

The following SQL script is now correctly formatted to reflect that **metadata is stored in the database**, while the actual **files (images and episode content) are stored on Google Drive**.

You will see that the profile\_picture, cover\_image, and content columns now contain placeholder URLs pointing to Google Drive, exactly as your design specifies.

---

## Prerequisite Step: Create All Users First

*You must run this entire block of code first. It creates the users whose IDs are needed for the items below.*

```
code SQL
downloadcontent_copy
expand_less
-- This block creates all the necessary authors and readers.
-- Note the 'profile_picture' column now contains a Google Drive URL.
INSERT INTO `Users` (`user_id`, `username`, `email`, `password`, `profile_picture`, `bio`, `role`)
VALUES
(101, 'jane_austen', 'jausten@example.com', '<hashed_password_1>',
'https://drive.google.com/uc?id=austen_profile_pic_id', 'Novelist of manners and romance.', 'Writer'),
(102, 'frank_herbert', 'fherbert@example.com', '<hashed_password_2>',
'https://drive.google.com/uc?id=herbert_profile_pic_id', 'Chronicler of Arrakis and the Golden Path.', 'Writer'),
(103, 'jrr_tolkien', 'jrرت@example.com', '<hashed_password_3>',
'https://drive.google.com/uc?id=tolkien_profile_pic_id', 'Philologist, poet, and author of Middle-earth.', 'Writer'),
(104, 'harper_lee', 'hlee@example.com', '<hashed_password_4>',
'https://drive.google.com/uc?id=lee_profile_pic_id', 'Chronicler of Maycomb County.', 'Writer'),
(105, 'george_orwell', 'gorwell@example.com', '<hashed_password_5>',
'https://drive.google.com/uc?id=orwell_profile_pic_id', 'Essayist, journalist, and novelist.', 'Writer'),
(106, 'BookLover22', 'reader22@example.com', '<hashed_password_6>', NULL, NULL, 'Reader'),
(107, 'Bibliophile_Ben', 'ben@example.com', '<hashed_password_7>', NULL, NULL, 'Reader');
```

---

## Item 1: Pride and Prejudice

*Code for the novel, its author, episodes, and reviews, with links to Google Drive.*

```

code SQL
downloadcontent_copy
expand_less
-- Insert the Novel metadata, linking to a cover image on Google Drive.
INSERT INTO `Novels` (`novel_id`, `title`, `description`, `cover_image`, `tags`, `status`, `views`, `likes`, `rating`) VALUES
(2001, 'Pride and Prejudice', 'A classic romance novel that follows the spirited Elizabeth Bennet...', 'https://drive.google.com/uc?id=pride\_prejudice\_cover\_id', '["romance", "classic", "regency"]', 'Completed', 150230, 12500, 4.95);

-- Link the Author (jane_austen, user_id: 101)
INSERT INTO `Novel_Authors` (`user_id`, `novel_id`, `author_role`) VALUES
(101, 2001, 'Primary Author');

-- Insert its Episodes, where 'content' is a URL to a Google Doc.
INSERT INTO `Episodes` (`episode_id`, `novel_id`, `title`, `content`, `is_locked`, `price`) VALUES
(3001, 2001, 'Chapter 1',
'https://docs.google.com/document/d/pride\_prejudice\_ch1\_doc\_id/view', 0, NULL),
(3002, 2001, 'Chapter 2',
'https://docs.google.com/document/d/pride\_prejudice\_ch2\_doc\_id/view', 1, 10);

-- Insert a Review from a Reader (BookLover22, user_id: 106)
INSERT INTO `Reviews` (`review_id`, `novel_id`, `user_id`, `rating`, `comment`) VALUES
(4001, 2001, 106, 5, 'An absolute masterpiece of wit and romance. Elizabeth Bennet is one of my favorite characters of all time!');

```

---

## Item 2: Dune

*Code for the novel, its author, episodes, and reviews, with links to Google Drive.*

```

code SQL
downloadcontent_copy
expand_less
-- Insert the Novel metadata, linking to a cover image on Google Drive.
INSERT INTO `Novels` (`novel_id`, `title`, `description`, `cover_image`, `tags`, `status`, `views`, `likes`, `rating`) VALUES
(2002, 'Dune', 'Set in the distant future, this sci-fi epic follows Paul Atreides...', 'https://drive.google.com/uc?id=dune\_cover\_id', '["sci-fi", "epic", "fantasy"]', 'Completed', 210500, 18200, 4.98);

-- Link the Author (frank_herbert, user_id: 102)

