Lab Task

Task 1: Wizard Duel Arena API

You've been hired to create an API for a fantasy game where players can duel with their wizards. The API will determine how duels are handled based on the wizard's experience, health status, and current magic energy. There's only one route for dueling, but middleware handles the complexity.

Routes:

- POST /duel: Simulates a duel between two wizards, adjusting based on type, health, and energy.
- PUT /wizards/upgrade: Upgrade a wizard's stats (attack or defense).
- GET /wizards: Get details of a specific wizard (stats, type, and health).

Middleware:

- **Wizard Type Check:**
  - Determine the type of wizard (fire, ice, or storm) based on the request body and apply different strategies to the duel.
  - Fire wizards get attack boosts, Ice wizards get defense boosts, and Storm wizards get speed boosts.

- **Health Check:**
  Evaluate the health level of the wizard. If the health is below 20%, reduce the effectiveness of their abilities by 30%.

- **Magic Energy Level:**
  Based on the wizard's magic energy (e.g., mana), determine if they can use high-level spells. If energy is below a threshold, only basic spells are available.

- **Duel Outcome Logic:**
  Finally, middleware dynamically determines the duel's outcome based on all the previous middleware steps (type, health, energy).

- **ErrorHandler**:
  Catches and logs errors, returning a 500 status code with an error message.

Sample request and response JSON

Sample Request for /duel

```
{
  "wizard1": {
    "name": "FireMage",
    "type": "fire",
    "health": 80,
    "energy": 100
  },
```

```
  "wizard2": {
    "name": "IceSorceress",
    "type": "ice",
    "health": 60,
    "energy": 80
  }
}
```

Sample Response for /duel


```
{
  "message": "Duel processed",
  "outcome": "FireMage wins!"
}
```

Sample Request for /wizards/upgrade


```
{
  "id": 1,
  "strength": 5,
  "agility": 0,
  "wisdom": 3
}
```

Sample Response for /wizards/upgrade


```
{
  "message": "Player stats upgraded",
  "player": {
    "id": 1,
    "name": "Hero",
    "strength": 85,
    "agility": 60,
    "wisdom": 73,
    "experience": 200,
    "resources": { "gold": 500, "potions": 3 },
    "questStatus": "pending"
  }
}
```

Sample Request for /players


```
{}
```

Sample Response for /players


```
[
  {
    "id": 1,
    "name": "Hero",
    "strength": 80,
    "agility": 60,
    "wisdom": 70,
    "experience": 200,
    "resources": { "gold": 500, "potions": 3 },
```

```
    "questStatus": "pending"
  },
  {
    "id": 2,
    "name": "Rogue",
    "strength": 60,
    "agility": 90,
    "wisdom": 50,
    "experience": 150,
    "resources": { "gold": 300, "potions": 1 },
    "questStatus": "pending"
  },
  {
    "id": 3,
    "name": "Mage",
    "strength": 50,
    "agility": 40,
    "wisdom": 100,
    "experience": 250,
    "resources": { "gold": 700, "potions": 5 },
    "questStatus": "pending"
  }
]
```


Task 2: Bank Loan Approval API

In this financial application, clients can apply for loans. The outcome of the loan approval process is determined by multiple middleware checks based on credit score, current debt, and account balance.

Routes:

- POST /apply-loan: Handles a loan application, checking financial details before approval.
- GET /customers/loan-status: Check the status of a customer's loan (approved, denied, pending).
- PUT /customers/update-info: Update a customer's financial information (balance, debt, income).

Middleware:

- **Credit Score Check:**
  Evaluate the credit score. If it's below 600, the loan application is denied upfront.

- **Debt-to-Income Ratio Check:**
  Check the client's debt-to-income ratio. If it's higher than 40%, the approval likelihood decreases.

- **Account Balance Check:**
  If the account balance is below a threshold (e.g., $1000), further restrict the loan approval terms.

- **Final Loan Approval Logic:**
  Based on the results of the previous checks, dynamically approve or reject the loan, or offer a reduced loan amount.

- **ErrorHandler**:
  Handles any errors that occur during request processing.

