**FLASK TODO APPLICATION TECHNICAL REPORT**

**Submitted By:**

**Karthik P**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **SI NO** | **SECTIONS** | **PAGE NO.** |
| **01** | Introduction | **01** |
| **02** | Application Architecture | **02** |
| **03** | Database Structure | **03** |
| **04** | Backend Implementation | **04** |
| **05** | Frontend Implementation | **05** |
| **06** | Application Flow | **06** |
| **07** | Key Features | **07** |
| **08** | User Interface Design | **08** |
| **09** | JavaScript Functionality | **09** |
| **10** | Error Handling | **10** |
| **11** | Recommendations for Improvement | **11** |
| **12** | Screenshots | **12** |
| **13** | Conclusion | **14** |
| **14** | License information | **14** |

**INTRODUCTION**

This report provides a comprehensive analysis of a Flask-based Todo application. The application offers a user-friendly interface for task management with features like adding, updating, deleting, and marking tasks as complete. Built with Python's Flask framework on the backend and HTML/CSS/JavaScript on the frontend, the application demonstrates a classic full-stack web development architecture with a SQLite database for persistence.

**APPLICATION ARCHITECTURE**

The application follows the Model-View-Controller (MVC) architectural pattern:

* **Model**: SQLAlchemy ORM for database interactions (Todo class)
* **View**: Jinja2 templates for HTML rendering
* **Controller**: Flask routes for handling HTTP requests

The project structure is organized as follows:

***-------------------------------------------------------------------------------------------------------***

todo-app/

│

├── app.py # Main application file with routes

├── flask-todo-report.md # Report File For Todo App In MarkDown File

├── todo.db # SQLite database (generated at runtime)

├── static/ # Static assets

│ ├── css/

│ │ └── style.css # Main stylesheet

│ ├── images/

│ │ └── calendar-checklist.svg # App icon

│ │ └── todo-bg-image.jpg # App background image

│ └── js/

│ ├── toggle\_checkbox.js # Task completion toggle functionality

│ └── update.js # Task update form handling

├── templates/ # Jinja2 templates

│ ├── base.html # Base template with common elements

│ ├── error.html # Error page template

│ ├── index.html # Homepage template

│ ├── task.html # Task management page template

│ └── update.html # Task update page template

├── License # MIT License

└── .gitignore # Git ignore file

*-----------------------------------------------------------------------------*

**DATABASE STRUCTURE**

The application uses SQLAlchemy to interact with a SQLite database. The database schema consists of a single **Todo** model with the following fields:

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| id | Integer | Primary key |
| text | String (200) | Task description (required) |
| date\_created | DateTime | Creation timestamp (auto-generated) |
| completed | Boolean | Task completion status (default: False) |

Database model definition from **app.py**:

***-------------------------------------------------------------------------------------------------------***

*class Todo(db.Model):*

*id = db.Column(db.Integer, primary\_key = True)*

*text = db.Column(db.String(200), nullable = False)*

*date\_created = db.Column(db.DateTime, default = datetime.now())*

*completed = db.Column(db.Boolean, default=False)*

*def \_\_repr\_\_(self):*

*return '<Task %r>' % self.id*

***-----------------------------------------------------------------------------------------------------------------***

**BACKEND IMPLEMENTATION**

The backend is implemented using Flask with the following key components:

**Route Structure**

|  |  |  |  |
| --- | --- | --- | --- |
| Route | Method | Function | Description |
| / | GET | get\_tasks() | Display homepage with all tasks |
| /task | GET/POST | task() | Display task management page / Add new task |
| /task/delete/<int:id> | GET | delete\_task(id) | Delete a specific task |
| /task/update/<int:id> | GET/POST | update(id) | Display update form / Update task |
| /task/toggle/<int:id> | POST | toggle\_task(id) | Toggle task completion status (AJAX) |

**KEY BACKEND FEATURES**

1. **Task Creation with Duplicate Detection**:
   * The application checks for duplicate tasks before insertion
   * Provides user feedback on success or failure
2. **AJAX Task Status Toggle**:
   * Allows updating task completion status without page refresh
   * Returns JSON responses for frontend state management
3. **Error Handling**:
   * Custom 404 error page
   * Try-except blocks for database operations
   * User-friendly error messages