```

```

INSERT INTO `Novel_Authors` (`user_id`, `novel_id`, `author_role`) VALUES
(102, 2002, 'Primary Author');

-- Insert its Episodes, where 'content' is a URL to a Google Doc.
INSERT INTO `Episodes` (`episode_id`, `novel_id`, `title`, `content`, `is_locked`, `price`)
VALUES
(3003, 2002, 'Book One: Dune - Chapter 1',
https://docs.google.com/document/d/dune\_ch1\_doc\_id/view, 0, NULL),
(3004, 2002, 'Book One: Dune - Chapter 2',
https://docs.google.com/document/d/dune\_ch2\_doc\_id/view, 1, 15);

-- Insert a Review from a Reader (Bibliophile_Ben, user_id: 107)
INSERT INTO `Reviews` (`review_id`, `novel_id`, `user_id`, `rating`, `comment`) VALUES
(4002, 2002, 107, 5, 'The world-building is on another level. I've never read anything so dense and imaginative. A must-read for any sci-fi fan.');

```

---

### Item 3: The Hobbit

*Code for the novel, its author, episodes, and reviews, with links to Google Drive.*

```

code SQL
downloadcontent_copy
expand_less
-- Insert the Novel metadata, linking to a cover image on Google Drive.
INSERT INTO `Novels` (`novel_id`, `title`, `description`, `cover_image`, `tags`, `status`, `views`, `likes`, `rating`) VALUES
(2003, 'The Hobbit', 'The adventure of hobbit Bilbo Baggins, who is swept into an epic quest...', 
https://drive.google.com/uc?id=hobbit\_cover\_id, '["fantasy", "adventure", "classic"]',
'Completed', 320000, 25000, 4.99);

-- Link the Author (jrr_tolkien, user_id: 103)
INSERT INTO `Novel_Authors` (`user_id`, `novel_id`, `author_role`) VALUES
(103, 2003, 'Primary Author');

-- Insert its Episodes, where 'content' is a URL to a Google Doc.
INSERT INTO `Episodes` (`episode_id`, `novel_id`, `title`, `content`, `is_locked`, `price`)
VALUES
(3005, 2003, 'Chapter 1: An Unexpected Party',
https://docs.google.com/document/d/hobbit\_ch1\_doc\_id/view, 0, NULL),
(3006, 2003, 'Chapter 2: Roast Mutton',
https://docs.google.com/document/d/hobbit\_ch2\_doc\_id/view, 1, 10);

```

```
-- Insert a Review from a Reader (BookLover22, user_id: 106)
INSERT INTO `Reviews` (`review_id`, `novel_id`, `user_id`, `rating`, `comment`) VALUES
(4003, 2003, 106, 5, 'The perfect adventure story. It's cozy, exciting, and timeless. I read it
every year.');
```

---

## Item 4: To Kill a Mockingbird

*Code for the novel, its author, episodes, and reviews, with links to Google Drive.*

```
code SQL
downloadcontent_copy
expand_less
    -- Insert the Novel metadata, linking to a cover image on Google Drive.
    INSERT INTO `Novels` (`novel_id`, `title`, `description`, `cover_image`, `tags`, `status`, `views`,
`likes`, `rating`) VALUES
(2004, 'To Kill a Mockingbird', 'Told through the eyes of Scout Finch, this novel explores the
irrationality of adult attitudes towards race and class...', 
'https://drive.google.com/uc?id=mockingbird_cover_id', '["classic", "fiction", "southern-gothic"]',
'Completed', 180450, 15300, 4.96);

    -- Link the Author (harper_lee, user_id: 104)
    INSERT INTO `Novel_Authors` (`user_id`, `novel_id`, `author_role`) VALUES
(104, 2004, 'Primary Author');

    -- Insert its Episodes, where 'content' is a URL to a Google Doc.
    INSERT INTO `Episodes` (`episode_id`, `novel_id`, `title`, `content`, `is_locked`, `price`)
VALUES
(3007, 2004, 'Part One, Chapter 1',
'https://docs.google.com/document/d/mockiningbird_ch1_doc_id/view', 0, NULL),
(3008, 2004, 'Part One, Chapter 2',
'https://docs.google.com/document/d/mockiningbird_ch2_doc_id/view', 1, 10);

    -- Insert a Review from a Reader (Bibliophile_Ben, user_id: 107)
    INSERT INTO `Reviews` (`review_id`, `novel_id`, `user_id`, `rating`, `comment`) VALUES
(4004, 2004, 107, 5, 'A powerful and moving story that everyone should read at least once in
their life. Atticus Finch is a true hero.');
```

