BucketLisk: Web Application Documentation

1. Overview

This document provides an overview of the BucketLisk web application, including its functionality, REST API endpoints, dependencies, credentials, feedback procedure, and code commit workflow.

Application Description:
BucketLisk is a full-stack task management web application that allows users to stay organised by creating, viewing, updating, completing, and deleting tasks. Each task has a title, description, due date, completion status, and creation date. The frontend is a static HTML/CSS/JavaScript site with a dashboard for managing tasks, while the backend is a Node.js/Express REST API that persists task data in a MongoDB database. BucketLisk supports sorting tasks by due date or creation date and provides a filtered view of tasks based on completion state.

---

2. REST API Endpoints

All BucketLisk endpoints are served from the Express backend running on `http://localhost:3001`. All requests and responses use JSON. The API does not currently require authentication tokens; access is restricted by network/host (see *Section 4: Credentials*).

`GET /api/tasks`

- **Method:** GET
- **Purpose:** Retrieve all tasks from the BucketLisk database, with optional sorting.
- **Authentication:** Not required.
- **Input parameters (query string, optional):**
    - `sortBy=dueDate`: sort tasks by due date ascending
    - `sortBy=dateCreated`: sort tasks by creation date ascending
- **Request body:** none
- **Response (200 OK):** JSON array of task objects
- **Error responses:** `404` if no tasks found, `500` on server error
- **Example request:**

```
GET http://localhost:3001/api/tasks?sortBy=dueDate
```

Example response:

```
[
  {
    "_id": "65f1a2b3c4d5e6f7a8b9c0d1",
    "title": "Finish documentation",
    "description": "Write the BucketLisk project documentation",
    "completed": false,
    "dueDate": "2026-05-10T00:00:00.000Z",
    "dateCreated": "2026-05-05T12:34:56.000Z"
  }
]
```

`POST /api/tasks/todo`

- **Method:** POST
- **Purpose:** Create a new task and save it to the BucketLisk database.
- **Authentication:** Not required.
- **Request body (JSON):**
  - `title` (string, required)
  - `description` (string, required)
  - `dueDate` (ISO date string, required)
- **Response (200 OK):** `{ "task": <new task object>, "message": "New task created successfully!" }`
- **Error responses:** `500` on validation or database failure
- **Example request body:**

```
{
  "title": "Buy groceries",
  "description": "Milk, bread, eggs",
  "dueDate": "2026-05-12"
}
```

`PATCH /api/tasks/complete/:id`

- **Method:** PATCH
- **Purpose:** Mark a task as complete.
- **Authentication:** Not required.
- **Input:** `id` URL parameter (the MongoDB `_id` of the task)
- **Request body (JSON):** `{ "completed": true }`
- **Response (200 OK):** `{ "task": <updated task>, "message": "Task set to 'Complete'" }`
- **Error responses:** `404` if task ID not found, `500` on server error
- **Example:**

```
PATCH http://localhost:3001/api/tasks/complete/65f1a2b3c4d5e6f7a8b9c0d1
Content-Type: application/json

{ "completed": true }
```

PATCH /api/tasks/notComplete/:id

- **Method:** PATCH
- **Purpose:** Mark a task as not complete (revert completion).
- **Authentication:** Not required.
- **Input:** `id` URL parameter
- **Request body (JSON):** `{ "completed": false }`
- **Response (200 OK):** `{ "task": <updated task>, "message": "Task set to 'Not Complete'" }`
- **Error responses:** `404`, `500`

DELETE /api/tasks/delete/:id

- **Method:** DELETE
- **Purpose:** Permanently delete a task by ID.
- **Authentication:** Not required.
- **Input:** `id` URL parameter
- **Request body:** none
- **Response (200 OK):** `{ "task": <deleted task>, "message": "Task deleted successfully!" }`
- **Error responses:** `404`, `500`
- **Example:**

```
DELETE http://localhost:3001/api/tasks/delete/65f1a2b3c4d5e6f7a8b9c0d1
```

PUT /api/tasks/update/:id

- **Method:** PUT
- **Purpose:** Update the editable fields of an existing task.
- **Authentication:** Not required.
- **Input:** `id` URL parameter, plus updated fields in the JSON request body.
- **Request body (JSON):**
  - `title` (string)
  - `description` (string)
  - `dueDate` (ISO date string)
- **Response (200 OK):** `{ "task": <updated task>, "message": "Task updated successfully!" }`
- **Error responses:** `404` if task ID not found, `500` on server error
- **Example request body:**

```
{
  "title": "Buy groceries (updated)",
  "description": "Milk, bread, eggs, butter",
  "dueDate": "2026-05-13"
}
```

---

3. Dependencies

BucketLisk is built on the following stack and third-party packages.

