Complete Database.js File

The Bridge Between Your Backend and Database

What This File Does

The `database.js` file is the connection between your backend (Node.js server) and your database (MySQL). It’ s like a waiter that takes orders from the kitchen (backend) and brings them to the storage room (database).

When your backend needs to:

- Check if a user exists
- Save a new user
- Get a list of files
- Save a file

It uses the functions in `database.js` to do it.

The Complete database.js File

Create a file called `backend/database.js` and paste this code:

```
// ============================================
// DATABASE CONNECTION AND QUERY HELPERS
// ============================================

// Import mysql library
const mysql = require('mysql');

// Create a connection pool (allows multiple connections)
const pool = mysql.createPool({
  connectionLimit: 10,
  host: 'localhost',      // Where MySQL is running
  user: 'root',           // MySQL username
  password: 'password',   // MySQL password (change this!)
  database: 'startup_db'  // Database name
});

// ============================================
// HELPER FUNCTION: Execute Query
// ============================================

/**
 * Execute a database query
 *
 * How it works:
 * 1. Takes a SQL query and parameters
 * 2. Sends query to database
 * 3. Calls callback with results
 *
 * Example:
 * query('SELECT * FROM users WHERE email = ?', ['test@email.com'], (err,
results) => {
 *   if (err) console.log('Error:', err);
 *   else console.log('Results:', results);
 * });
 */
function query(sql, args, callback) {
  pool.query(sql, args, function(err, results) {
    if (err) {
      return callback(err);
    }
    callback(null, results);
  });
}

// ============================================
```

```
// USER FUNCTIONS
// ========================================

/**
 * Get user by email
 *
 * What it does:
 * - Searches database for user with specific email
 * - Returns user if found, null if not found
 *
 * Example:
 * getUserByEmail('test@email.com', (err, user) => {
 *   if (err) console.log('Error:', err);
 *   else console.log('User:', user);
 * });
 */
function getUserByEmail(email, callback) {
  query('SELECT * FROM users WHERE email = ?', [email], (err, results) => {
    if (err) {
      return callback(err, null);
    }

    // If user found, return first result
    if (results.length > 0) {
      callback(null, results[0]);
    } else {
      // If no user found, return null
      callback(null, null);
    }
  });
}

/**
 * Get user by ID
 *
 * What it does:
 * - Searches database for user with specific ID
 * - Returns user if found
 *
 * Example:
 * getUserById(1, (err, user) => {
 *   if (err) console.log('Error:', err);
 *   else console.log('User:', user);
 * });
 */
function getUserById(id, callback) {
```

```
  query('SELECT * FROM users WHERE id = ?', [id], (err, results) => {
    if (err) {
      return callback(err, null);
    }

    if (results.length > 0) {
      callback(null, results[0]);
    } else {
      callback(null, null);
    }
  });
}

/**
 * Create new user
 *
 * What it does:
 * - Inserts new user into database
 * - Returns the new user's ID
 *
 * Example:
 * createUser('John Doe', 'john@email.com', 'password123', (err, userId) =>
{
 *   if (err) console.log('Error:', err);
 *   else console.log('New user ID:', userId);
 * });
 */
function createUser(name, email, password, callback) {
  query(
    'INSERT INTO users (name, email, password) VALUES (?, ?, ?)',
    [name, email, password],
    (err, results) => {
      if (err) {
        return callback(err, null);
      }

      // Return the ID of the newly created user
      callback(null, results.insertId);
    }
  );
}

/**
 * Update user
 *
 * What it does:
```

```
 * - Updates user information in database
 * - Can update name, email, or password
 *
 * Example:
 * updateUser(1, 'Jane Doe', 'jane@email.com', 'newpassword', (err) => {
 *   if (err) console.log('Error:', err);
 *   else console.log('User updated');
 * });
 */
function updateUser(id, name, email, password, callback) {
  query(
    'UPDATE users SET name = ?, email = ?, password = ? WHERE id = ?',