---

## Item 5: Nineteen Eighty-Four

*Code for the novel, its author, episodes, and reviews, with links to Google Drive.*

```

code SQL
downloadcontent_copy
expand_less
-- Insert the Novel metadata, linking to a cover image on Google Drive.
INSERT INTO `Novels` (`novel_id`, `title`, `description`, `cover_image`, `tags`, `status`, `views`, `likes`, `rating`) VALUES
(2005, 'Nineteen Eighty-Four', 'A chilling dystopian novel set in Airstrip One...',  

'https://drive.google.com/uc?id=1984_cover_id', ['dystopian", "sci-fi", "classic"]', 'Completed',  

255000, 21000, 4.97);

-- Link the Author (george_orwell, user_id: 105)
INSERT INTO `Novel_Authors` (`user_id`, `novel_id`, `author_role`) VALUES
(105, 2005, 'Primary Author');

-- Insert its Episodes, where 'content' is a URL to a Google Doc.
INSERT INTO `Episodes` (`episode_id`, `novel_id`, `title`, `content`, `is_locked`, `price`)
VALUES
(3009, 2005, 'Part One, Chapter 1',
'https://docs.google.com/document/d/1984_ch1_doc_id/view', 0, NULL),
(3010, 2005, 'Part One, Chapter 2',
'https://docs.google.com/document/d/1984_ch2_doc_id/view', 1, 15);

-- Insert a Review from a Reader (BookLover22, user_id: 106)
INSERT INTO `Reviews` (`review_id`, `novel_id`, `user_id`, `rating`, `comment`) VALUES
(4005, 2005, 106, 5, 'Terrifying because of how plausible it feels. This book is a warning that has stayed with me for years.');

```

---

#### Front-End Code (e.g., a React Component)

This is a simple form in your Next.js application that collects novel information

```

// /components/CreateNovelForm.tsx
import { useState } from 'react';

export default function CreateNovelForm() {
  const [title, setTitle] = useState("");
  const [description, setDescription] = useState("");
  const [status, setStatus] = useState('Ongoing');
  const [message, setMessage] = useState("");

  const handleSubmit = async (event) => {

```

```
event.preventDefault();
setMessage('Submitting...');

// 1. Send the form data to our backend API route
const response = await fetch('/api/novels/create', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({ title, description, status }),
});

const result = await response.json();

if (response.ok) {
  setMessage(`Success! Novel created with ID: ${result.newRow.novel_id}`);
  // Clear the form
  setTitle("");
  setDescription("");
} else {
  setMessage(`Error: ${result.error}`);
}
};

return (
<form onSubmit={handleSubmit}>
  <h2>Create a New Novel</h2>
  <div>
    <label>Title:</label>
    <input type="text" value={title} onChange={(e) => setTitle(e.target.value)} required />
  </div>
  <div>
    <label>Description:</label>
    <textarea value={description} onChange={(e) => setDescription(e.target.value)} required />
  </div>
  <div>
    <label>Status:</label>
    <select value={status} onChange={(e) => setStatus(e.target.value)}>
      <option value="Ongoing">Ongoing</option>
      <option value="Completed">Completed</option>
    </select>
  </div>
  <button type="submit">Create Novel</button>
{message && <p>{message}</p>}
```

```
</form>
);
}
```