**FRONTEND IMPLEMENTATION**

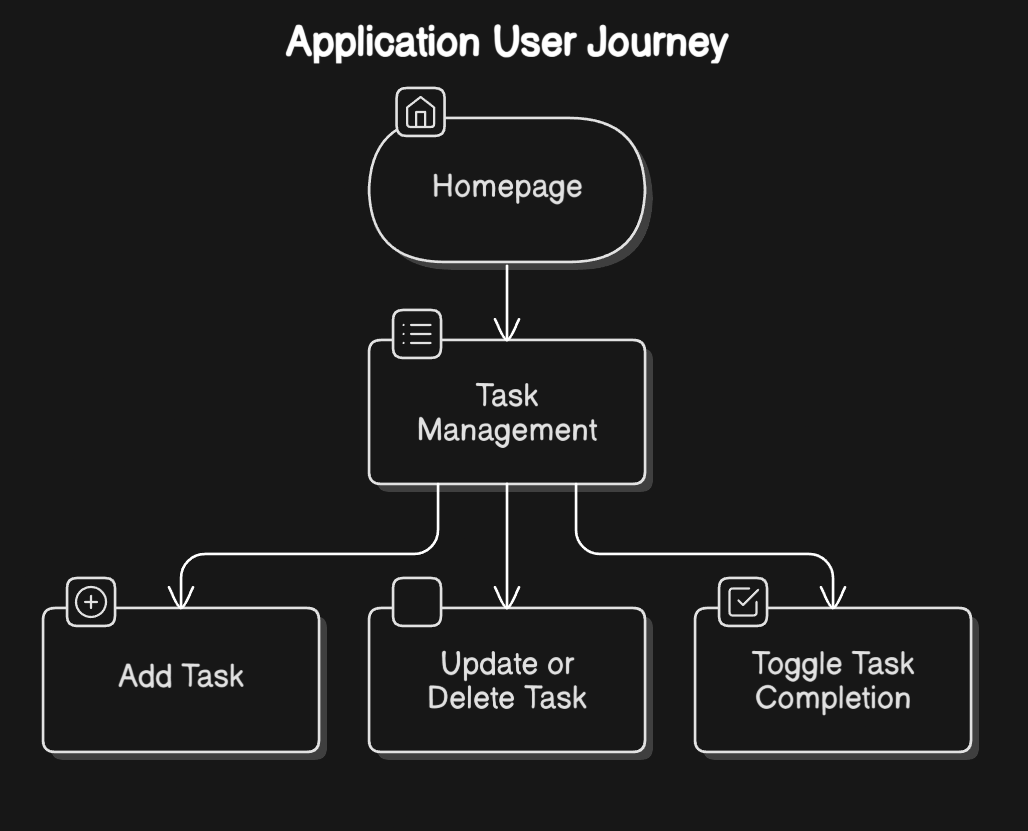
The frontend utilizes a combination of technologies:

* **HTML**: Jinja2 templates for dynamic content rendering
* **CSS**: Custom styling with CSS variables and animations
* **JavaScript**: AJAX requests and DOM manipulations

**TEMPLATE STRUCTURE**

1. **base.html**: Base template with common elements
   * HTML boilerplate
   * Meta tags
   * CSS and JavaScript includes
   * Favicon
2. **index.html**: Simple homepage displaying all tasks
3. **task.html**: Main task management interface
   * Task creation form
   * Task listing with checkboxes
   * Edit/delete controls
   * Notification system
4. **update.html**: Task update form
   * Input field for task text
   * Success popup with redirect
5. **error.html**: Custom 404 page

**APPLICATION FLOW**

****

The typical user journey through the application follows this flow:

1. **Homepage**: User lands on the homepage (/) showing a list of all tasks
2. **Task Management**: User navigates to the task management page (/task)
3. **Add Task**: User submits the form to add a new task
4. **Update/Delete**: User can edit or delete existing tasks
5. **Toggle Completion**: User can mark tasks as complete/incomplete

**KEY FEATURES**

**1. TASK MANAGEMENT**

* **Create**: Add new tasks with duplicate detection
* **Read**: View all tasks with timestamps
* **Update**: Edit task text
* **Delete**: Remove unwanted tasks

**2. TASK COMPLETION TRACKING**

* Interactive checkboxes to mark tasks as complete
* Visual indication of completed tasks (strikethrough)
* AJAX implementation for seamless experience

**3. USER NOTIFICATIONS**

* Success messages for task operations
* Error messages for failures
* Auto-dismissing notifications

**4. RESPONSIVE DESIGN**

* Flexible layout that works on different screen sizes
* Modern UI with animations and transitions

**USER INTERFACE DESIGN**

The UI follows modern design principles with attention to usability:

**COLOR SCHEME**

The application uses a defined color palette with CSS variables:

* **Primary colors:** Blue variants (--info: #3498db)
* **Status colors:** Success green (--success: #2ecc71), Warning yellow (--warning: #f9ca24), Danger red (--danger: #f37272)
* **Background colors:** Dark (--dark: #2c3e50), Light (--light: #ecf0f1)

