High Level Design Document for Online Scrabble-Game

1. Key Components and Their Interactions

1.1 Backend (Node.js, Express, MongoDB, Socket.io)

- **API Server**: Manages user authentication, game sessions, moves, and leaderboards.
- **Database (MongoDB)**: Stores players, games, leaderboards, and dictionaries.
- **WebSocket Server (Socket.io)**: Handles real-time updates for game actions, player moves, and board state synchronization.

1.2 Frontend (React, Redux)

- **Game Interface**: Displays the board, player tiles, and notifications.
- **Leaderboard**: Shows rankings based on scores and games won.
- **Admin Dashboard**: Allows dictionary and game board management.

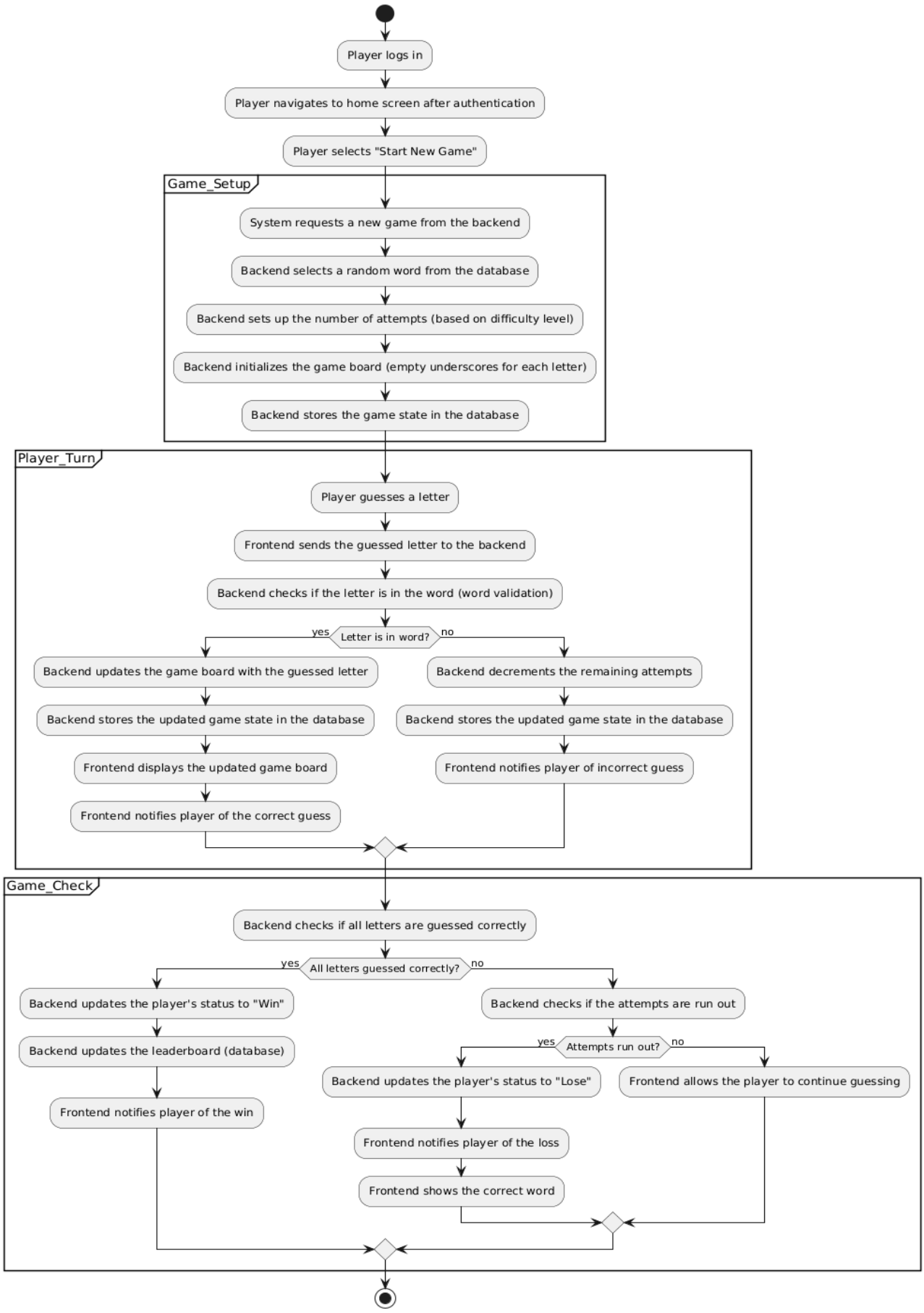
1.3 Real-time Communication

- **WebSocket**: Ensures live game updates (tile placements, turn changes, chat).
- **REST API**: Handles game creation, game history retrieval, and leaderboard updates.

