FIT5032 Assessed Lab 8 Submission
Firestore Database Integration

Student Name: [Du Daoan]
Student ID: [35523166]

1 EFOLIO TASK 8.1 - Basic Firestore Integration

1.1 Screenshot Set 1: Add Book Implementation

1.1.1 Browser View

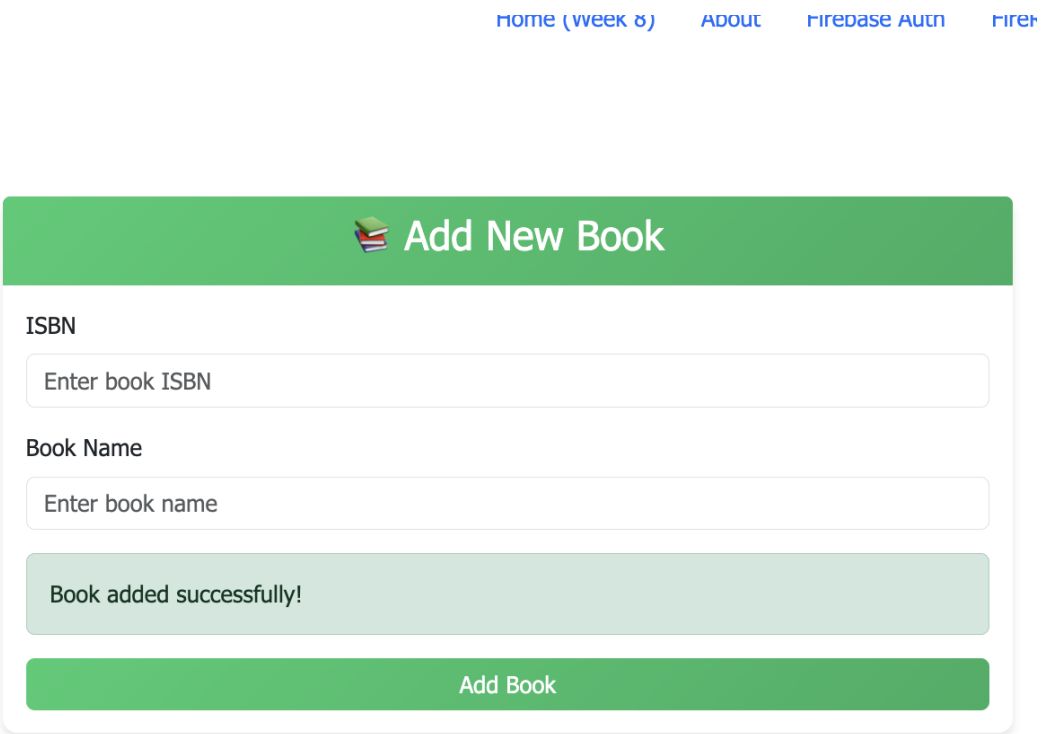

Figure 1: AddBookView.vue page showing the form and book list

Required: Screenshot showing the AddBook page with form and book list components.

1.1.2 Visual Studio Code Implementation

Required: Screenshot showing the AddBookView.vue code implementation.