    [name, email, password, id],
    (err, results) => {
      if (err) {
        return callback(err);
      }
      callback(null);
    }
  );
}

/**
 * Delete user
 *
 * What it does:
 * - Removes user from database
 *
 * Example:
 * deleteUser(1, (err) => {
 *   if (err) console.log('Error:', err);
 *   else console.log('User deleted');
 * });
 */
function deleteUser(id, callback) {
  query('DELETE FROM users WHERE id = ?', [id], (err, results) => {
    if (err) {
      return callback(err);
    }
    callback(null);
  });
}

// ================================================
// FILE FUNCTIONS
// ================================================
```

```
/**
 * Get all files for a user
 *
 * What it does:
 * - Gets all files uploaded by a specific user
 * - Returns array of files
 *
 * Example:
 * getFilesByUserId(1, (err, files) => {
 *   if (err) console.log('Error:', err);
 *   else console.log('Files:', files);
 * });
 */
function getFilesByUserId(userId, callback) {
  query(
    'SELECT * FROM files WHERE user_id = ? ORDER BY uploaded_at DESC',
    [userId],
    (err, results) => {
      if (err) {
        return callback(err, null);
      }
      callback(null, results);
    }
  );
}

/**
 * Get file by ID
 *
 * What it does:
 * - Gets specific file by ID
 * - Checks that file belongs to user
 *
 * Example:
 * getFileById(1, 1, (err, file) => {
 *   if (err) console.log('Error:', err);
 *   else console.log('File:', file);
 * });
 */
function getFileById(fileId, userId, callback) {
  query(
    'SELECT * FROM files WHERE id = ? AND user_id = ?',
    [fileId, userId],
    (err, results) => {
      if (err) {
```

```
        return callback(err, null);
      }

      if (results.length > 0) {
        callback(null, results[0]);
      } else {
        callback(null, null);
      }
    }
  );
}

/**
 * Save file
 *
 * What it does:
 * - Saves file information to database
 * - Stores filename, file path, and user ID
 *
 * Example:
 * saveFile(1, 'report.pdf', '/uploads/report.pdf', (err, fileId) => {
 *   if (err) console.log('Error:', err);
 *   else console.log('File saved with ID:', fileId);
 * });
 */
function saveFile(userId, fileName, filePath, callback) {
  query(
    'INSERT INTO files (user_id, name, path) VALUES (?, ?, ?)',
    [userId, fileName, filePath],
    (err, results) => {
      if (err) {
        return callback(err, null);
      }

      // Return the ID of the newly saved file
      callback(null, results.insertId);
    }
  );
}

/**
 * Delete file
 *
 * What it does:
 * - Removes file record from database
 * - Does NOT delete the actual file from server
```

```
 *
 * Example:
 * deleteFile(1, 1, (err) => {
 *   if (err) console.log('Error:', err);
 *   else console.log('File deleted from database');
 * });
 */
function deleteFile(fileId, userId, callback) {
  query(
    'DELETE FROM files WHERE id = ? AND user_id = ?',
    [fileId, userId],
    (err, results) => {
      if (err) {
        return callback(err);
      }
      callback(null);
    }
  );
}

/**
 * Get all files (admin only)
 *
 * What it does:
 * - Gets all files from all users
 * - Only use this for admin features
 *
 * Example:
 * getAllFiles((err, files) => {
 *   if (err) console.log('Error:', err);
 *   else console.log('All files:', files);
 * });
 */
function getAllFiles(callback) {
  query(
    'SELECT * FROM files ORDER BY uploaded_at DESC',
    [],
    (err, results) => {
      if (err) {
        return callback(err, null);
      }
      callback(null, results);
    }
  );
}
```

```
// ============================================
// EXPORT ALL FUNCTIONS
// ============================================

/**
 * Export all functions so they can be used in other files
 *
 * Example of using in another file:
 * const db = require('./database');
 * db.getUserByEmail('test@email.com', (err, user) => {
 *   // Use the user
 * });
 */
module.exports = {
  query: query,
  getUserByEmail: getUserByEmail,
  getUserById: getUserById,
  createUser: createUser,
  updateUser: updateUser,
  deleteUser: deleteUser,
  getFilesByUserId: getFilesByUserId,
  getFileById: getFileById,
  saveFile: saveFile,
  deleteFile: deleteFile,
  getAllFiles: getAllFiles
};
```

