Secure Todo List Application - Complete Project Files

Quick Setup Instructions

1. Create a new directory: `mkdir secure-todo-app && cd secure-todo-app`
2. Create the following folder structure and files
3. Run the setup commands provided below
4. Start the application

Project Structure

```
secure-todo-app/
├── server/
│   ├── controllers/
│   ├── middleware/
│   ├── models/
│   ├── routes/
│   ├── utils/
│   ├── uploads/
│   ├── temp/
│   ├── tests/
│   ├── server.js
│   ├── package.json
│   └── .env
├── client/
│   ├── public/
│   ├── src/
│   ├── package.json
│   └── .env
└── README.md
```

SERVER FILES

server/package.json

json

```json
{
  "name": "secure-todo-server",
  "version": "1.0.0",
  "description": "Secure Todo API Server",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js",
    "test": "jest",
    "test:coverage": "jest --coverage"
  },
  "dependencies": {
    "express": "^4.18.2",
    "mongoose": "^7.5.0",
    "bcryptjs": "^2.4.3",
    "jsonwebtoken": "^9.0.2",
    "cors": "^2.8.5",
    "helmet": "^7.0.0",
    "express-rate-limit": "^6.10.0",
    "multer": "^1.4.5-lts.1",
    "csv-writer": "^1.6.0",
    "express-validator": "^7.0.1",
    "sanitize-html": "^2.11.0",
    "dotenv": "^16.3.1"
  },
  "devDependencies": {
    "jest": "^29.6.2",
    "supertest": "^6.3.3",
    "nodemon": "^3.0.1"
  },
  "jest": {
    "testEnvironment": "node",
    "collectCoverageFrom": [
      "**/*.js",
      "!node_modules/**",
      "!coverage/**"
    ]
  }
}
```

server/.env

```env
PORT=5000
MONGODB_URI=mongodb://localhost:27017/secure-todo
JWT_SECRET=your-super-secure-jwt-secret-key-change-this-in-production
NODE_ENV=development
CLIENT_URL=http://localhost:3000
MAX_FILE_SIZE=5242880
ALLOWED_FILE_TYPES=.txt,.pdf
```

server/server.js

```
javascript
```

```javascript
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const helmet = require('helmet');
const rateLimit = require('express-rate-limit');
const path = require('path');
require('dotenv').config();

const authRoutes = require('./routes/auth');
const todoRoutes = require('./routes/todos');
const fileRoutes = require('./routes/files');

const app = express();

// Security middleware
app.use(helmet({
  contentSecurityPolicy: {
    directives: {
      defaultSrc: ["'self'"],
      styleSrc: ["'self'", "'unsafe-inline'"],
      scriptSrc: ["'self'"],
      imgSrc: ["'self'", "data:", "https:"],
    },
  },
}));

// Rate limiting
const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100,
  message: 'Too many requests from this IP, please try again later.',
  standardHeaders: true,
  legacyHeaders: false,
});
app.use(limiter);

// CORS configuration
app.use(cors({
  origin: process.env.CLIENT_URL || 'http://localhost:3000',
  credentials: true,
  optionsSuccessStatus: 200
}));
```

```javascript
// Body parsing middleware
app.use(express.json({ limit: '10mb' }));
app.use(express.urlencoded({ extended: true, limit: '10mb' }));

// Create necessary directories
const fs = require('fs');
const uploadsDir = path.join(__dirname, 'uploads');
const tempDir = path.join(__dirname, 'temp');

if (!fs.existsSync(uploadsDir)) {
  fs.mkdirSync(uploadsDir, { recursive: true });
}
if (!fs.existsSync(tempDir)) {
  fs.mkdirSync(tempDir, { recursive: true });
}

// Database connection
mongoose.connect(process.env.MONGODB_URI || 'mongodb://localhost:27017/secure-todo', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
})
.then(() => console.log('Connected to MongoDB'))
.catch(err => console.error('MongoDB connection error:', err));

// Routes
app.use('/api/auth', authRoutes);
app.use('/api/todos', todoRoutes);
app.use('/api/files', fileRoutes);

// Health check
app.get('/api/health', (req, res) => {
  res.json({ success: true, message: 'Server is running' });
});

// Error handling middleware
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({
    success: false,
    message: 'Something went wrong!',
    ...(process.env.NODE_ENV === 'development' && { error: err.message })
  });
});
```

```
// 404 handler
app.use('*', (req, res) => {
  res.status(404).json({ success: false, message: 'Route not found' });
});

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});

module.exports = app;
```