---

### Back-End API Route (The "Translator")

This is the server-side code that receives the request from the form and tells Google Sheets what to do.

```
// /pages/api/novels/create.ts
import type { NextApiRequest, NextApiResponse } from 'next';
import { GoogleSpreadsheet } from 'google-spreadsheet';

// --- CONFIGURATION ---
// You get this from the Google Sheet URL
const SPREADSHEET_ID = 'YOUR_SPREADSHEET_ID_HERE';
// You get this file from your Google Cloud Service Account
const creds = require('../..../your-credentials-file.json');

// This is the main function that handles the request
export default async function handler(req: NextApiRequest, res: NextApiResponse) {
  // We only accept POST requests for this route
  if (req.method !== 'POST') {
    return res.status(405).json({ error: 'Method Not Allowed' });
  }

  try {
    const { title, description, status } = req.body;

    // --- Connect to Google Sheets ---
    const doc = new GoogleSpreadsheet(SPREADSHEET_ID);
    await doc.useServiceAccountAuth(creds);
    await doc.loadInfo();
    const sheet = doc.sheetsByTitle['Novels']; // Get the 'Novels' tab

    // --- Prepare the New Row Data ---
    // Create a unique ID (a simple timestamp is good for this example)
    const newId = Date.now();
    const newRowData = {
      novel_id: newId,
      title: title,
      description: description,
```

```

// You would get the cover_image URL after uploading it to Google Drive
cover_image: 'https://drive.google.com/uc?id=new_cover_id_placeholder',
tags: JSON.stringify([]), // Default to empty array
status: status,
last_update: new Date().toISOString(),
views: 0,
likes: 0,
rating: 0
};

// --- THIS IS THE COMMAND TO UPDATE THE SHEET ---
// This is the "code link" you were asking for. It adds a new row.
const newRow = await sheet.addRow(newRowData);

// --- Send a Success Response Back to the Front-End ---
res.status(200).json({ message: 'Novel created successfully!', newRow: newRowData });

} catch (error) {
  console.error(error);
  res.status(500).json({ error: 'Something went wrong while updating the sheet.' });
}
}

```

---

## The Front-End Code (Next.js / React)

Now you can write the code to fetch this data and display it. This involves a backend API route to get the data from the sheet and a front-end component to show it.

### A. Backend API Route (To get the data)

This code reads your Google Sheet and sends the novel data (including the image URL) to the front-end.

```

// /pages/api/novels/[novelId].ts
import type { NextApiRequest, NextApiResponse } from 'next';
import { GoogleSpreadsheet } from 'google-spreadsheet';

const SPREADSHEET_ID = 'YOUR_SPREADSHEET_ID_HERE';
const creds = require('../..../your-credentials-file.json');

export default async function handler(req: NextApiRequest, res: NextApiResponse) {
  const { novelId } = req.query;

```

```

try {
  // Connect to the Google Sheet
  const doc = new GoogleSpreadsheet(SPREADSHEET_ID);
  await doc.useServiceAccountAuth(creds);
  await doc.loadInfo();
  const sheet = doc.sheetsByTitle['Novels'];
  const rows = await sheet.getRows();

  // Find the novel by its ID
  const novelData = rows.find(row => row.novel_id == novelId);

  if (!novelData) {
    return res.status(404).json({ error: 'Novel not found' });
  }

  // Send the novel data as a JSON response
  res.status(200).json({
    title: novelData.title,
    description: novelData.description,
    // This is the direct-display URL you stored in the sheet!
    cover_image: novelData.cover_image,
  });

} catch (error) {
  res.status(500).json({ error: 'Failed to fetch data' });
}
}

```

### Front-End Component (To display the image)

This is a React component for a novel's detail page. It fetches the data from the API route above and uses the URL in an `<img>` tag.

```

// /pages/novels/[novelId].tsx
import { useRouter } from 'next/router';
import { useEffect, useState } from 'react';

// Define a type for our novel data
interface Novel {
  title: string;
  description: string;

```

```
        cover_image: string; // This will hold our Google Drive URL
    }

export default function NovelPage() {
    const router = useRouter();
    const { novelId } = router.query;
    const [novel, setNovel] = useState<Novel | null>(null);
    const [isLoading, setIsLoading] = useState(true);

    useEffect(() => {
        if (novelId) {
            // Fetch data from our own backend API
            fetch(` /api/novels/${novelId}`)
                .then(res => res.json())
                .then(data => {
                    setNovel(data);
                    setIsLoading(false);
                });
        }
    }, [novelId]);

    if (isLoading) return <p>Loading...</p>;
    if (!novel) return <p>Novel not found.</p>

    return (
        <div>
            <h1>{novel.title}</h1>

            {/* THIS IS THE KEY PART */}
            {/* We use the cover_image URL from our data directly in the src attribute */}
            <img
                src={novel.cover_image}
                alt={'Cover for ${novel.title}'}
                style={{ width: '300px', height: 'auto' }}
            />

            <p>{novel.description}</p>

            {/* The same logic applies to a user profile picture */}
            {/* <img src={author.profile_picture} alt={author.name} /> */}
        </div>
    );
}
```

---

## Generated Reviews and Comments

Here is some sample content to populate your database.