Sample request and response JSON

Sample Request for /apply-loan

```
{
   "customerId": 1,
   "creditScore": 650,
   "currentDebt": 1500,
   "accountBalance": 2000
}
```

Sample Response for /apply-loan

```
{
   "message": "Loan application processed",
   "status": "approved",
   "loanAmount": 5000
}
```

Sample Request for /customers/loan-status

```
{
   "customerId": 1
}
```

Sample Response for /customers/loan-status

```
{
   "customerId": 1,
   "loanStatus": "approved"
}
```

Sample Request for /customers/update-info

```
{
   "customerId": 1,
   "balance": 2500,
   "debt": 1200,
   "income": 5000
}
```

Sample Response for /customers/update-info

```
{
   "message": "Customer financial information updated",
   "customer": {
      "customerId": 1,
      "balance": 2500,
```

```
    "debt": 1200,
    "income": 5000
  }
}
```

Task 3: Adventure Quest API

This API handles quests in an adventure game. Based on the player's skills, experience, and available resources, quests are completed differently, with middleware altering the outcomes dynamically.

Routes:

- POST /complete-quest: Completes a quest based on player skills, experience, and resources.
- PUT /players/upgrade-stats: Upgrade player stats like strength, agility, or wisdom.
- GET /players: Get player details (current stats, experience, resources).

Middleware:

- **Skill Level Check:**
  Evaluate the player's skill level. Higher skill levels provide bonuses in quest outcomes.

- **Experience Level Check:**
  Based on experience, players might get additional quest rewards.

- **Resource Availability Check:**
  If the player lacks the necessary resources for the quest, the quest will fail.

- **Final Quest Outcome:**
  The final quest outcome (success or failure) is determined based on the previous middleware checks.
- **ErrorHandler**:
  Handles errors that occur during the request lifecycle.

Sample request and response JSON

Sample Request for /complete-quest

```
{
  "playerId": 1,
  "questDifficulty": "moderate"
}
```

Sample Response for /complete-quest

```
{
  "message": "Quest completion processed",
  "outcome": "success"
}
```

Sample Request for /players/upgrade-stats

```
{
  "id": 1,
  "strength": 5,
  "agility": 2,
  "wisdom": 0
}
```

Sample Response for /players/upgrade-stats


```
{
  "message": "Player stats upgraded",
  "player": {
    "id": 1,
    "name": "Hero",
    "strength": 85,
    "agility": 62,
    "wisdom": 70,
    "experience": 200,
    "resources": { "gold": 500, "potions": 3 }
  }
}
```

Sample Request for /players


```
{}
```

Sample Response for /players


```
[
  {
    "id": 1,
    "name": "Hero",
    "strength": 80,
    "agility": 60,
    "wisdom": 70,
    "experience": 200,
    "resources": { "gold": 500, "potions": 3 }
  },
  {
    "id": 2,
    "name": "Rogue",
    "strength": 60,
    "agility": 90,
    "wisdom": 50,
    "experience": 150,
    "resources": { "gold": 300, "potions": 1 }
  },
  {
    "id": 3,
    "name": "Mage",
    "strength": 50,
    "agility": 40,
```

```
    "wisdom": 100,
    "experience": 250,
    "resources": { "gold": 700, "potions": 5 }
  }
]
```

Task 4: Wildlife Rescue Mission API

You've been tasked with building an API for a wildlife rescue organization that manages animal rescue missions. The organization classifies rescue missions based on the type of animal, severity of the situation, and available resources. The API has a single route to handle all rescue missions, but middleware processes different factors to determine how the mission will be handled.

Route:

- POST /rescue-mission: Handles animal rescue missions. Middleware assesses the animal type, the severity of the situation, and available resources to adjust the mission outcome dynamically.

Middleware:

- **Animal Type Check:**
  Determine the type of animal being rescued (e.g., bird, mammal, and reptile). Depending on the animal type, different rescue strategies or precautions will be applied.

- **Severity Level Check:**
  Evaluate the severity of the situation (e.g., mild, moderate, severe). In severe situations, more resources will be required, and mission difficulty increases.

- **Resource Availability Check:**
  Check whether there are enough rescue team members, vehicles, and equipment available. If resources are insufficient, the rescue may be delayed or require extra time.

- **Mission Outcome Determination:**
  Based on the animal type, severity, and resources, the middleware will dynamically determine if the rescue mission is successful, delayed, or unsuccessful due to lack of resources.

- **ErrorHandler**: Captures and responds to errors during the request lifecycle.

Sample request and response JSON

Sample Request for /rescue-mission

```
{
  "animalType": "mammal",
  "severity": "moderate"
}
```

Sample Response for /rescue-mission

```
{
   "message": "Rescue mission processed",
   "outcome": "success"
}
```