How to Use This File

In Your Auth Routes (backend/routes/auth.js)

```
const db = require('../database');

// In your login route
router.post('/login', (req, res) => {
  const { email, password } = req.body;

  // Use the database function
  db.getUserByEmail(email, (err, user) => {
    if (err) {
      return res.json({ success: false, message: 'Database error' });
    }

    if (!user) {
      return res.json({ success: false, message: 'Email not found' });
    }

    if (user.password !== password) {
      return res.json({ success: false, message: 'Password incorrect' });
    }

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

// In your signup route
router.post('/signup', (req, res) => {
  const { name, email, password } = req.body;

  // First check if email exists
  db.getUserByEmail(email, (err, user) => {
    if (err) {
      return res.json({ success: false, message: 'Database error' });
    }

    if (user) {
      return res.json({ success: false, message: 'Email already exists' });
    }

    // Create new user
```

```
    db.createUser(name, email, password, (err, userId) => {
      if (err) {
        return res.json({ success: false, message: 'Could not create user'
});
      }

      res.json({ success: true, message: 'User created successfully' });
    });
  });
});
```

In Your Files Routes (backend/routes/files.js)

```
const db = require('../database');

// Get all files for a user
router.get('/files', (req, res) => {
  const userId = req.session.userId;

  db.getFilesByUserId(userId, (err, files) => {
    if (err) {
      return res.json({ success: false, message: 'Database error' });
    }

    res.json({ success: true, files: files });
  });
});

// Save uploaded file
router.post('/upload', upload.single('file'), (req, res) => {
  const userId = req.session.userId;
  const fileName = req.file.originalname;
  const filePath = req.file.path;

  db.saveFile(userId, fileName, filePath, (err, fileId) => {
    if (err) {
      return res.json({ success: false, message: 'Could not save file' });
    }

    res.json({ success: true, message: 'File uploaded successfully' });
  });
});

// Download file
router.get('/download/:id', (req, res) => {
  const fileId = req.params.id;
  const userId = req.session.userId;

  db.getFileById(fileId, userId, (err, file) => {
    if (err || !file) {
      return res.json({ success: false, message: 'File not found' });
    }

    res.download(file.path, file.name);
```

```
  });
});
```

In Your Users Routes (backend/routes/users.js)

```
const db = require('../database');

// Get user info
router.get('/user-info', (req, res) => {
  if (!req.session || !req.session.userId) {
    return res.json({ success: false, message: 'Not logged in' });
  }

  const userId = req.session.userId;

  db.getUserById(userId, (err, user) => {
    if (err) {
      return res.json({ success: false, message: 'Database error' });
    }

    if (!user) {
      return res.json({ success: false, message: 'User not found' });
    }

    res.json({ success: true, user: user });
  });
});
```

Understanding Each Function

getUserByEmail(email, callback)

What it does: Searches for a user by email address.

When to use: Login (check if email exists), signup (check if email already registered).

How it works:

1. Takes email as input

2. Queries database: `SELECT * FROM users WHERE email = ?`
3. If user found, returns user object
4. If not found, returns null

Example:

```
db.getUserByEmail('test@email.com', (err, user) => {
  if (err) {
    console.log('Error:', err);
  } else if (user) {
    console.log('User found:', user.name);
  } else {
    console.log('User not found');
  }
});
```

---

createUser(name, email, password, callback)

What it does: Creates a new user in the database.

