Electronic Golf League Scheduler

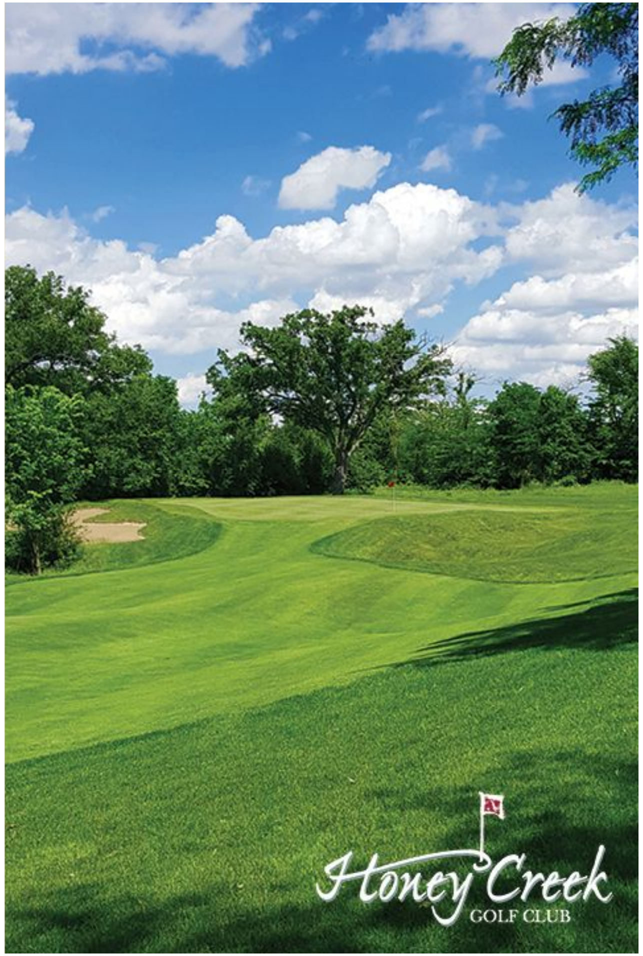
Ethan Evans, Aidan Andreas, Brady Zalasky, Nick Landon, Maxwell Farver

Team: sddec21-03
Website: https://sddec21-03.sd.ece.iastate.edu
Advisor: Mai Zheng
Client: Tina Prouty

Project Overview

Problem Statement

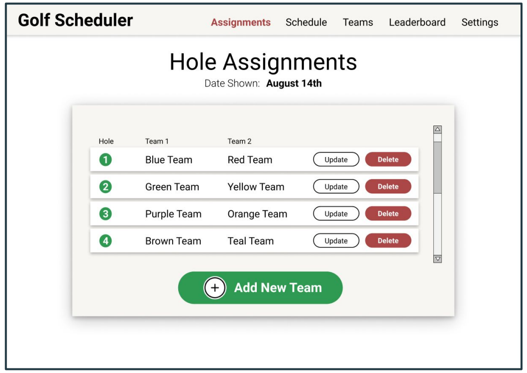
- Create a web application to handle golf teams and matchups
- Current system can't handle the current club size
- New system should be user friendly and easy to maintain
- Ideally should cost no more than a few dollars a month



Team Member Roles

Frontend	Brady
Backend	Aidan and Nick
Full-Stack	Max and Ethan

Wireframe Designs

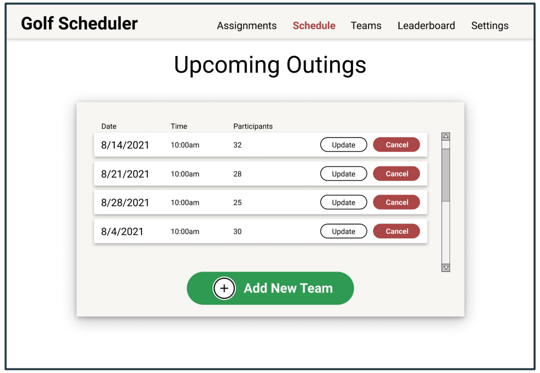

Golf Scheduler — Assignments Schedule Teams Leaderboard Settings

Hole Assignments

Date Shown: **August 14th**

Hole	Team 1	Team 2		
1	Blue Team	Red Team	Update	Delete
2	Green Team	Yellow Team	Update	Delete
3	Purple Team	Orange Team	Update	Delete
4	Brown Team	Teal Team	Update	Delete