**VISUAL ELEMENTS**

1. **TASK ITEMS**:
   * Clean, card-like appearance
   * Custom checkbox animations
   * Clear visual hierarchy with task text, timestamp, and actions
2. **FORMS**:
   * Minimal, focused design
   * Animated input fields
   * Clear button styling
3. **NOTIFICATIONS**:
   * Color-coded messages (green for success, yellow for errors)
   * Subtle animations for appearance/disappearance
4. **BACKGROUND**:
   * Gradient background
   * Appropriate spacing and padding

**JAVASCRIPT FUNCTIONALITY**

Two main JavaScript files handle client-side functionality:

**1. toggle\_checkbox.js**

* Manages task completion status via AJAX
* Provides visual feedback on status change
* Handles error states and reverts UI on failure

**Key implementation:**

***-------------------------------------------------------------------------------------------------------***

// Send AJAX request to update task status

function updateTaskStatus(taskId, isCompleted) {

fetch(`/task/toggle/${taskId}`, {

method: "POST",

headers: {

"Content-Type": "application/json",

"X-Requested-With": "XMLHttpRequest",

},

body: JSON.stringify({ completed: isCompleted }),

})

.then((response) => {

if (!response.ok) {

throw new Error("Network response was not ok");

}

return response.json();

})

.then((data) => {

console.log("Task updated:", data);

})

.catch((error) => {

// Revert checkbox state if there was an error

const checkbox = document.getElementById(taskId);

if (checkbox) {

checkbox.checked = !isCompleted;

}

});

}

***-----------------------------------------------------------------------------------------------------------------***

**2. update.js**

* Handles the task update form submission
* Displays success popup after update
* Manages redirection after update

**ERROR HANDLING**

The application implements several error handling mechanisms:

1. **BACKEND ERROR HANDLING**:
   * Try-except blocks for database operations
   * Appropriate HTTP status codes
   * Informative error messages
2. **FRONTEND ERROR HANDLING**:
   * Form validation
   * AJAX error handling
   * UI state management for errors
3. **404 ERROR PAGE**:
   * Custom error template
   * User-friendly message
   * Navigation back to homepage

**RECOMMENDATIONS FOR IMPROVEMENT**

While the application is functional and well-designed, several enhancements could be made:

**1. SECURITY IMPROVEMENTS**

* Implement CSRF protection for forms
* Add input sanitization for user-provided content
* Implement user authentication and authorization

**2. FEATURE ENHANCEMENTS**

* Add task categories/tags
* Implement task due dates
* Add task priority levels
* Enable task search and filtering

**3. TECHNICAL IMPROVEMENTS**

* Add automated tests (unit and integration)
* Implement proper logging
* Create a configuration file for environment settings
* Consider using a more scalable database for production

