Processing Game: API Analysis and Explanation

Introduction

This API manages a leaderboard for a game. It supports features like:

- Submitting and updating scores.
- Maintaining a top-5 leaderboard for each difficulty level.
- Ensuring secure storage and validation of user data.
- Retrieving scores for display in the game.

---

1. Middleware Setup

dotenv

- Loads environment variables from a .env file into process.env.
- Example: process.env.MONGODB_URI and process.env.SALT_ROUNDS.

helmet

- Adds security headers to prevent common vulnerabilities like cross-site scripting (XSS) and clickjacking.

cors

- Enables Cross-Origin Resource Sharing to allow requests from specified origins.

Example:

```
const allowedOrigins = process.env.ALLOWED_ORIGINS.split(',');
app.use((req, res, next) => {
  const origin = req.headers.origin;
  if (allowedOrigins.includes(origin)) {
    res.setHeader('Access-Control-Allow-Origin', origin);
  }
  res.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS');
  res.setHeader('Access-Control-Allow-Headers', 'Content-Type');
  res.setHeader('Access-Control-Allow-Credentials', true);
  next();
});
```

mongoSanitize

- Prevents NoSQL injection by stripping prohibited characters from query inputs.

Rate Limiting

- Limits repeated requests to /api/scores to prevent abuse.

Example:

```
// Rate limiting
const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100 // limit each IP to 100 requests per window
});
```

---

2. MongoDB Connection

- Uses mongoose to connect to a MongoDB database.

---

3. Data Validation

Joi Schema

- Validates incoming data for the POST /api/scores endpoint.

Mongoose Schema

- Defines the database structure for a score.

---

4. API Endpoints

POST /api/scores

Handles adding and updating scores while ensuring security and compliance with the rules.

1. **Validation:**
    - Incoming requests are validated against the Joi schema.

    Example:

```
// Validate the incoming data
const { error } = scoreSchema.validate(req.body);
if (error) {
  return res.status(400).json({
    status: 400,
    message: "Validation Error",
    details: error.details[0].message,
  });
}
```

2. **Score Update Logic:**
    - Checks if the username exists:
        - **If it exists:**
            - Compares the hashed PIN with the stored PIN using bcrypt.
            - Updates the score only if the new score is higher.
        - **If it doesn't exist:**
            - Hashes the PIN and saves the new score.

      Example:

```
if (existingScore) {
  // Verify PIN
  const isMatch = await bcrypt.compare(pin, existingScore.pin);
  if (!isMatch) {
    return res.status(401).json({
      status: 401,
      message: "Invalid PIN",
      details: "The provided PIN does not match our records for this username.",
    });
  }

  // Only update the score if it's higher than the existing score
  if (score > existingScore.score) {
    existingScore.score = score;
    existingScore.difficulty_level = difficulty_level;
    await existingScore.save();
    await maintainTopFive(difficulty_level);
    return res.status(200).json({ status: 200, message: "Score updated successfully." });
  }
```

3. **Top-5 Maintenance:**
    - The maintainTopFive function ensures only the top 5 scores for a difficulty level are kept in the database.

      Example:

```
// Function to maintain only top 5 scores
async function maintainTopFive(difficulty) {
  try {
    // Find all scores for the difficulty level, sorted in descending order
    const scores = await Score.find({ difficulty_level: difficulty }).sort({ score: -1 });

    // Check if there are more than 5 scores
    if (scores.length > 5) {
      // Get the IDs of the scores that need to be removed
      const idsToRemove = scores.slice(5).map(score => score._id);

      // Remove the lowest scores (those beyond the top 5)
      await Score.deleteMany({ _id: { $in: idsToRemove } });
    }
```

GET /api/scores/:difficulty

Retrieves the Top-5 scores for a specific difficulty level.

1. **Database Query:**
    - Finds scores for the requested difficulty, sorted in descending order.
    - Limits the results to the top 5.
2. **Error Handling:**
    - Returns appropriate error messages for failed requests.

---

5. Security Features

1. **Preventing Duplicate Submissions:**
    - Checks if a username already exists and ensures that only higher scores are updated.
2. **Rate Limiting and Input Sanitization:**
    - Protects the API from abuse and injection attacks.
3. **Hashing PINs:**
    - Uses `bcrypt` to hash PINs before storing them, ensuring they are not stored as plaintext.

---

6. Sever Setup

Starts the Express server and listens for requests on the specified port:

```
// Start server
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```