server/models/User.js

javascript

```
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');

const userSchema = new mongoose.Schema({
  email: {
    type: String,
    required: [true, 'Email is required'],
    unique: true,
    lowercase: true,
    trim: true,
    match: [/^\w+([.-]?\w+)*@\w+([.-]?\w+)*(\.\w{2,3})+$/, 'Please enter a valid email']
  },
  password: {
    type: String,
    required: [true, 'Password is required'],
    minlength: [8, 'Password must be at least 8 characters long']
  },
  isLocked: {
    type: Boolean,
    default: false
  },
  loginAttempts: {
    type: Number,
    default: 0
  },
  lockUntil: Date,
  refreshTokens: [{
    token: String,
    createdAt: { type: Date, default: Date.now }
  }]
}, {
  timestamps: true
});

// Indexes
userSchema.index({ email: 1 });
userSchema.index({ lockUntil: 1 }, { sparse: true });

// Virtual for checking if account is locked
userSchema.virtual('isAccountLocked').get(function() {
  return !!(this.isLocked || (this.lockUntil && this.lockUntil > Date.now()));
});
```

```javascript
// Pre-save hook to hash password
userSchema.pre('save', async function(next) {
  if (!this.isModified('password')) return next();

  try {
    const salt = await bcrypt.genSalt(12);
    this.password = await bcrypt.hash(this.password, salt);
    next();
  } catch (error) {
    next(error);
  }
});

// Method to compare password
userSchema.methods.comparePassword = async function(candidatePassword) {
  if (this.isAccountLocked) {
    return false;
  }

  try {
    const isMatch = await bcrypt.compare(candidatePassword, this.password);

    if (isMatch) {
      if (this.loginAttempts > 0) {
        this.loginAttempts = 0;
        this.isLocked = false;
        this.lockUntil = undefined;
        await this.save();
      }
      return true;
    } else {
      this.loginAttempts += 1;

      if (this.loginAttempts >= 5) {
        this.isLocked = true;
        this.lockUntil = Date.now() + 30 * 60 * 1000; // 30 minutes
      }

      await this.save();
      return false;
    }
  } catch (error) {
    throw error;
  }
};

module.exports = mongoose.model('User', userSchema);
```

server/models/Todo.js

javascript

```javascript
const mongoose = require('mongoose');
const sanitizeHtml = require('sanitize-html');

const todoSchema = new mongoose.Schema({
  title: {
    type: String,
    required: [true, 'Title is required'],
    trim: true,
    maxlength: [100, 'Title cannot exceed 100 characters'],
    set: function(value) {
      return sanitizeHtml(value, { allowedTags: [], allowedAttributes: {} });
    }
  },
  description: {
    type: String,
    required: [true, 'Description is required'],
    trim: true,
    maxlength: [1000, 'Description cannot exceed 1000 characters'],
    set: function(value) {
      return sanitizeHtml(value, { allowedTags: [], allowedAttributes: {} });
    }
  },
  completed: {
    type: Boolean,
    default: false
  },
  tags: [{
    type: String,
    trim: true,
    match: [/^#[a-zA-Z0-9_]+$/, 'Tags must start with # and contain only alphanumeric character
    set: function(value) {
      return sanitizeHtml(value, { allowedTags: [], allowedAttributes: {} });
    }
  }],
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
    index: true
  },
  attachments: [{
    filename: String,
    originalName: String,
    path: String,
    size: Number,
    mimetype: String,
    uploadDate: { type: Date, default: Date.now }
  }]
}, {
  timestamps: true
});

// Indexes
todoSchema.index({ userId: 1, createdAt: -1 });
todoSchema.index({ userId: 1, tags: 1 });
todoSchema.index({ userId: 1, completed: 1 });

// Validation for tags
todoSchema.pre('save', function(next) {
  this.tags = [...new Set(this.tags)];

  for (let tag of this.tags) {
    if (!tag.startsWith('#') || tag.length < 2) {
      return next(new Error('Invalid tag format. Tags must start with # and have at least one c
    }
  }

  next();
});

module.exports = mongoose.model('Todo', todoSchema);
```

