can see their associated tasks, and they can be completed as well (soft removal). However, completing a Tag won’t complete their associated tasks

- ID
- Name
- Description (Notes)
- Completion Status (Whether the task is completed or not)
- Color (Customization)
- User_ID: Which User does the tag belong to (User is parent, tag is child)
- (Pivot table/Many-to-Many relationship): Tasks
- Time Stamps: Date Created, Date Updated, Date Completed

Days:
A model so that the user can see past days and what tasks they completed on that date. Can also be used to plan for future tasks. When the application reads a new date from the desktop, it should add a new day, populated with the tasks reoccurring/planned on that day.

- ID
- Date

• (Pivot table/Many-to-Many relationship): Tasks

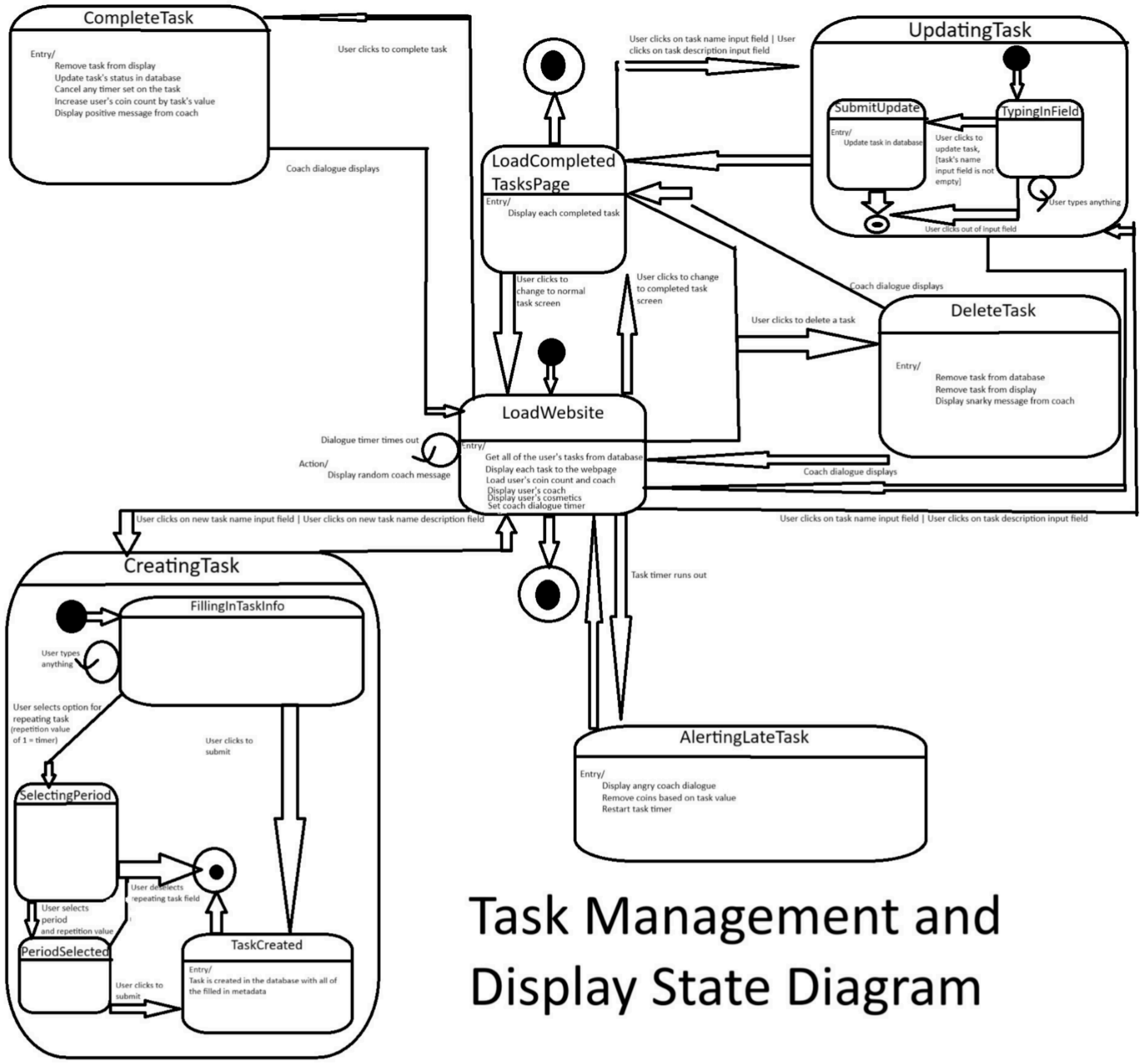
User/Game State/Shop:
A model to keep track of which user is using the application, and the game state of that user (e.g. the number of coins obtained/which items unlocked). When the user buys an item, it'll send a form request to mark that item as unlocked.