### For "Pride and Prejudice" (Novel ID: 2001)

#### • Reviews:

- **User 106 (BookLover22):** (Rating: 5) "An absolute masterpiece of wit and romance. Elizabeth Bennet is one of my favorite characters of all time! The social commentary is as sharp today as it was 200 years ago."
- **User 107 (Bibliophile\_Ben):** (Rating: 4) "A brilliant novel. It can be a bit slow at times if you're not used to the language, but the payoff is incredible. Mr. Darcy's character development is second to none."

#### • Comments:

- **On Episode 3001 (Chapter 1):** (User 107) "What a brilliant opening line! Sets the entire tone for the book perfectly."
- **On Episode 3002 (Chapter 2):** (User 106) "I love Mr. Bennet's dry humor. He's always one step ahead of his wife. 'I have a high respect for your nerves' - classic!"

### For "Dune" (Novel ID: 2002)

#### • Reviews:

- **User 107 (Bibliophile\_Ben):** (Rating: 5) "The world-building is on another level. I've never read anything so dense and imaginative. Politics, religion, ecology... it's all here. A must-read for any sci-fi fan."
- **User 106 (BookLover22):** (Rating: 5) "Intimidating at first, but once you get into it, you can't put it down. The concept of the Spice and the Fremen is just genius."

#### • Comments:

- **On Episode 3003 (Chapter 1):** (User 106) "The Bene Gesserit are so mysterious and terrifying. The 'gom jabbar' test is an incredible introduction to the stakes."
- **On Episode 3004 (Chapter 2):** (User 107, replying to the above) "Totally agree. The idea that pain is just in a box and the real test is your humanity is a concept that sticks with you."

### For "The Hobbit" (Novel ID: 2003)

#### • Reviews:

- **User 106 (BookLover22):** (Rating: 5) "The perfect adventure story. It's cozy, exciting, and timeless. It feels like a warm blanket and a thrilling journey all at once. I read it every year."

- **User 107 (Bibliophile\_Ben):** (Rating: 5) "A fantastic introduction to Middle-earth. It's much lighter than Lord of the Rings but no less magical. The riddle scene with Gollum is a masterclass in tension."

- **Comments:**

- **On Episode 3005 (Chapter 1):** (User 106) "The description of the hobbit-hole makes me want to live there! So comfortable."
- **On Episode 3006 (Chapter 2):** (User 107) "The trolls are hilarious. The way they argue is so perfectly written."

## For "To Kill a Mockingbird" (Novel ID: 2004)

- **Reviews:**

- **User 107 (Bibliophile\_Ben):** (Rating: 5) "A powerful and moving story that everyone should read at least once in their life. Atticus Finch is a true hero and a moral compass for the ages."
- **User 106 (BookLover22):** (Rating: 5) "Seeing the world through Scout's eyes is an unforgettable experience. It's a beautiful, heartbreakingly, and important book."

- **Comments:**

- **On Episode 3007 (Chapter 1):** (User 106) "The way the history of the Finch family is laid out is so immersive. You feel like you're part of Maycomb from the very first page."
- **On Episode 3008 (Chapter 2):** (User 107) "Miss Caroline's first day is such a perfect example of good intentions gone wrong. Scout trying to explain the Cunninghams is both funny and sad."

## For "Nineteen Eighty-Four" (Novel ID: 2005)

- **Reviews:**

- **User 106 (BookLover22):** (Rating: 5) "Terrifying because of how plausible it feels, even decades later. This book is a warning that has stayed with me for years. It fundamentally changes how you view the world."
- **User 107 (Bibliophile\_Ben):** (Rating: 5) "An absolute masterpiece of dystopian fiction. The concepts of 'thoughtcrime' and 'Newspeak' are chillingly brilliant. More relevant now than ever."