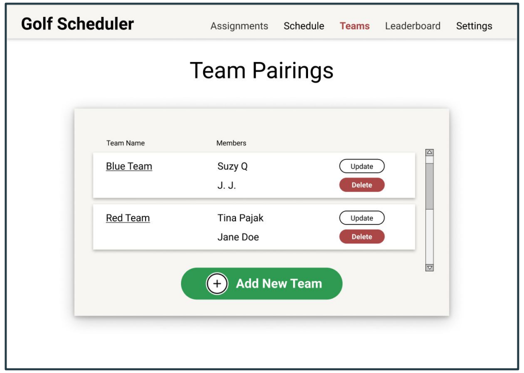
Add New Team


Golf Scheduler — Assignments Schedule Teams Leaderboard Settings

Upcoming Outings

Date	Time	Participants		
8/14/2021	10:00am	32	Update	Cancel
8/21/2021	10:00am	28	Update	Cancel
8/28/2021	10:00am	25	Update	Cancel
8/4/2021	10:00am	30	Update	Cancel

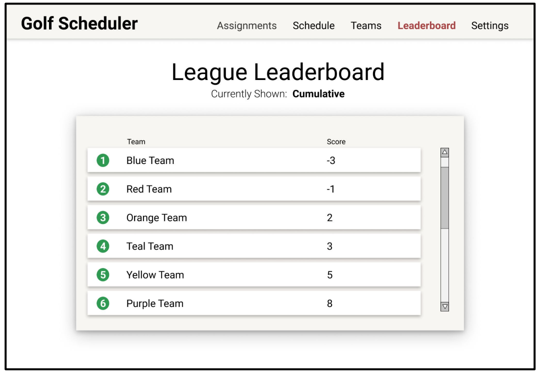
Add New Team


Golf Scheduler — Assignments Schedule Teams Leaderboard Settings

Team Pairings

Team Name	Members	
Blue Team	Suzy Q	Update
	J. J.	Delete
Red Team	Tina Pajak	Update
	Jane Doe	Delete

Add New Team


Golf Scheduler — Assignments Schedule Teams Leaderboard Settings

League Leaderboard

Currently Shown: **Cumulative**

	Team	Score
1	Blue Team	-3
2	Red Team	-1
3	Orange Team	2
4	Teal Team	3
5	Yellow Team	5
6	Purple Team	8

Project Requirements

- Functional
  - User input for teams and players
  - Leaderboard to keep track of score
  - Schedule for upcoming outings
  - Creates a schedule for when a new outing is created
- Economic
  - Minimal or low monthly cost
- User Interface
  - Simple to use
  - Features clearly defined
  - Modern and professional design

Technical Considerations

- Cost played an important role
- AWS preferred cloud provider
  - Generous free tier
- DynamoDB
  - Low latency
- NodeJS
  - Handle cold starts well
- Netlify
  - Simplicity of deployment and cost

Development Approach

- Utilized a Trello board for Project Management
  - Backlog, In Progress, Ready for Review, Done
  - Color coding and assigning tasks
- Slack was form of a communication to bring up questions/concerns
- Development had to be done on a feature branch
  - Code must get approved by a reviewer
  - Reviewer was responsible for local testing of new feature

Market Survey

- Competing applications: PlayPass and GolfSoftware
  - Cheap or free
  - Ability to schedule a tournament or round robin league
- Our Application
  - Includes the features competing applications offer
  - Built for Tina’s league specifically
  - Automatic pairing shuffling and reassignment
  - Tracks weekly scores, winners, and standings

Resource Requirements

- Project hosted on AWS at no cost
- Application built with free resources
    - Node.js, DynamoDB, React, Netlify
- Since we are using resources free to the public only requirement is that we have internet access

Potential Risks & Mitigation

Identified our biggest risks as:

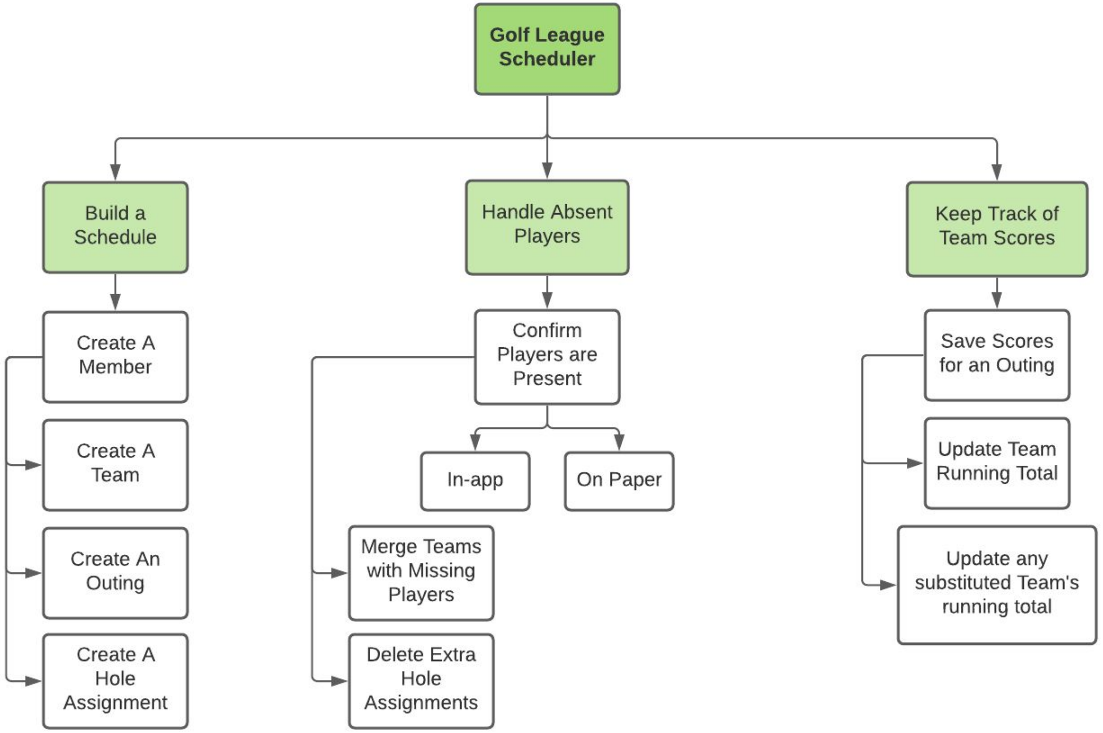
- Environment setup
- User interface design
- Unforeseen issues during testing process

These were mitigated by:

- Frequent communication with teammates and client
- Working together on tasks and having teammates review each merge request
- Making use of online tutorials and other free resources

System Design

Functional Decomposition


Golf League Scheduler
Build a Schedule
Handle Absent Players
Keep Track of Team Scores
Create A Member
Create A Team
Create An Outing
Create A Hole Assignment
Confirm Players are Present
In-app
On Paper
Merge Teams with Missing Players
Delete Extra Hole Assignments
Save Scores for an Outing
Update Team Running Total
Update any substituted Team's running total

