# Omnitrix Timer - Backend API Specification

## Overview

This document outlines the backend requirements for the Omnitrix Timer application. The backend must maintain a persistent timer state that syncs across all devices and continues running even when no clients are connected.

---

## Technology Stack Recommendations

### Option 1: Node.js + Express + MongoDB (Recommended)

- **Backend Framework**: Express.js
- **Database**: MongoDB (for storing timer state)
- **Deployment**: Heroku, Railway, or DigitalOcean
- **Estimated Setup Time**: 2-3 hours

### Option 2: Firebase (Easiest)

- **Backend**: Firebase Realtime Database or Firestore
- **Auth** (optional): Firebase Authentication
- **Deployment**: Automatic with Firebase
- **Estimated Setup Time**: 1-2 hours

### Option 3: Supabase (Modern Alternative)

- **Backend**: Supabase (PostgreSQL + REST API)
- **Real-time**: Built-in real-time subscriptions
- **Deployment**: Managed by Supabase
- **Estimated Setup Time**: 1-2 hours

---

## Database Schema

### Timer State Document/Table

```javascript
```

```
{
  _id: "unique_timer_id", // Usually just one timer with ID "main"
  remainingTime: 86400, // Seconds remaining (starts at 24*60*60 = 86400)
  isRunning: false, // Boolean - is timer currently running
  startedAt: null, // Timestamp when timer was started (ISO 8601)
  lastUpdated: "2025-01-05T10:30:00Z", // Last update timestamp
  totalDuration: 86400 // Total duration in seconds (24 hours)
}
```

---

## API Endpoints

### Base URL

```
http://your-backend-url.com/api
```

### 1. GET /timer

**Description**: Fetch current timer state

**Response**:

```json
{
  "success": true,
  "data": {
    "remainingTime": 82345,
    "isRunning": true,
    "startedAt": "2025-01-05T10:30:00Z",
    "lastUpdated": "2025-01-05T11:45:23Z"
  }
}
```

**Logic**:

- If `isRunning` is true, calculate the actual remaining time:

```javascript
const elapsedSeconds = Math.floor((Date.now() - new Date(startedAt)) / 1000);
const actualRemainingTime = Math.max(0, remainingTime - elapsedSeconds);
```

- Return the calculated remaining time

---

## 2. PUT /timer

**Description**: Update timer state manually

**Request Body**:

```
json

{
  "remainingTime": 86400,
  "isRunning": false
}
```

**Response**:

```
json

{
  "success": true,
  "message": "Timer state updated",
  "data": {
    "remainingTime": 86400,
    "isRunning": false,
    "lastUpdated": "2025-01-05T12:00:00Z"
  }
}
```

---

## 3. POST /timer/start

**Description**: Start the timer

**Request Body**: None

**Response**:

```
json

{
  "success": true,
  "message": "Timer started",
  "data": {
    "remainingTime": 86400,
    "isRunning": true,
    "startedAt": "2025-01-05T12:00:00Z"
  }
}
```

**Logic**:

1. Set `isRunning` to `true`
2. Set `startedAt` to current timestamp
3. Save to database
4. Return updated state

---

## 4. POST /timer/stop

**Description**: Pause/stop the timer

**Request Body**: None

**Response**:

```json
{
  "success": true,
  "message": "Timer stopped",
  "data": {
    "remainingTime": 82345,
    "isRunning": false,
    "startedAt": null
  }
}
```

**Logic**:

1. Calculate elapsed time since `startedAt`
2. Update `remainingTime` = `remainingTime - elapsedSeconds`
3. Set `isRunning` to `false`
4. Set `startedAt` to `null`
5. Save to database
6. Return updated state

---

## 5. POST /timer/reset

**Description**: Reset timer to 24 hours

**Request Body**: None

**Response**:

```json
{
  "success": true,
  "message": "Timer reset",
  "data": {
    "remainingTime": 86400,
    "isRunning": false,
    "startedAt": null
  }
}
```

**Logic**:

1. Set `remainingTime` to 86400 (24 hours)
2. Set `isRunning` to `false`
3. Set `startedAt` to `null`
4. Save to database

---

## Important Backend Logic

### Timer Calculation on Every Request

Every time a GET request is made, the backend should:

```javascript
```

```javascript
function getActualRemainingTime(timerState) {
  if (!timerState.isRunning) {
    return timerState.remainingTime;
  }

  const elapsedMs = Date.now() - new Date(timerState.startedAt).getTime();
  const elapsedSeconds = Math.floor(elapsedMs / 1000);
  const actualRemaining = Math.max(0, timerState.remainingTime - elapsedSeconds);

  // If timer reached 0, stop it
  if (actualRemaining === 0) {
    timerState.isRunning = false;
    timerState.remainingTime = 0;
    // Save to database
  }

  return actualRemaining;
}
```