1.4 Interaction Flow

- Players create/join a game -> WebSocket updates for all participants.
- Players submit moves -> Backend validates word and updates scores.
- Game ends -> Leaderboard is updated in the database.

2. Data Flow Diagram (DFD)


Player logs in
Player navigates to home screen after authentication
Player selects "Start New Game"
Game_Setup
System requests a new game from the backend
Backend selects a random word from the database
Backend sets up the number of attempts (based on difficulty level)
Backend initializes the game board (empty underscores for each letter)
Backend stores the game state in the database
Player_Turn
Player guesses a letter
Frontend sends the guessed letter to the backend
Backend checks if the letter is in the word (word validation)
yes
Letter is in word?
no
Backend updates the game board with the guessed letter
Backend stores the updated game state in the database
Frontend displays the updated game board
Frontend notifies player of the correct guess
Backend decrements the remaining attempts
Backend stores the updated game state in the database
Frontend notifies player of incorrect guess
Game_Check
Backend checks if all letters are guessed correctly
yes
All letters guessed correctly?
no
Backend updates the player's status to "Win"
Backend updates the leaderboard (database)
Frontend notifies player of the win
Backend checks if the attempts are run out
yes
Attempts run out?
no
Backend updates the player's status to "Lose"
Frontend notifies player of the loss
Frontend shows the correct word
Frontend allows the player to continue guessing

3. Request / Response Examples

3.1 Create Game

Request (POST /games/create)

```
{
  "username": "player1",
  "maxPlayers": 4
}
```

Response
json
```
{
  "gameId": "abcd1234",
  "status": "Game created",
  "players": ["player1"]
}
```

3.2 Submit Move

Request (POST /games/:id/move)

json

```
{
  "gameId": "abcd1234",
  "playerId": "player1",
  "word": "HELLO",
  "position": { "row": 7, "col": 7, "direction": "horizontal" }
}
```

Response

json
```
{
  "valid": true,
  "score": 10,
  "nextTurn": "player2"
}
```

3.3 Fetch Leaderboard

Request (GET /leaderboard) **Response**

json

```
[
  { "player": "player1", "score": 120 },
  { "player": "player2", "score": 110 }
]
```

4. Non-functional Requirements

- **Scalability**: The system should handle 1,000+ concurrent players.
- **Availability**: 99.9% uptime with fault-tolerant WebSocket connections.
- **Usability**: Intuitive UI with mobile responsiveness.
- **Extensibility**: Easy addition of new game types and boards

5. Latency Considerations

- **Real-time updates**: Max latency of 200ms for WebSocket interactions.
- **API Requests**: Response times under 500ms for game actions.
- **Database Queries**: Indexed collections to reduce query time.

6. Security

- **Authentication**: Secure login using **JWT tokens**.
- **Data Protection**: Use **HTTPS** for encrypted communication.
- **Input Validation**: Sanitize inputs to prevent **SQL Injection** and **XSS** attacks.
- **Rate Limiting**: Prevent abuse of APIs using request throttling.
- **Access Control**: Role-based access for Admin features.

7. Technology Stack / Choices

- **Frontend**: React, Tailwind CSS, Context API / Redux
- **Backend**: Node.js, Express, Socket.io
- **Database**: MongoDB
- **Authentication**: JWT (JSON Web Tokens), bcrypt for password encryption
- **Real-time Communication**: WebSocket (Socket.io)
- **DevOps**: Docker, GitHub Actions, AWS EC2, Vercel for frontend hosting
- **Analytics**: Google Analytics, Mixpanel
- **Offline Support**: Service Workers, IndexedDB

8. Cost of Goods Sold (COGS)

8.1 Initial Costs

- Domain & Hosting: ~$50/month
- Cloud Database (MongoDB Atlas): ~$60/month (based on usage)
- WebSocket Server (Socket.io): ~$40/month
- SSL Certificate: ~$10/month

8.2 Development & Maintenance Costs

- Developer Salaries: ~$3,000/month per developer
- Tools: GitHub, Docker, CI/CD (~$20/month)
- Analytics & Monitoring: ~$30/month for services like Google Analytics or Mixpanel

Estimated Monthly Cost (excluding salaries): ~$200 - $300

9. Conclusion

This design document outlines the architecture for building a scalable and interactive online multiplayer Scrabble-like game using the MERN stack. It includes decisions on technology, security measures, and projected costs to ensure a smooth, engaging player experience.