Demo

Technologies Used

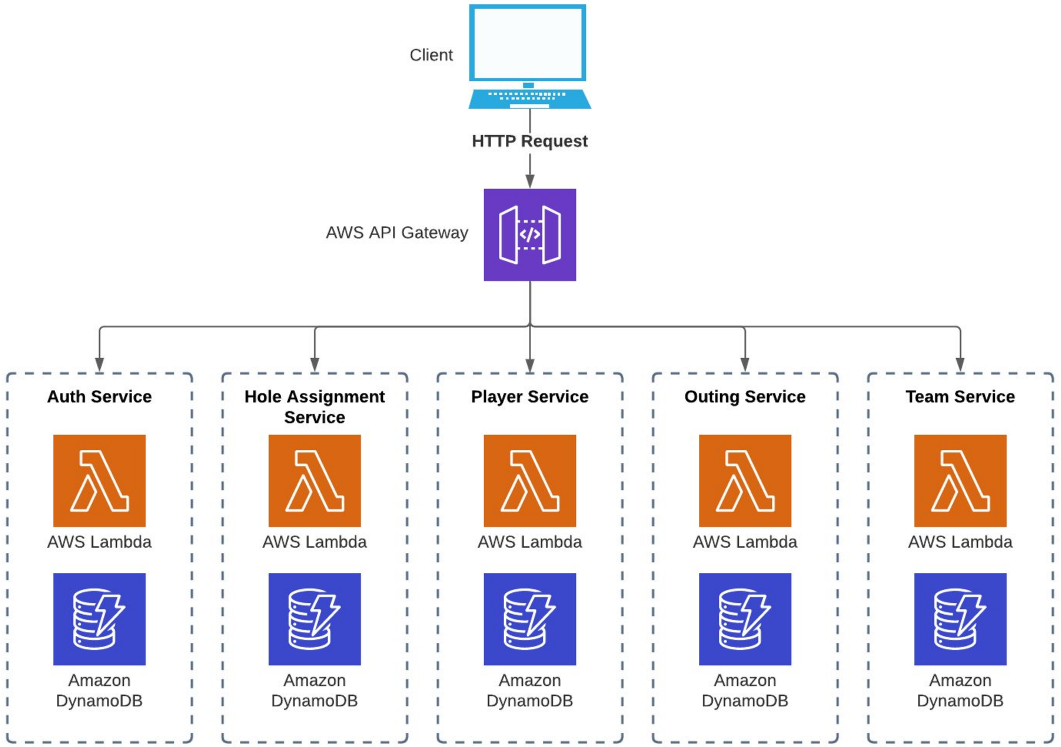
Production

- Utilize AWS free tier for backend
- Node.js chosen for its cold starts which will be the majority of interactions
- DynamoDB chosen for compatibility with AWS free tier and is ideal for serverless architecture
- Netlify chosen for simplicity of deployment and domain configuration
- Single Page Application written in React

Local Dev

- Docker
- DynamoDB-local
- Serverless framework

Cloud Architecture



Scheduling Algorithm Requirements

Initial Schedule Generation

- Teams should be paired if possible

Schedule Shuffling when check-in is complete

- Keep outings short
  - Fewer Hole Assignments
  - Teams start in line:
    - Good starting holes: 1/2/3/4
    - Bad starting holes: 1/3/4/7
- Scores must propagate to each player's original team for the season cumulative score

Testing

- **Integration Tests**
  - Full integration test suite using Postman to ensure HTTP handlers are functioning correctly.
- **Unit Testing**
  - 89% test coverage of all business logic and data access.
  - DynamoDB local test environments created and re-seeded before every test to ensure database queries are correct
- **Manual UI testing**
  - All Merge Requests are QA'd by a second member of the team.
  - If a bug was found during the QA process, the author of the Merge Request was required to make the necessary changes before requesting another round of testing.

Process Evolution / Lessons Learned

Process Evolution

- Continuous Integration Issues
    - Decided to abandon CI, instead testing locally for each Merge Request
- Local development setup difficulties
- Automated Interface Testing

Lessons Learned

- A good development environment setup can save hours of work
- Testing in a Serverless environment can be difficult
- Proper planning makes development much simpler

Questions?

Scheduling Shuffling Algorithm Implementation

```
fillMissingSlots():
    // empty lists that will contain hole assignments, corresponding to
    // what should be done with that hole assignment

    // teams and players that need a new team/pairing

    i ← 0
    j ← length of allHoleAssignments
    while (j < i):
        // store players and teams that need to be moved
        orphanedPlayers, orphanedTeams

        // find next hole assignment with absent players
        // - must have at least one absent player
        i ← index of next hole needing players

        // find next hole assignment to pull players from
        // - must have at least one absent player
        j ← index of next hole with players than can be moved

        // make sure that i and j have not crossed after the previous loops
        if (j > i):
            // the i'th hole assignment will be retained
            // the j'th hole assignment will be deleted

        // grab orphaned team from allHoleAssignments[j] if possible
        orphanedTeams ← any teams with both players confirmed

        // grab orphaned players from allHoleAssignments[j] if possible
        orphanedPlayers ← any players without a confirmed partner
    // end while loop
    ...
```

```
...

// loop through hole assignments that will be retained,
//     but need slots filled
foreach (hole assignment in toUpdate):
    // replace a team if one is missing
    // replace any missing players on either team

while (there are still orphaned players or teams):
    // create new hole assignment object

    // place any remaining orphaned teams on the new hole assignments

    // place any remaining orphaned players on new teams

    // update toCreate to later save the new hole assignment

// complete database operations for the toUpdate, toDelete, and toCreate lists
```