• ID (User)
• Number of Coins
• Current UI (custom style)
• Items (bought or not yet unlocked)
• Time Stamps: Date Created, Date Updated

Coach:
Think of the Coach as actually being a bunch of dialogue with "state" tags attached (such as "Task Completed" or "Suggestion") in an SQL table. When triggered (either by form or by JavaScript), the coach should pull dialogue from one of the states and output that to the user.

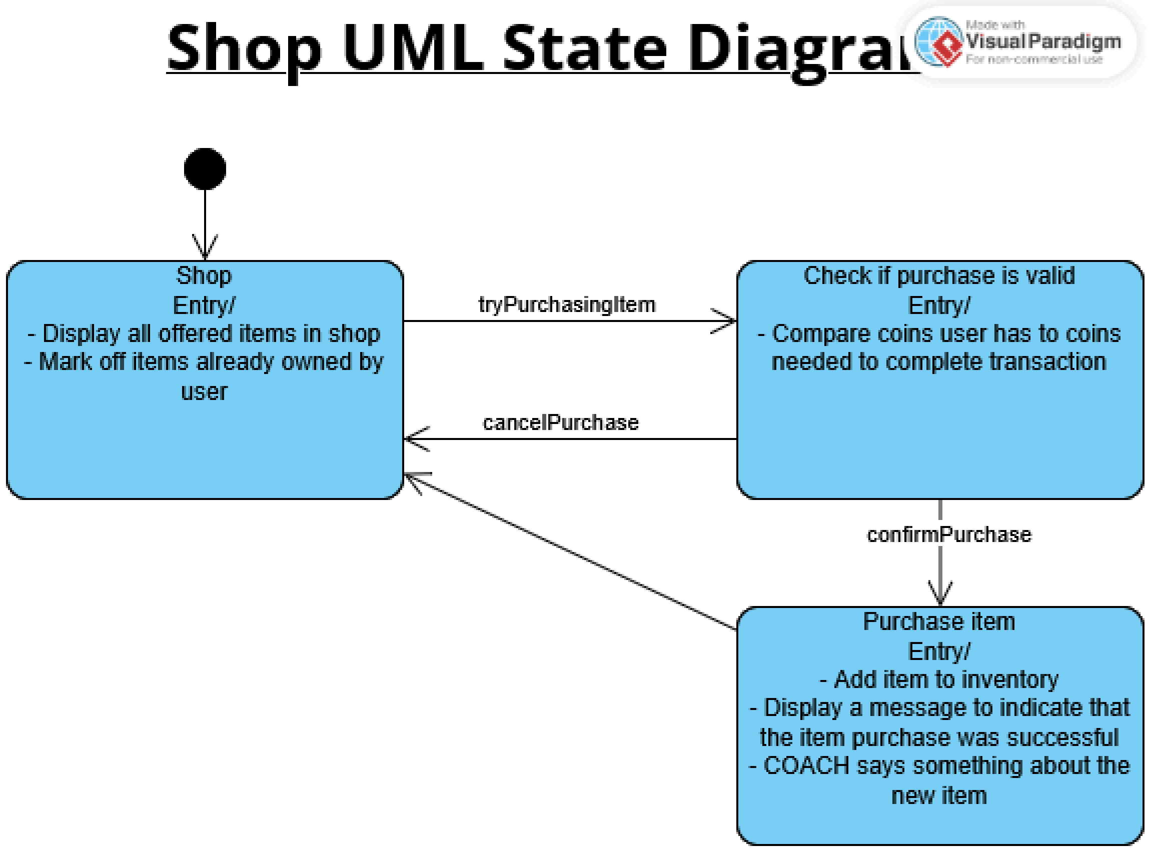
• ID (Dialogue)
• Coach ID (In case multiple coaches are implemented)
• State (or Prompt, Trigger)
• Dialogue

Coach reaction images will be handled on the front end (a.k.a views) by retrieving the state variable (Blade) and then loading the appropriate assets.


CompleteTask
Entry/
Remove task from display
Update task's status in database
Cancel any timer set on the task
Increase user's coin count by task's value
Display positive message from coach
User clicks to complete task
Coach dialogue displays
LoadCompleted TasksPage
Entry/
Display each completed task
User clicks on task name input field | User clicks on task description input field
UpdatingTask
SubmitUpdate
Entry/
Update task in database
TypingInField
User clicks to update task, [task's name input field is not empty]
User types anything
User clicks out of input field
Coach dialogue displays
User clicks to change to normal task screen
User clicks to change to completed task screen
User clicks to delete a task
DeleteTask
Entry/
Remove task from database
Remove task from display
Display snarky message from coach
LoadWebsite
Entry/
Get all of the user's tasks from database
Display each task to the webpage
Load user's coin count and coach
Display user's coach
Display user's cosmetics
Set coach dialogue timer
Dialogue timer times out
Action/
Display random coach message
Coach dialogue displays
User clicks on task name input field | User clicks on task description input field
User clicks on new task name input field | User clicks on new task name description field
Task timer runs out
CreatingTask
FillingInTaskInfo
User types anything
User selects option for repeating task (repetition value of 1 = timer)
User clicks to submit
SelectingPeriod
User deselects repeating task field
User selects period and repetition value
PeriodSelected
User clicks to submit
TaskCreated
Entry/
Task is created in the database with all of the filled in metadata
AlertingLateTask
Entry/
Display angry coach dialogue
Remove coins based on task value
Restart task timer