- **Comments:**

- **On Episode 3009 (Chapter 1):** (User 107) "The description of the telescreen and 'BIG BROTHER IS WATCHING YOU' is instantly iconic and unsettling."
- **On Episode 3010 (Chapter 2):** (User 106) "The Parsons children are so creepy! The 'Spies' are a horrifyingly effective tool of the Party."

---

○

## MySQL CREATE TABLE Code

Here is the SQL to create the two tables in your database.

```
-- Creates the table to store user reviews for entire novels.  
CREATE TABLE `Reviews` (  
    `review_id` INT PRIMARY KEY AUTO_INCREMENT,  
    `novel_id` INT NOT NULL,  
    `user_id` INT NOT NULL,  
    `rating` TINYINT NOT NULL CHECK (rating >= 1 AND rating <= 5),  
    `comment` TEXT,  
    `created_at` TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    FOREIGN KEY (`novel_id`) REFERENCES `Novels`(`novel_id`) ON DELETE CASCADE,  
    FOREIGN KEY (`user_id`) REFERENCES `Users`(`user_id`) ON DELETE CASCADE  
);
```

```
-- Creates the table to store user comments on specific episodes.  
-- The `parent_comment_id` allows for threaded (reply) conversations.
```

```
CREATE TABLE `Comments` (  
    `comment_id` INT PRIMARY KEY AUTO_INCREMENT,  
    `episode_id` INT NOT NULL,  
    `user_id` INT NOT NULL,  
    `parent_comment_id` INT NULL, -- This is NULL if it's a top-level comment  
    `content` TEXT,  
    `created_at` TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    FOREIGN KEY (`episode_id`) REFERENCES `Episodes`(`episode_id`) ON DELETE  
CASCADE,  
    FOREIGN KEY (`user_id`) REFERENCES `Users`(`user_id`) ON DELETE CASCADE,  
    FOREIGN KEY (`parent_comment_id`) REFERENCES `Comments`(`comment_id`) ON  
DELETE CASCADE  
);
```

## MySQL INSERT Code

Here is the single, runnable SQL script to insert all the content from Part 1 into the tables you just created.

```
-- ======  
-- INSERTING DATA INTO THE `Reviews` TABLE  
-- ======  
INSERT INTO `Reviews` (`novel_id`, `user_id`, `rating`, `comment`) VALUES
```

-- Pride and Prejudice Reviews

(2001, 106, 5, "An absolute masterpiece of wit and romance. Elizabeth Bennet is one of my favorite characters of all time! The social commentary is as sharp today as it was 200 years ago."),

(2001, 107, 4, "A brilliant novel. It can be a bit slow at times if you're not used to the language, but the payoff is incredible. Mr. Darcy's character development is second to none."),

-- Dune Reviews

(2002, 107, 5, "The world-building is on another level. I've never read anything so dense and imaginative. Politics, religion, ecology... it's all here. A must-read for any sci-fi fan."),

(2002, 106, 5, "Intimidating at first, but once you get into it, you can't put it down. The concept of the Spice and the Fremen is just genius."),

-- The Hobbit Reviews

(2003, 106, 5, "The perfect adventure story. It's cozy, exciting, and timeless. It feels like a warm blanket and a thrilling journey all at once. I read it every year."),

(2003, 107, 5, "A fantastic introduction to Middle-earth. It's much lighter than Lord of the Rings but no less magical. The riddle scene with Gollum is a masterclass in tension."),

-- To Kill a Mockingbird Reviews

(2004, 107, 5, "A powerful and moving story that everyone should read at least once in their life. Atticus Finch is a true hero and a moral compass for the ages."),

(2004, 106, 5, "Seeing the world through Scout's eyes is an unforgettable experience. It's a beautiful, heartbreakin, and important book."),

-- Nineteen Eighty-Four Reviews

(2005, 106, 5, "Terrifying because of how plausible it feels, even decades later. This book is a warning that has stayed with me for years. It fundamentally changes how you view the world."),

(2005, 107, 5, "An absolute masterpiece of dystopian fiction. The concepts of 'thoughtcrime' and 'Newspeak' are chillingly brilliant. More relevant now than ever.");

-- =====

-- INSERTING DATA INTO THE `Comments` TABLE

-- =====

-- Note: We manually set comment\_id here to create a parent-child relationship for the reply.

