## **Craft note management API – Group B Project Acceptance Test (PAT) Document**

**1. Introduction**

This document outlines the Project Acceptance Test (PAT) for the Craft note management API, designed to manage note records within a database and integrate with an external Zip Code API (optional).

**2. Scope**

These tests focus on the core functionalities of the API as described in the project overview.

**3. Acceptance Criteria**

The API will be considered successful if it meets the following criteria for each endpoint:

**3.1 GET /api/notes**

* **Functionality:**
  + Retrieves a list of all notes stored in the database.
* **Data Validation:**
  + Ensures all returned notes have a populated updatedAt field (not null).
* **Response Codes:**
  + Returns a 200 OK status code for a successful request.
  + Returns appropriate error codes for invalid requests (e.g., 400 Bad Request for malformed queries).

**3.2 GET /api/notes/{noteId}**

* **Functionality:**
  + Retrieves a single note where the ID matches the provided noteId.
* **Data Validation:**
  + Validates that the retrieved note has a populated updatedAt field.
* **Response Codes:**
  + Returns a 200 OK status code for a successful retrieval.
  + Returns a 404 Not Found status code if the specified note ID doesn't exist.
  + Returns appropriate error codes for invalid requests (e.g., 400 Bad Request for invalid note ID format).

**3.3 POST /api/notes**

* **Functionality:**
  + Inserts a new note record into the database.
* **Request Validation:**
  + Validates the request body against a defined JSON schema, ensuring required fields are present:
    - id (integer)
    - title (string)
    - content (string)
    - createdAt (string, date-time format compatible with database)
    - updatedAt (string, date-time format compatible with database)
* **Response Codes:**
  + Returns a 201 Created status code for a successful insertion.
  + Returns a 400 Bad Request status code for invalid request bodies (e.g., missing required fields, malformed data).

**3.4 PUT /api/notes/{noteId}**

* **Functionality:**
  + Updates an existing note record or creates a new one if the ID doesn't exist.
* **Request Validation:**
  + Validates the request body format if provided (optional update fields).
* **Response Codes:**
  + Returns a 200 OK status code for successful update or creation.
  + Returns a 404 Not Found status code if the specified note ID doesn't exist during update.
  + Returns appropriate error codes for invalid requests (e.g., 400 Bad Request for malformed data).

**3.5 DELETE /api/notes/{noteId}**

* **Functionality:**
  + Deletes the note record associated with the provided noteId.
* **Response Codes:**
  + Returns a 204 No Content status code for successful deletion.
  + Returns a 404 Not Found status code if the specified note ID doesn't exist.
  + Returns a 400 Bad Request status code for invalid requests (e.g., invalid note ID format).

**3.6 Optional Zip Code Integration**

* If implemented, the API should provide an additional endpoint (e.g., /api/notes/{noteId}/zip-code) to retrieve Zip Code details from an external API.
* Functionality and acceptance criteria for this feature can be defined separately based on specific implementation details.

**4. Testing Methodology**

* Use a testing tool like Postman to send requests to the API endpoints.
* Test various scenarios covering successful requests, invalid requests, and edge cases.
* Verify response codes, data content, and error messages.

**5. Documentation Review**

* Review the RAML API specification for clarity and completeness.
* Verify that the documentation accurately reflects the implemented API behavior.
* Ensure the documentation includes examples, error codes, and versioning information.

**6. Security Considerations**

* Verify that sensitive data like passwords and API keys are not exposed in logs or runtime outputs.
* Confirm that the API uses secure authentication mechanisms like client credentials flow.

**7. Deployment and Environment**

* Ensure the API is deployed to separate development, testing, and production environments.
* Verify that CI/CD practices are in place for automated deployments.

**8. Pass/Fail Criteria**

The API will be considered successful if it meets the following criteria:

* All functionalities listed in the above sections (or with optional Zip Code integration) work as expected.
* The API adheres to security best practices .
* The API is well-documented and easy to use for potential consumers.
* Unit test coverage reaches at least 75%.
* The API is successfully deployed across development, testing, and production environments with CI/CD practices in place.

**9. Next Steps**

* Conduct user acceptance testing (UAT) to gather feedback from craft Team.
* Refine the API based on UAT results and ongoing feedback.

This PAT document serves as a guideline for testing and acceptance of the Craft note management API.