1.2 Screenshot Set 2: Firestore Database

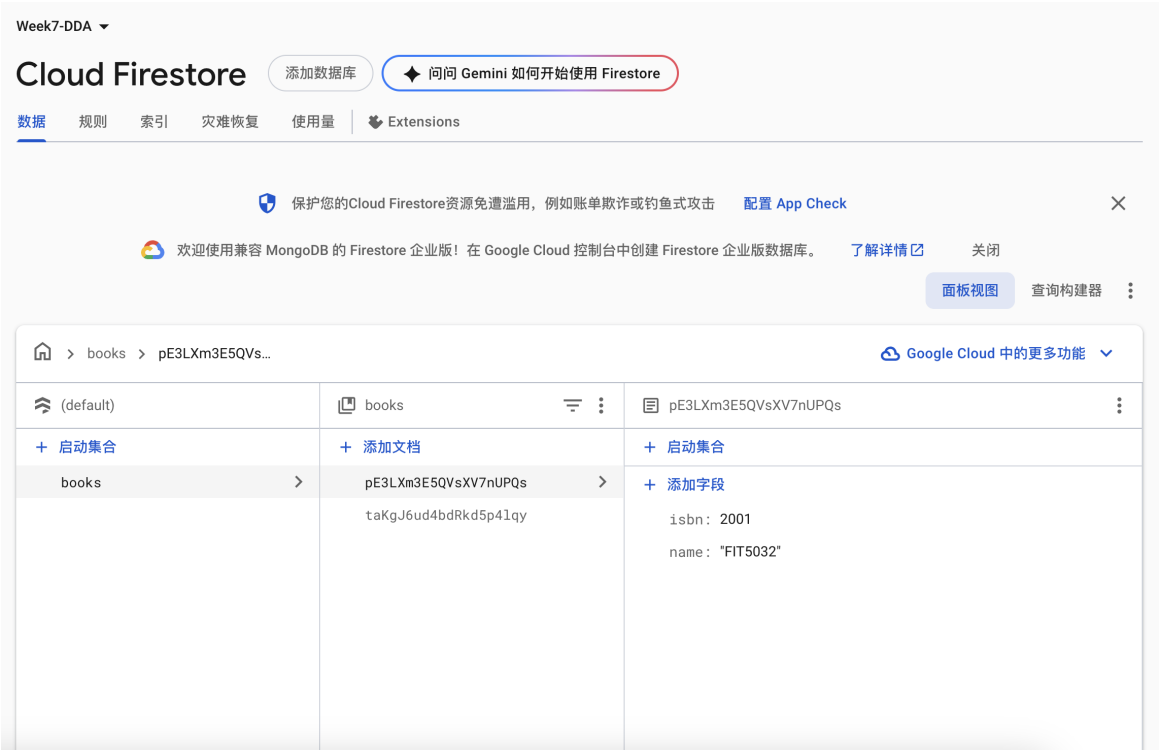

Figure 2: Firestore console showing added book data

Required: Screenshot of Firestore console showing the books collection with added data.

2 EFOLIO TASK 8.2 - Advanced Firestore Operations

2.1 Screenshot Set 1: Update and Delete Operations

2.1.1 Update Operation

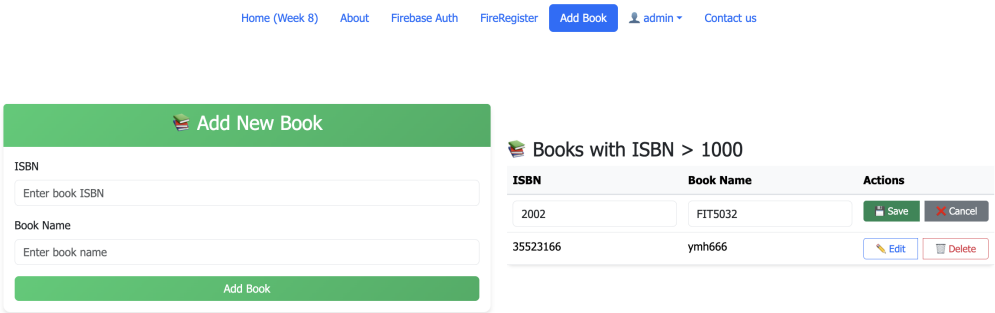

Figure 3: Book update functionality in browser

```
src > components > BookList.vue > {} script setup
<script setup>
const fetchBooks = async () => {
    loading.value = false
  }
}

// Edit functions
const startEdit = (book) => {
  editingBook.value = { ...book }
}

const cancelEdit = () => {
  editingBook.value = null
}
  ⌘L to chat, ⌘K to generate
const saveEdit = async () => {
  try {
    if (!editingBook.value) return

    const bookRef = doc(db, "books", editingBook.value.id)
    await updateDoc(bookRef, {
      isbn: editingBook.value.isbn,
      name: editingBook.value.name
    })

    console.log('🔥 Book updated successfully')
    await fetchBooks()
    editingBook.value = null
  } catch (err) {
    console.error('Error updating book:', err)
    error.value = 'Failed to update book: ' + err.message
  }
}

// Delete functions
const confirmDelete = (book) => {
  bookToDelete.value = book
  const modal = new Modal(deleteModal.value)
  modal.show()
}
                                        Review next file >
```