Backend (Node.js) dependencies

Package	Version	Purpose	License
express	^5.2.1	HTTP server and routing framework	MIT
mongoose	^9.3.3	MongoDB ODM for schema/model management	MIT
mongodb	^7.1.1	Official MongoDB Node.js driver	Apache 2.0
cors	^2.8.6	Cross-Origin Resource Sharing middleware	MIT
dotenv	^17.3.1	Loads environment variables from `.env`	BSD-2-Clause

Runtime / external services

- **Node.js:** v18 LTS or higher recommended (required to run the BucketLisk backend).
- **MongoDB:** either a local instance or a hosted MongoDB Atlas cluster. The connection string is provided via the `MONGO_URI` environment variable.
- **Web browser:** any modern browser (Chrome, Firefox, Edge, Safari) for the BucketLisk frontend pages.

Frontend dependencies

The BucketLisk frontend is plain HTML, CSS, and vanilla JavaScript, with no build step or package manager. It uses the browser's native `fetch` API to communicate with the backend. No CDN-hosted libraries or external scripts are required.

Installation instructions

1. Install Node.js (v18+) from nodejs.org.
2. Clone the BucketLisk repository.
3. From the `backend/` directory, run `npm install` to install all dependencies listed in `package.json`.
4. Create a `.env` file in `backend/` containing your `MONGO_URI`.

5. Start the backend with `npm start` (or `node index.js`).
6. Open `frontend/index.html` in a browser, or serve the `frontend/` directory with any static file server.

Notes:

- `package-lock.json` ensures consistent dependency versions across installs.
- All listed packages use permissive open-source licenses (MIT/Apache/BSD) and are free for commercial and academic use.

---

4. Credentials & Authentication

Environment variables

BucketLisk uses MongoDB as its database. Sensitive connection details are stored in a `.env` file inside the `backend/` directory and are loaded at runtime by the `dotenv` package. The `.env` file is excluded from version control via `.gitignore` and must never be committed to the repository.

Example `backend/.env` file:

```
MONGO_URI=mongodb+srv://<username>:<password>@<cluster>.mongodb.net/bucketlisk
```

Components requiring credentials

Component	Credential type	Where stored
MongoDB Atlas (BucketLisk database)	Connection string with username/password	`backend/.env` as `MONGO_URI`
MongoDB Atlas web console	Email + password (with optional MFA)	Personal password manager (not in repo)
BucketLisk GitHub repository	SSH key or personal access token	Developer's local machine
Hosting account (if deployed)	Provider login	Personal password manager

Authentication mechanism

The current BucketLisk REST API does not enforce per-request authentication. It is intended for local development and demonstration. For production deployment, a token-based authentication layer (e.g. JWT) should be added before exposing the API publicly.

Credential handling rules

- Never commit `.env` files or any secrets to the BucketLisk Git repository.
- Rotate the MongoDB password if a credential is suspected to be leaked.
- Use MongoDB Atlas IP allow-listing to restrict database access to known hosts.
- Share credentials between team members only through a secure password manager, never via chat, email, or commits.

---

5. Procedure for Feedback

Policies and guidelines

- **Privacy:** BucketLisk stores only task data the user enters; no personal data is collected.
- **Acceptable use:** Users must not submit unlawful, abusive, or copyrighted content as task data.
- **Security:** Vulnerability reports must be sent privately to the BucketLisk maintainer rather than disclosed publicly.
- **Content management:** The maintainer reserves the right to remove any content that violates the acceptable use policy.