server/middleware/auth.js

javascript

```javascript
const jwt = require('jsonwebtoken');
const User = require('../models/User');

const auth = async (req, res, next) => {
  try {
    const token = req.header('Authorization')?.replace('Bearer ', '');

    if (!token) {
      return res.status(401).json({ success: false, message: 'Access denied. No token provided.
    }

    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    const user = await User.findById(decoded.userId).select('-password -refreshTokens');

    if (!user) {
      return res.status(401).json({ success: false, message: 'Invalid token.' });
    }

    if (user.isAccountLocked) {
      return res.status(423).json({ success: false, message: 'Account is locked.' });
    }

    req.user = user;
    next();
  } catch (error) {
    if (error.name === 'TokenExpiredError') {
      return res.status(401).json({ success: false, message: 'Token expired.' });
    }
    if (error.name === 'JsonWebTokenError') {
      return res.status(401).json({ success: false, message: 'Invalid token.' });
    }
    res.status(500).json({ success: false, message: 'Server error during authentication.' });
  }
};

module.exports = auth;
```

server/middleware/validation.js

javascript

```javascript
const { body, validationResult } = require('express-validator');
const rateLimit = require('express-rate-limit');

// Validation rules
const registerValidation = [
  body('email')
    .isEmail()
    .normalizeEmail()
    .withMessage('Please provide a valid email'),
  body('password')
    .isLength({ min: 8 })
    .withMessage('Password must be at least 8 characters long')
    .matches(/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]/)
    .withMessage('Password must contain at least one uppercase letter, one lowercase letter, or
];

const loginValidation = [
  body('email').isEmail().normalizeEmail().withMessage('Please provide a valid email'),
  body('password').notEmpty().withMessage('Password is required')
];

const todoValidation = [
  body('title')
    .trim()
    .isLength({ min: 1, max: 100 })
    .withMessage('Title must be between 1 and 100 characters'),
  body('description')
    .trim()
    .isLength({ min: 1, max: 1000 })
    .withMessage('Description must be between 1 and 1000 characters'),
  body('completed')
    .optional()
    .isBoolean()
    .withMessage('Completed must be a boolean'),
  body('tags')
    .optional()
    .isArray()
    .withMessage('Tags must be an array')
    .custom((tags) => {
      if (tags.length > 10) {
        throw new Error('Maximum 10 tags allowed');
      }
      tags.forEach(tag => {
        if (typeof tag !== 'string' || !tag.match(/^#[a-zA-Z0-9_]+$/)) {
          throw new Error('Invalid tag format');
        }
      });
      return true;
    })
];

// Rate limiters
const createAccountLimiter = rateLimit({
  windowMs: 60 * 60 * 1000,
  max: 5,
  message: 'Too many account creation attempts, please try again later.',
});

const loginLimiter = rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 3,
  message: 'Too many login attempts, please try again later.',
});

const todoLimiter = rateLimit({
  windowMs: 60 * 1000,
  max: 30,
  message: 'Too many requests, please try again later.',
});

const handleValidationErrors = (req, res, next) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({
      success: false,
      message: 'Validation failed',
      errors: errors.array()
    });
  }
  next();
};

module.exports = {
  registerValidation,
  loginValidation,
  todoValidation,
  createAccountLimiter,
  loginLimiter,
  todoLimiter,
  handleValidationErrors
};
```