Figure 4: Update implementation in VS Code

2.1.2 Delete Operation

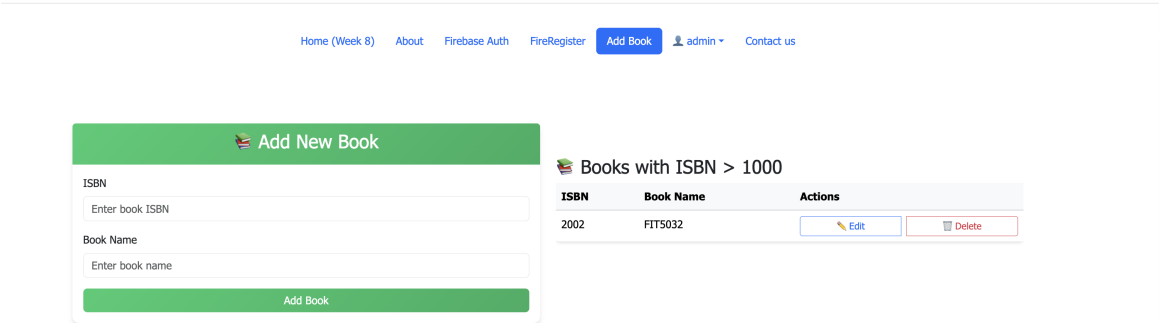

Figure 5: Book deletion functionality in browser

src > components > BookList.vue > {} script setup

```
<script setup>
}

// Delete functions
const confirmDelete = (book) => {
  bookToDelete.value = book
  const modal = new Modal(deleteModal.value)
  modal.show()
}

const deleteBook = async () => {
  try {
    if (!bookToDelete.value) return

    const bookRef = doc(db, "books", bookToDelete.value.id)
    await deleteDoc(bookRef)

    console.log('🔥 Book deleted successfully')
    const modal = Modal.getInstance(deleteModal.value)
    modal.hide()
    await fetchBooks()
    bookToDelete.value = null
  } catch (err) {
    console.error('Error deleting book:', err)
    error.value = 'Failed to delete book: ' + err.message
  }
}

// Fetch books when component is mounted
onMounted(fetchBooks)

// Expose the fetchBooks method to parent components
defineExpose({ fetchBooks })
</script>

<style scoped>
.table {
  background-color: white;
  border-radius: 8px;
  box-shadow: 0 2px 4px rgba(0, 0, 0,
}
```

Review next file >

Figure 6: Delete implementation in VS Code

2.2 Screenshot Set 2: Advanced Queries

2.2.1 Query Implementation

```
const fetchBooks = async () => {
  try {
    loading.value = true
    error.value = null

    // Build query constraints
    const constraints = []

    // ISBN range filter
    if (queryParams.value.isbnRange === 'over1000') {
      constraints.push(where('isbn', '>', 1000))
    } else if (queryParams.value.isbnRange === 'over5000') {
      constraints.push(where('isbn', '>', 5000))
    }

    // Sort order
    constraints.push(orderBy(
      queryParams.value.sortBy,
      queryParams.value.sortDir
    ))

    // Result limit
    if (queryParams.value.limit) {
      constraints.push(limit(queryParams.value.limit))
    }

    // Create and execute query
    console.log('🔍 Executing query with constraints:', constraints)
    const q = query(collection(db, "books"), ...constraints)
    const querySnapshot = await getDocs(q)

    books.value = querySnapshot.docs.map
      id: doc.id,
```

Review next file

Figure 7: Implementation of where, orderBy, and limit queries

2.2.2 Query Results

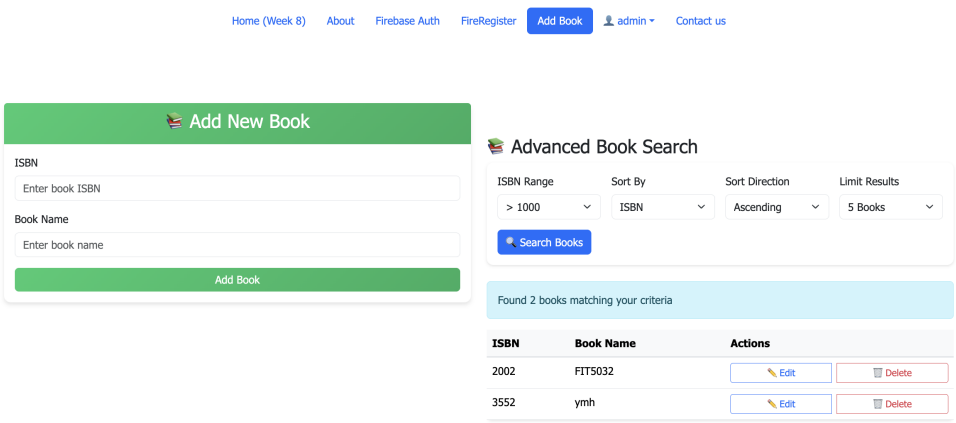

Figure 8: Query results displayed in browser

3 Technical Achievement Summary

This implementation demonstrates:

- **Firestore Integration:** Complete setup with Vue.js
- **CRUD Operations:** Create, Read, Update, Delete functionality
- **Advanced Queries:** Using where, orderBy, and limit
- **Real-time Updates:** Automatic UI updates on data changes
- **Component Architecture:** Reusable BookList component
- **Form Validation:** Required fields and type checking
- **Error Handling:** Proper error messages and loading states