💰 Expense Tracker - Final Project Requirements

🎯 Project Overview

Welcome to your final JavaScript project! You have been provided with a **complete working expense tracker** that currently uses **localStorage** to store data. Your mission is to convert this application to use a **real API backend** instead of localStorage.

This project will test everything you've learned in the bootcamp:

- DOM Manipulation
- Event Listeners
- Functions
- Fetch API
- Local Storage (which you'll replace)
- Async/Await
- Error Handling

📋 What You Have (Template)

You are provided with three files:

1. **index.html** - The HTML structure
2. **styles.css** - Basic styling
3. **script.js** - JavaScript with localStorage implementation

Current Features (Using localStorage):

- ✅ Add new expenses
- ✅ View all expenses with pagination
- ✅ Edit existing expenses
- ✅ Delete expenses (with confirmation)
- ✅ Search expenses by name
- ✅ Calculate total expenses
- ✅ Pagination controls

🎯 Your Mission

REQUIRED TASKS (Must Complete to Pass)

1. Convert to API Integration (70 points)

Replace all localStorage operations with fetch() API calls.

API Base URL:

```
https://pennypath-server.vercel.app
```

API Endpoints:

```
GET    /api/v1/expenses                Get all expenses (supports pagination and search)
GET    /api/v1/expenses/:expenseId     Get single expense by ID
POST   /api/v1/expenses                Create new expense
PUT    /api/v1/expenses/:expenseId     Update expense by ID
DELETE /api/v1/expenses/:expenseId     Delete expense by ID
```

Query Parameters for GET /api/v1/expenses:

- `page` - Page number (default: 1)
- `limit` - Items per page (default: 10)

- `search` - Search expenses by name (optional)

Request/Response Format:

All requests and responses use JSON format.

Expense Object Structure:

```
{
  "id": "generated-by-server",
  "name": "Grocery Shopping",
  "date": "2025-01-01",
  "amount": 150.50,
  "description": "Weekly groceries" // Optional
}
```

2. Implement Error Handling (10 points)

- Add try/catch blocks for all API calls
- Display user-friendly error messages using `alert()`
- Check `response.ok` before parsing JSON

3. Update UI After API Calls (10 points)

- Refresh the expense list after create/update/delete operations
- Handle empty states properly

4. Remove localStorage Code (5 points)

- Delete all localStorage-related functions
- Remove `saveExpensesToLocalStorage()` and `loadExpensesFromLocalStorage()`
- Clean up unused code

5. Test All Features (5 points)

- Verify all CRUD operations work correctly
- Test pagination with different limit values

- Test search functionality
- Test edge cases (empty data, errors, etc.)

🌟 BONUS TASKS (For Golden Certificate Candidates)

UI/UX Improvements (Up to 50 bonus points)

The current design is intentionally basic. Your task is to make it beautiful and professional:

- Redesign the UI to make it modern and visually appealing
- Add smooth animations and transitions
- Improve mobile responsiveness
- Add custom color schemes or themes
- Add icons and better visual feedback
- Create a professional, polished look

This is your chance to stand out! The students with the best-designed projects will receive the Golden Certificate.

🔍 What You Need to Research

To complete this project, you should research and understand:

1. Fetch API

- How to make GET, POST, PUT, DELETE requests
- Setting headers (`Content-Type: application/json`)
- Sending JSON data in request body
- Reading JSON responses

Search Terms:

- "JavaScript fetch API tutorial"
- "fetch POST request with JSON"
- "fetch API CRUD operations"

2. Async/Await

- How to use async/await with fetch
- Error handling with try/catch
- Waiting for API responses

Search Terms:

- "async await JavaScript"
- "fetch with async await"
- "JavaScript async error handling"

3. URL Query Parameters

- How to add pagination and search parameters to URLs
- Understanding `page`, `limit`, and `search` parameters

Search Terms:

- "URL query parameters JavaScript"
- "fetch API with query parameters"
- "JavaScript pagination with API"