server/routes/auth.js

```javascript
const express = require('express');
const router = express.Router();
const authController = require('../controllers/authController');
const {
  registerValidation,
  loginValidation,
  createAccountLimiter,
  loginLimiter,
  handleValidationErrors
} = require('../middleware/validation');

router.post('/register',
  createAccountLimiter,
  registerValidation,
  handleValidationErrors,
  authController.register
);

router.post('/login',
  loginLimiter,
  loginValidation,
  handleValidationErrors,
  authController.login
);

router.post('/logout', authController.logout);
router.post('/refresh', authController.refreshToken);

module.exports = router;
```

server/controllers/authController.js

javascript

```javascript
const User = require('../models/User');
const jwt = require('jsonwebtoken');

const generateTokens = (userId) => {
  const accessToken = jwt.sign(
    { userId, type: 'access' },
    process.env.JWT_SECRET,
    { expiresIn: '15m' }
  );

  const refreshToken = jwt.sign(
    { userId, type: 'refresh' },
    process.env.JWT_SECRET,
    { expiresIn: '7d' }
  );

  return { accessToken, refreshToken };
};

const register = async (req, res) => {
  try {
    const { email, password } = req.body;

    const existingUser = await User.findOne({ email });
    if (existingUser) {
      return res.status(409).json({
        success: false,
        message: 'User already exists with this email'
      });
    }

    const user = new User({ email, password });
    await user.save();

    const { accessToken, refreshToken } = generateTokens(user._id);

    user.refreshTokens.push({ token: refreshToken });
    if (user.refreshTokens.length > 5) {
      user.refreshTokens = user.refreshTokens.slice(-5);
    }
    await user.save();

    res.status(201).json({
      success: true,
      message: 'User registered successfully',
      data: {
        accessToken,
        refreshToken,
        user: {
          id: user._id,
          email: user.email
        }
      }
    });

  } catch (error) {
    console.error('Registration error:', error);
    res.status(500).json({
      success: false,
      message: 'Server error during registration'
    });
  }
};

const login = async (req, res) => {
  try {
    const { email, password } = req.body;

    const user = await User.findOne({ email });
    if (!user) {
      return res.status(401).json({
        success: false,
        message: 'Invalid credentials'
      });
    }

    if (user.isAccountLocked) {
      return res.status(423).json({
        success: false,
        message: 'Account is temporarily locked'
      });
    }

    const isValidPassword = await user.comparePassword(password);
    if (!isValidPassword) {
      return res.status(401).json({
        success: false,
        message: 'Invalid credentials'
      });
    }

    const { accessToken, refreshToken } = generateTokens(user._id);

    user.refreshTokens.push({ token: refreshToken });
    if (user.refreshTokens.length > 5) {
      user.refreshTokens = user.refreshTokens.slice(-5);
    }
    await user.save();

    res.json({
      success: true,
      message: 'Login successful',
      data: {
        accessToken,
        refreshToken,
        user: {
          id: user._id,
          email: user.email
        }
      }
    });

  } catch (error) {
    console.error('Login error:', error);
    res.status(500).json({
      success: false,
      message: 'Server error during login'
    });
  }
};

const logout = async (req, res) => {
  try {
    const { refreshToken } = req.body;

    if (refreshToken) {
      const decoded = jwt.verify(refreshToken, process.env.JWT_SECRET);
      const user = await User.findById(decoded.userId);

      if (user) {
        user.refreshTokens = user.refreshTokens.filter(
          tokenObj => tokenObj.token !== refreshToken
        );
        await user.save();
      }
    }

    res.json({
      success: true,
      message: 'Logout successful'
    });

  } catch (error) {
    console.error('Logout error:', error);
    res.json({
      success: true,
      message: 'Logout successful'
    });
  }
};

const refreshToken = async (req, res) => {
  try {
    const { refreshToken } = req.body;

    if (!refreshToken) {
      return res.status(401).json({
        success: false,
        message: 'Refresh token required'
      });
    }

    const decoded = jwt.verify(refreshToken, process.env.JWT_SECRET);

    if (decoded.type !== 'refresh') {
      return res.status(401).json({
        success: false,
        message: 'Invalid token type'
      });
    }

    const user = await User.findById(decoded.userId);
    if (!user || !user.refreshTokens.some(tokenObj => tokenObj.token === refreshToken)) {
      return res.status(401).json({
        success: false,
        message: 'Invalid refresh token'
      });
    }

    const { accessToken, refreshToken: newRefreshToken } = generateTokens(user._id);

    user.refreshTokens = user.refreshTokens.filter(
      tokenObj => tokenObj.token !== refreshToken
    );
    user.refreshTokens.push({ token: newRefreshToken });
    await user.save();

    res.json({
      success: true,
      data: {
        accessToken,
        refreshToken: newRefreshToken
      }
    });

  } catch (error) {
    console.error('Refresh token error:', error);
    if (error.name === 'TokenExpiredError' || error.name === 'JsonWebTokenError') {
      return res.status(401).json({
        success: false,
        message: 'Invalid or expired refresh token'
      });
    }
    res.status(500).json({
      success: false,
      message: 'Server error during token refresh'
    });
  }
};

module.exports = {
  register,
  login,
  logout,
  refreshToken
};
```