### Background Timer Check (Optional but Recommended)

Set up a cron job or interval that runs every minute to:

1. Check if timer is running
2. Calculate remaining time
3. If remaining time is 0, set `isRunning` to false
4. Update database

---

## CORS Configuration

**IMPORTANT**: Enable CORS for your frontend domain

```javascript
// Express.js example
const cors = require('cors');

app.use(cors({
  origin: ['http://localhost:3000', 'https://your-frontend-domain.com'],
  methods: ['GET', 'POST', 'PUT', 'DELETE'],
  credentials: true
}));
```

---

## Error Handling

All endpoints should return proper error responses:

```json
{
  "success": false,
  "error": "Error message here",
  "code": "ERROR_CODE"
}
```

**Common HTTP Status Codes**:

- 200: Success
- 400: Bad Request
- 404: Not Found
- 500: Internal Server Error

---

## Sample Implementation (Node.js + Express + MongoDB)

### Installation

```bash
npm install express mongoose cors dotenv
```

### Basic Server Structure (server.js)

```javascript
```

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
require('dotenv').config();

const app = express();

// Middleware
app.use(cors());
app.use(express.json());

// MongoDB Connection
mongoose.connect(process.env.MONGODB_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true
});

// Timer Schema
const timerSchema = new mongoose.Schema({
  _id: { type: String, default: 'main' },
  remainingTime: { type: Number, default: 86400 },
  isRunning: { type: Boolean, default: false },
  startedAt: { type: Date, default: null },
  lastUpdated: { type: Date, default: Date.now },
  totalDuration: { type: Number, default: 86400 }
});

const Timer = mongoose.model('Timer', timerSchema);

// Helper function to get actual remaining time
const getActualRemainingTime = (timer) => {
  if (!timer.isRunning) {
    return timer.remainingTime;
  }

  const elapsedMs = Date.now() - new Date(timer.startedAt).getTime();
  const elapsedSeconds = Math.floor(elapsedMs / 1000);
  return Math.max(0, timer.remainingTime - elapsedSeconds);
};

// GET /api/timer
app.get('/api/timer', async (req, res) => {
  try {
    let timer = await Timer.findById('main');

    if (!timer) {
```

```
      timer = new Timer({ _id: 'main' });
      await timer.save();
    }

    const actualRemainingTime = getActualRemainingTime(timer);

    // If timer reached 0, stop it
    if (actualRemainingTime === 0 && timer.isRunning) {
      timer.isRunning = false;
      timer.remainingTime = 0;
      timer.startedAt = null;
      await timer.save();
    }

    res.json({
      success: true,
      data: {
        remainingTime: actualRemainingTime,
        isRunning: timer.isRunning,
        startedAt: timer.startedAt,
        lastUpdated: timer.lastUpdated
      }
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      error: error.message
    });
  }
});

// PUT /api/timer
app.put('/api/timer', async (req, res) => {
  try {
    const { remainingTime, isRunning } = req.body;

    let timer = await Timer.findById('main');
    if (!timer) {
      timer = new Timer({ _id: 'main' });
    }

    if (remainingTime !== undefined) {
      timer.remainingTime = remainingTime;
    }
    if (isRunning !== undefined) {
      timer.isRunning = isRunning;
      if (!isRunning) {
```

```
      timer.startedAt = null;
    }
  }

  timer.lastUpdated = new Date();
  await timer.save();

  res.json({
    success: true,
    message: 'Timer state updated',
    data: {
      remainingTime: timer.remainingTime,
      isRunning: timer.isRunning,
      lastUpdated: timer.lastUpdated
    }
  });
 } catch (error) {
  res.status(500).json({
    success: false,
    error: error.message
  });
 }
});

// POST /api/timer/start
app.post('/api/timer/start', async (req, res) => {
 try {
  let timer = await Timer.findById('main');
  if (!timer) {
    timer = new Timer({ _id: 'main' });
  }

  // Get actual remaining time before starting
  const actualRemainingTime = getActualRemainingTime(timer);

  timer.remainingTime = actualRemainingTime;
  timer.isRunning = true;
  timer.startedAt = new Date();
  timer.lastUpdated = new Date();

  await timer.save();

  res.json({
    success: true,
    message: 'Timer started',
    data: {
      remainingTime: timer.remainingTime,
```

```
      isRunning: timer.isRunning,
      startedAt: timer.startedAt
    }
  });
 } catch (error) {
  res.status(500).json({
    success: false,
    error: error.message
  });
 }
});