4. HTTP Status Codes

- Understanding 200, 201, 404, 500 status codes
- Checking response.ok

Search Terms:

- "HTTP status codes"

- "fetch API check response status"

✅ Submission Checklist

Before submitting your project, make sure you have:

- [ ] Replaced all localStorage code with fetch API calls
- [ ] Implemented all 5 API endpoints correctly
- [ ] Added proper error handling with try/catch
- [ ] Tested create, read, update, delete operations
- [ ] Verified pagination works correctly with different limits
- [ ] Tested search functionality
- [ ] Removed unused localStorage functions
- [ ] Updated comments in your code
- [ ] Tested the app thoroughly
- [ ] (Bonus) Improved the UI/UX design

🏆 Grading Criteria

Category	Points	Description
API Integration	70	All CRUD operations using fetch API
Error Handling	10	Try/catch blocks and user feedback
UI Updates	10	Proper refresh and state management
Code Quality	5	Clean code, removed localStorage
Testing	5	All features work correctly
UI/UX (Bonus)	50	Beautiful design and animations
TOTAL	100	(+50 bonus points possible)

Grading Scale:

- **95-100+**: Golden Certificate 🏆
- **85-94**: Excellent
- **75-84**: Very Good

- **65-74**: Good
- **Below 65**: Needs Improvement

💡 Tips for Success

1. **Start Simple**: First, get ONE operation working (e.g., GET all expenses)
2. **Test Often**: Test each function as you convert it
3. **Console.log Everything**: Use console.log to see API responses
4. **Read Error Messages**: They tell you what's wrong!
5. **Use Browser DevTools**: Check the Network tab to see API calls
6. **Ask Questions**: Don't hesitate to ask when stuck
7. **Be Creative**: Make the UI yours - stand out!

🎨 Design Inspiration (For Golden Certificate)

To make your project exceptional:

- Look at popular expense trackers for UI ideas
- Use modern color palettes (coolors.co)
- Add smooth transitions and animations
- Make it responsive and mobile-friendly
- Add meaningful icons (Font Awesome, Material Icons)
- Create a professional, polished look

Remember: The template design is intentionally basic. Students with the most creative and professional designs will receive the Golden Certificate!

⚠️ Common Mistakes to Avoid

1. ❌ Forgetting to add `Content-Type` header for POST/PUT
2. ❌ Not checking `response.ok` before parsing JSON
3. ❌ Forgetting `async/await` or `.then()`
4. ❌ Not handling errors with try/catch
5. ❌ Hardcoding page/limit values instead of using variables
6. ❌ Not updating UI after API operations
7. ❌ Leaving localStorage code in the final version
8. ❌ Not testing pagination and search together

📚 Recommended Resources

- MDN Web Docs: Fetch API
- MDN Web Docs: Async/Await
- JavaScript.info: Network Requests
- YouTube: "JavaScript Fetch API Tutorial"
- YouTube: "Async Await Explained"

🚀 Getting Started Steps

1. **Set up your files**: Copy the three template files
2. **Test the template**: Make sure it works with localStorage
3. **Start with GET**: Convert the `getAllExpenses()` function first
4. **Test and verify**: Make sure expenses display correctly
5. **Move to POST**: Convert `createExpense()` next
6. **Continue**: Convert UPDATE and DELETE
7. **Add search**: Update GET to include search parameter

8. **Clean up**: Remove localStorage code
9. **Improve**: Add your creative touch to the design!

📞 Support

If you get stuck:

- Review the bootcamp materials on fetch and async/await
- Check the browser console for errors
- Use the Network tab in DevTools
- Ask your instructor for help
- Discuss with classmates (but write your own code!)

🎓 Final Note

This project is your opportunity to demonstrate everything you've learned. The template is functional, but **YOUR** version should be better! Show your creativity, problem-solving skills, and attention to detail.

The students who create the most professional and beautiful applications will receive the **Golden Certificate** 🏆

Good luck, and happy coding! 🚀

JavaScript Bootcamp - Final Project Assessment