Feedback collection channels

1. **Support page:** users submit feedback, bug reports, or suggestions via the `support.html` page on the BucketLisk website.
2. **GitHub Issues:** technical bugs and feature requests are tracked as issues in the BucketLisk GitHub repository.
3. **Direct contact:** users may email the BucketLisk maintainer for sensitive or security-related feedback.

How feedback is identified, applied, and actioned

1. **Receive:** feedback arrives via the support form, GitHub Issues, or email.
2. **Acknowledge:** the maintainer acknowledges the submitter within 3 working days.
3. **Triage:** each item is logged as a GitHub Issue and labelled (`bug`, `enhancement`, `question`, `security`) and assigned a priority (`P1` to `P3`).
4. **Backlog:** triaged items are added to the BucketLisk backlog and ordered by priority and effort.
5. **Implementation:** a developer creates a feature branch from `develop` referencing the issue number (see Section 6) and implements the change.
6. **Testing:** the change is tested locally and on the `develop` branch.
7. **Release:** after merge to `master`, the issue is closed and the submitter is notified that their feedback has been actioned.
8. **Review:** closed feedback is reviewed quarterly to identify recurring themes and inform the BucketLisk product roadmap.

This process ensures that every piece of feedback received is recorded in a tracked location, given a status, assigned to a person, and either delivered as a code change or formally declined with a reason.

---

6. Procedure for Code Commit Changes (Git Workflow)

BucketLisk uses a feature-branch workflow with two long-lived branches: `develop` (integration) and `master` (production-ready).

Step-by-step process

1. **Pull the latest** `develop`**:**

```
git checkout develop
git pull origin develop
```

2. **Create a feature branch from** `develop`**:**

```
git checkout -b feature/<short-description>
```

For example: `git checkout -b feature/add-task-filter`.

3. **Make and test changes locally.** Run the BucketLisk backend (`npm start`) and exercise the affected pages in the browser. Add or update tests as appropriate.
4. **Stage and commit changes with a clear message:**

```
git add <files>
git commit -m "Add task filter dropdown to dashboard"
```

Commit frequently in small logical units; use the imperative mood ("Add", "Fix", "Refactor"). Reference an issue number where applicable (e.g. `"Fix #42: prevent duplicate task submission"`).

5. **Push the branch to the remote repository:**

```
git push -u origin feature/<short-description>
```

6. **Open a Pull Request** from the feature branch into `develop` on GitHub. The PR description should explain *what* changed and *why*, and include screenshots for UI changes.
7. **Code review:** at least one reviewer must approve before merge. Address all review feedback by pushing additional commits to the same branch.
8. **CI/CD:** the BucketLisk CI pipeline runs tests and linters automatically on every push and pull request. The PR cannot be merged until checks pass.
9. **Merge into** `develop`**:** once approved and green, merge the PR into `develop`.

10. **Post-merge cleanup:**

```
git checkout develop
git pull origin develop
git branch -d feature/<short-description>
git push origin --delete feature/<short-description>
```

11. **Promote to `master`:** after thorough integration testing on `develop`, open a PR from `develop` into `master` for the next BucketLisk release. Once merged, the deployment pipeline publishes the new version automatically.

Best practices

- Commit frequently and in small, focused units.
- Use clear, descriptive commit messages in the imperative mood.
- Always test locally before pushing.
- Participate constructively in code reviews, both as author and reviewer.
- Maintain and update tests alongside code changes.
- Update this documentation whenever BucketLisk endpoints, dependencies, or workflows change.
- Never commit secrets or `.env` files.
- Never force-push to `develop` or `master`.

Handling problems

- **Merge conflicts:** resolve them locally by rebasing the feature branch onto the latest `develop`:

```
git fetch origin
git rebase origin/develop
# resolve conflicts, then:
git add <resolved files>
git rebase --continue
git push --force-with-lease
```

- **Access issues:** if a developer cannot push to the BucketLisk repository, contact the repository administrator to verify their GitHub permissions and SSH key.
- **Failing CI:** reproduce the failure locally, fix the underlying issue, and push a new commit. Never bypass CI checks (e.g. `--no-verify`) without administrator approval.