CLIENT FILES

client/package.json

json

```
{
  "name": "secure-todo-client",
  "version": "1.0.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.17.0",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^14.4.3",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.15.0",
    "react-scripts": "5.0.1",
    "axios": "^1.5.0"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "test:coverage": "react-scripts test --coverage --watchAll=false",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  },
  "jest": {
    "collectCoverageFrom": [
      "src/**/*.{js,jsx}",
      "!src/index.js",
        "!src/reportWebVitals.js"
      ]
    }
  }
```

client/.env

env

```env
REACT_APP_API_URL=http://localhost:5000/api
REACT_APP_MAX_FILE_SIZE=5242880
```

client/src/App.js

javascript

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom';
import { AuthProvider, useAuth } from './contexts/AuthContext';
import Login from './components/Auth/Login';
import Register from './components/Auth/Register';
import Dashboard from './components/Dashboard/Dashboard';
import UpdateTodo from './components/TodoUpdate/UpdateTodo';
import './App.css';

const ProtectedRoute = ({ children }) => {
  const { isAuthenticated } = useAuth();
  return isAuthenticated ? children : <Navigate to="/login" />;
};

const PublicRoute = ({ children }) => {
  const { isAuthenticated } = useAuth();
  return !isAuthenticated ? children : <Navigate to="/dashboard" />;
};

function App() {
  return (
    <AuthProvider>
      <Router>
        <div className="App">
          <Routes>
            <Route path="/" element={<Navigate to="/dashboard" />} />
            <Route
              path="/login"
              element={
                <PublicRoute>
                  <Login />
                </PublicRoute>
              }
            />
            <Route
              path="/register"
              element={
                <PublicRoute>
                  <Register />
                </PublicRoute>
              }
            />
            <Route
          path="/dashboard"
          element={
            <ProtectedRoute>
              <Dashboard />
            </ProtectedRoute>
          }
        />
        <Route
          path="/todo/:id/edit"
          element={
            <ProtectedRoute>
              <UpdateTodo />
            </ProtectedRoute>
          }
        />
      </Routes>
    </div>
  </Router>
</AuthProvider>
  );
}

export default App;
```

Setup Commands

Backend Setup

```bash
cd server
npm install
mkdir uploads temp
```

Frontend Setup

```bash
cd client
npm install
```

Start the Application

```bash
# Terminal 1 - Start MongoDB (if local)
mongod

# Terminal 2 - Start Backend
cd server
npm run dev

# Terminal 3 - Start Frontend
cd client
npm start
```

Testing

```bash
# Backend tests
cd server
npm test

# Frontend tests
cd client
npm test
```

Important Notes

1. **Security**: Change the JWT_SECRET in production
2. **Database**: Set up MongoDB locally or use MongoDB Atlas
3. **File Uploads**: The uploads directory will be created automatically
4. **Environment Variables**: Make sure all .env files are properly configured
5. **CORS**: Update CLIENT_URL in server .env for production

This gives you a complete, secure todo application with all 8 required components and comprehensive security features. Each component addresses the security requirements specified in your prompt.