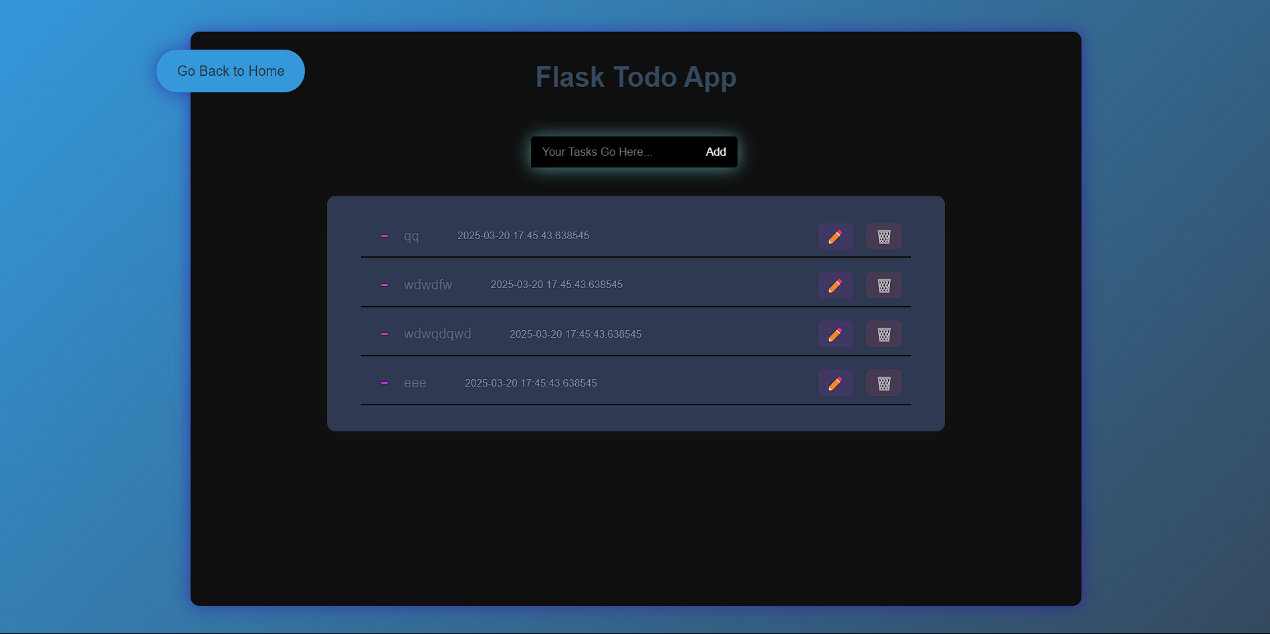
**4. UX IMPROVEMENTS**

* Add confirmation dialogs for delete operations
* Implement drag-and-drop task reordering
* Add keyboard shortcuts for common operations
* Implement dark/light theme toggle