When to use: Signup (when user submits the signup form).

How it works:

1. Takes name, email, password as input
2. Inserts into database: `INSERT INTO users (name, email, password) VALUES (?, ?, ?)`
3. Returns the ID of the new user

Example:

```
db.createUser('John Doe', 'john@email.com', 'password123', (err, userId) =>
{
  if (err) {
    console.log('Error creating user:', err);
  } else {
    console.log('User created with ID:', userId);
  }
});
```

---

getFilesByUserId(userId, callback)

What it does: Gets all files uploaded by a specific user.

When to use: Dashboard (show list of files user can download).

How it works:

1. Takes user ID as input
2. Queries database: `SELECT * FROM files WHERE user_id = ?`
3. Returns array of files

Example:

```
db.getFilesByUserId(1, (err, files) => {
  if (err) {
    console.log('Error:', err);
  } else {
    console.log('User has', files.length, 'files');
    files.forEach(file => {
      console.log('File:', file.name);
    });
  }
});
```

---

saveFile(userId, fileName, filePath, callback)

What it does: Saves file information to database.

When to use: File upload (after file is uploaded to server).

How it works:

1. Takes user ID, filename, and file path as input
2. Inserts into database: `INSERT INTO files (user_id, name, path) VALUES (?, ?, ?)`
3. Returns the ID of the new file

Example:

```
db.saveFile(1, 'report.pdf', '/uploads/report.pdf', (err, fileId) => {
  if (err) {
    console.log('Error saving file:', err);
  } else {
    console.log('File saved with ID:', fileId);
  }
});
```

Installation Steps

Step 1: Install MySQL Package

```
npm install mysql
```

Step 2: Create the File

Create a file at `backend/database.js` and paste the complete code above.

Step 3: Update Configuration

In the database.js file, update these values with your MySQL credentials:

```
const pool = mysql.createPool({
  connectionLimit: 10,
  host: 'localhost',      // Change if MySQL is on different server
  user: 'root',           // Change to your MySQL username
  password: 'password',   // Change to your MySQL password
  database: 'startup_db'  // Change to your database name
});
```

Step 4: Create Database

Before running your app, create the database and tables:

```
# Login to MySQL
mysql -u root -p

# Create database
CREATE DATABASE startup_db;

# Use the database
USE startup_db;

# Create users table
CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL UNIQUE,
  password VARCHAR(255) NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

# Create files table
CREATE TABLE files (
  id INT AUTO_INCREMENT PRIMARY KEY,
  user_id INT NOT NULL,
  name VARCHAR(255) NOT NULL,
  path VARCHAR(255) NOT NULL,
  uploaded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (user_id) REFERENCES users(id)
);
```

Step 5: Test Connection

Add this to your server.js to test the connection:

```
const db = require('./backend/database');

// Test database connection
db.query('SELECT 1', [], (err, results) => {
  if (err) {
    console.log('Database connection failed:', err);
  } else {
    console.log('Database connected successfully!');
  }
});
```

Common Errors and Solutions

Error: "Cannot find module 'mysql'"

Solution: Install the mysql package:

```
npm install mysql
```

Error: "Access denied for user 'root'@'localhost'"

Solution: Check your MySQL username and password in database.js. Make sure they match your MySQL credentials.

Error: "Unknown database 'startup_db'"

Solution: Create the database first:

```
mysql -u root -p
CREATE DATABASE startup_db;
```

Error: "Table 'startup_db.users' doesn't exist"

Solution: Create the tables using the SQL commands above.

Summary

The `database.js` file is your bridge to the database. It contains:

1. **Connection setup:** Connects to MySQL
2. **Helper functions:** Reusable functions for common database operations
3. **User functions:** Create, read, update, delete users
4. **File functions:** Save, retrieve, delete files

Every time your backend needs to access the database, it uses these functions. This keeps your code organized and prevents repetition.

Now you have the complete database layer. Your system is ready to go! 🚀