// POST /api/timer/stop
app.post('/api/timer/stop', async (req, res) => {
 try {
  let timer = await Timer.findById('main');
  if (!timer) {
    return res.status(404).json({
      success: false,
      error: 'Timer not found'
    });
  }

  // Calculate actual remaining time
  const actualRemainingTime = getActualRemainingTime(timer);

  timer.remainingTime = actualRemainingTime;
  timer.isRunning = false;
  timer.startedAt = null;
  timer.lastUpdated = new Date();

  await timer.save();

  res.json({
    success: true,
    message: 'Timer stopped',
    data: {
      remainingTime: timer.remainingTime,
      isRunning: timer.isRunning,
      startedAt: timer.startedAt
    }
  });
 } catch (error) {
  res.status(500).json({
    success: false,
    error: error.message
  });
```

```
  }
});

// POST /api/timer/reset
app.post('/api/timer/reset', async (req, res) => {
  try {
    let timer = await Timer.findById('main');
    if (!timer) {
      timer = new Timer({ _id: 'main' });
    }

    timer.remainingTime = 86400; // 24 hours
    timer.isRunning = false;
    timer.startedAt = null;
    timer.lastUpdated = new Date();

    await timer.save();

    res.json({
      success: true,
      message: 'Timer reset',
      data: {
        remainingTime: timer.remainingTime,
        isRunning: timer.isRunning,
        startedAt: timer.startedAt
      }
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      error: error.message
    });
  }
});

// Server start
const PORT = process.env.PORT || 3001;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

## Environment Variables (.env)

```
MONGODB_URI=mongodb+srv://username:password@cluster.mongodb.net/omnitrix-timer
PORT=3001
```

## Deployment Steps

### 1. MongoDB Setup

- Create free account at MongoDB Atlas
- Create a new cluster
- Get connection string
- Add to `.env` file

### 2. Deploy Backend

**Option A - Railway**:

1. Push code to GitHub
2. Connect Railway to GitHub repo
3. Add environment variables
4. Deploy automatically

**Option B - Heroku**:

1. Install Heroku CLI
2. `heroku create your-app-name`
3. `git push heroku main`
4. Add environment variables in Heroku dashboard

**Option C - DigitalOcean**:

1. Create droplet
2. SSH into server
3. Install Node.js and MongoDB
4. Clone repo and run with PM2

### 3. Update Frontend

Replace `API_BASE_URL` in frontend code with your deployed backend URL:

```javascript
const API_BASE_URL = 'https://your-backend-url.com/api';
```

---

## Testing the API

Use Postman or curl to test:

```bash
# Get timer state
curl http://localhost:3001/api/timer

# Start timer
curl -X POST http://localhost:3001/api/timer/start

# Stop timer
curl -X POST http://localhost:3001/api/timer/stop

# Reset timer
curl -X POST http://localhost:3001/api/timer/reset
```

---

## Security Considerations

1. **Rate Limiting**: Add rate limiting to prevent abuse

```javascript
const rateLimit = require('express-rate-limit');

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

app.use('/api/', limiter);
```

2. **Authentication** (Optional): Add API key or JWT authentication if needed
3. **Input Validation**: Validate all input data

---

## Asset Upload Requirements

The frontend requires these assets. Host them on a CDN or include in your project:

1. **Alien Images** (5 images):
   - alien1.jpg
   - alien2.jpg
   - alien3.jpg

- alien4.jpg
- alien5.jpg

2. **Omnitrix Icon**:
   - omnitrix.png (128x128 recommended)
3. **Activation Sound**:
   - activate.mp3 (short beep/activation sound)

**Hosting Options**:

- Cloudinary (free tier)
- AWS S3
- Your own server
- Replace with placeholder URLs (already done in frontend)

---

## Estimated Costs

### Free Options:

- MongoDB Atlas: Free tier (512 MB storage)
- Railway: Free tier (500 hours/month)
- Heroku: Free tier (discontinued, but alternatives available)

### Paid Options (if scaling):

- MongoDB Atlas: $9/month (Shared cluster)
- Railway: $5/month
- DigitalOcean: $4-6/month (Basic droplet)

---

## Support & Troubleshooting

**Common Issues**:

1. **CORS Errors**: Make sure CORS is properly configured
2. **MongoDB Connection**: Check connection string and IP whitelist
3. **Timer Drift**: Backend calculates on every request, minimal drift
4. **Multiple Devices**: Timer syncs every 2 seconds automatically

---

## Next Steps

1. Set up MongoDB database
2. Deploy backend code
3. Update frontend with backend URL
4. Upload assets or use placeholders
5. Test across multiple devices
6. Monitor and optimize

**Questions?** Contact your backend developer with this document!