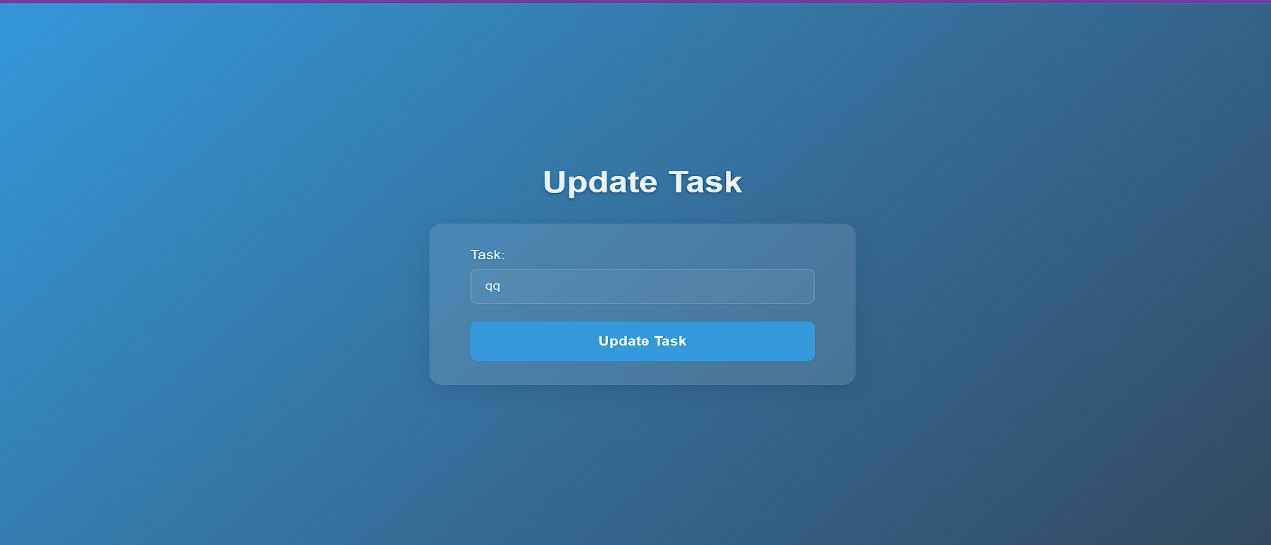
**SCREENSHOTS**

**HOMEPAGE**

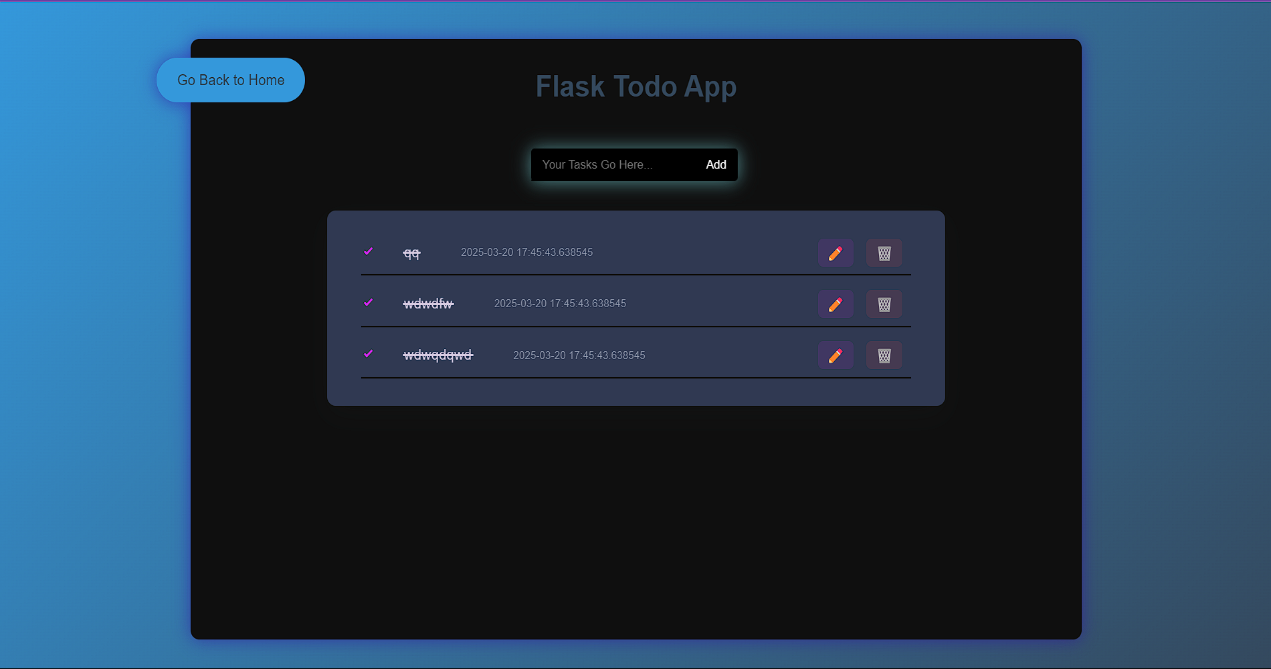


**TASK MANAGER**

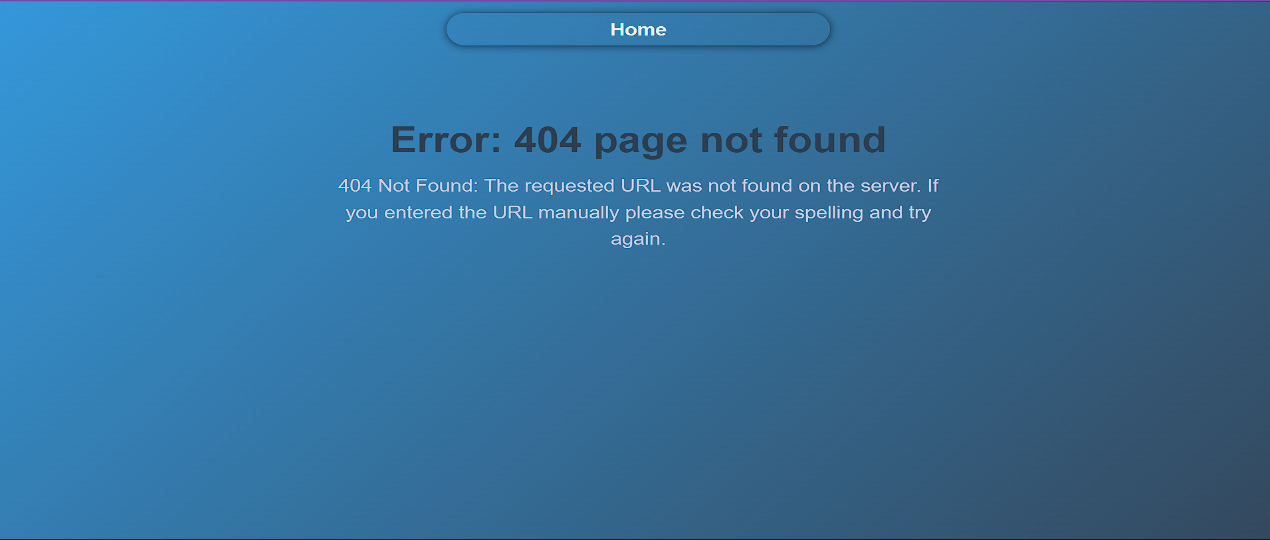
**UPDATE TASK**

****

**TASK COMPLETED**

****

**ERROR PAGE NOT FOUND**

****

**CONCLUSION**

This Flask Todo application demonstrates a well-structured web application with a clean separation of concerns. It provides core task management functionality with an intuitive user interface and smooth interactions. The use of modern web technologies like AJAX for asynchronous updates enhances the user experience.

The application showcases the power of Flask as a lightweight but capable web framework for building interactive web applications. With the suggested improvements, it could be further enhanced to provide an even more robust and feature-rich task management solution.

**LICENSE**

This project is licensed under the MIT License - see the LICENSE file for details.