Frontend.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Task Submission</title>
</head>
<body>
    <h1>User Task Submission</h1>
    <form id="taskForm">
        <label for="userId">Enter User ID:</label>
        <input type="text" id="userId" name="userId" required>
        <button type="submit">Submit Task</button>
    </form>
    <div id="status"></div>

    <script>
        document.getElementById("taskForm").addEventListener("submit", function(event) {
            event.preventDefault();
            const userId = document.getElementById("userId").value;

            fetch('http://localhost:3000/submit-task', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                },
                body: JSON.stringify({ user_id: userId }),
            })
            .then(response => response.json())
            .then(data => {
```

```
            document.getElementById("status").innerText = data.message;
        })
        .catch(error => {
            document.getElementById("status").innerText = 'Error: ' + error;
        });
    });
  </script>
</body>
</html>
```

Backend.html

```
const express = require('express');
const rateLimit = require('express-rate-limit');
const Redis = require('ioredis');
const fs = require('fs');
const cluster = require('cluster');
const os = require('os');

const app = express();
const redis = new Redis();
const port = 3000;

app.use(express.json());

// Task function provided in the PDF to log task completion
async function task(user_id) {
  const log = ${user_id}-task completed at-${Date.now()}\n;
  fs.appendFileSync('task_log.txt', log, 'utf8');
  console.log(log);
}
```

```
// Middleware for rate limiting - limiting each user to 1 task per second and 20 tasks per minute
const taskLimiter = rateLimit({
    windowMs: 60 * 1000, // 1 minute window
    max: 20, // Limit each user ID to 20 requests per window per minute
    keyGenerator: (req) => req.body.user_id, // Apply rate limit based on user_id
    handler: (req, res) => {
        res.status(429).json({ message: 'Rate limit exceeded, task queued' });
    }
});

// Task queue function using Redis
const taskQueue = async (user_id) => {
    await redis.lpush(task_queue:${user_id}, JSON.stringify({ user_id }));
    processQueue(user_id);
};

// Function to process queued tasks - ensures tasks are processed at a rate of 1 task per second
const processQueue = async (user_id) => {
    const taskInQueue = await redis.rpop(task_queue:${user_id});
    if (taskInQueue) {
        const parsedTask = JSON.parse(taskInQueue);
        await task(parsedTask.user_id);
        setTimeout(() => processQueue(parsedTask.user_id), 1000); // 1 task per second per user
    }
};

// Route to submit tasks - applies rate limiting and queues tasks if limit is exceeded
app.post('/submit-task', taskLimiter, (req, res) => {
    const { user_id } = req.body;
    if (!user_id) {
        return res.status(400).json({ message: 'User ID is required' });
```

```
    }
    taskQueue(user_id);
    res.json({ message: 'Task queued' });
});

// Clustering setup to create two replicas
if (cluster.isMaster) {
    const numCPUs = Math.min(2, os.cpus().length); // Create two replicas
    for (let i = 0; i < numCPUs; i++) {
        cluster.fork();
    }

    cluster.on('exit', (worker, code, signal) => {
        console.log(Worker ${worker.process.pid} died);
        cluster.fork(); // Restart worker if it dies
    });
} else {
    app.listen(port, () => {
        console.log(Worker ${process.pid} running on port ${port});
    });
}
```