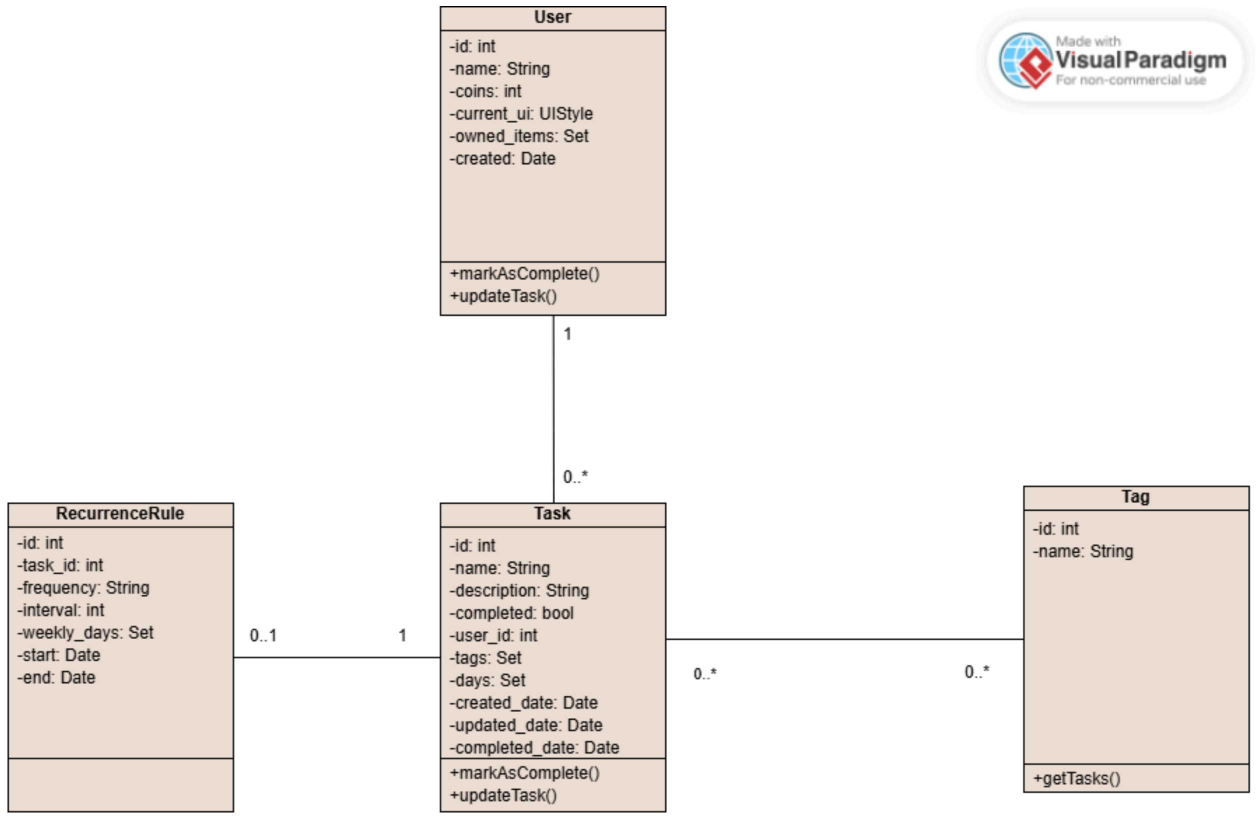
Task Management and Display State Diagram

Shop UML State Diagram



Made with Visual Paradigm For non-commercial use
Shop
Entry/
- Display all offered items in shop
- Mark off items already owned by user
tryPurchasingItem
cancelPurchase
Check if purchase is valid
Entry/
- Compare coins user has to coins needed to complete transaction
confirmPurchase
Purchase item
Entry/
- Add item to inventory
- Display a message to indicate that the item purchase was successful
- COACH says something about the new item

User and Task UML Class Diagram



User
-id: int
-name: String
-coins: int
-current_ui: UIStyle
-owned_items: Set
-created: Date
+markAsComplete()
+updateTask()
Made with Visual Paradigm For non-commercial use
1
0..*
RecurrenceRule
-id: int
-task_id: int
-frequency: String
-interval: int
-weekly_days: Set
-start: Date
-end: Date
0..1
1
Task
-id: int
-name: String
-description: String
-completed: bool
-user_id: int
-tags: Set
-days: Set
-created_date: Date
-updated_date: Date
-completed_date: Date
+markAsComplete()
+updateTask()
0..*
0..*
Tag
-id: int
-name: String
+getTasks()