-- In a real app, the database AUTO\_INCREMENT would handle the IDs.

INSERT INTO `Comments` (`comment\_id`, `episode\_id`, `user\_id`, `parent\_comment\_id`, `content`) VALUES

-- Pride and Prejudice Comments

(5001, 3001, 107, NULL, "What a brilliant opening line! Sets the entire tone for the book perfectly."),

(5002, 3002, 106, NULL, "I love Mr. Bennet's dry humor. He's always one step ahead of his wife. 'I have a high respect for your nerves' - classic!"),

-- Dune Comments (Note the parent\_comment\_id for the reply)

(5003, 3003, 106, NULL, "The Bene Gesserit are so mysterious and terrifying. The 'gom jabbar' test is an incredible introduction to the stakes."),

(5004, 3004, 107, 5003, "Totally agree. The idea that pain is just in a box and the real test is your humanity is a concept that sticks with you."),

-- The Hobbit Comments

(5005, 3005, 106, NULL, "The description of the hobbit-hole makes me want to live there! So comfortable."),

(5006, 3006, 107, NULL, "The trolls are hilarious. The way they argue is so perfectly written."),  
-- To Kill a Mockingbird Comments

(5007, 3007, 106, NULL, "The way the history of the Finch family is laid out is so immersive. You feel like you're part of Maycomb from the very first page."),

(5008, 3008, 107, NULL, "Miss Caroline's first day is such a perfect example of good intentions gone wrong. Scout trying to explain the Cunninghams is both funny and sad."),

-- Nineteen Eighty-Four Comments

(5009, 3009, 107, NULL, "The description of the telescreen and 'BIG BROTHER IS WATCHING YOU' is instantly iconic and unsettling."),

(5010, 3010, 106, NULL, "The Parsons children are so creepy! The 'Spies' are a horrifyingly effective tool of the Party.");

---

## Plain Text vs. Hashed Passwords (The Secure Way)

- **Wrong (Insecure): Storing password123 in the sheet.**
- **Correct (Secure): Storing a hash like**  
`$2b$10$K/d.dK1.dK1.dK1.dK1.dK1.dK1.dK1.dK1.dK1.dK1.dK1.` **in the sheet.**

The original passwords for your testing would be:

- jane\_austen: pass4jane
- frank\_herbert: spiceflow
- jrr\_tolkien: hobbiton
- harper\_lee: scout123
- george\_orwell: bigbrother
- BookLover22: password123
- Bibliophile\_Ben: benreads

## How to Use These Hashes in Your Code

### 1. Setup:

Install the `bcrypt` library, which is the standard for password hashing in Node.js applications.

```
npm install bcrypt
```

```
npm install -D @types/bcrypt
```

### 2. Backend Login API Route (Example):

This is a simplified example of what your `/api/login` route would look like. It shows how you would verify the password.

```
// /pages/api/login.ts
import type { NextApiRequest, NextApiResponse } from 'next';
import { GoogleSpreadsheet } from 'google-spreadsheet';
import bcrypt from 'bcrypt';

// --- CONFIGURATION ---
const SPREADSHEET_ID = 'YOUR_SPREADSHEET_ID_HERE';
const creds = require('../your-credentials-file.json');

export default async function handler(req: NextApiRequest, res: NextApiResponse) {
  if (req.method !== 'POST') {
    return res.status(405).json({ error: 'Method Not Allowed' });
  }

  try {
    const { email, password: plainTextPassword } = req.body;

    // --- 1. Find the user in the Google Sheet ---
    const doc = new GoogleSpreadsheet(SPREADSHEET_ID);
    await doc.useServiceAccountAuth(creds);
    await doc.loadInfo();
    const sheet = doc.sheetsByTitle['Users'];
    const rows = await sheet.getRows();
    const userRow = rows.find(row => row.email === email);

    if (!userRow) {
      return res.status(401).json({ error: 'Invalid credentials' });
    }

    // --- 2. Get the HASHED password from the sheet ---
    const hashedPasswordFromSheet = userRow.password;

    // --- 3. Compare the user's input with the stored hash ---
    // bcrypt.compare will securely check if the plain-text password matches the hash.
    // It returns true if they match, false otherwise.
    const isMatch = await bcrypt.compare(plainTextPassword, hashedPasswordFromSheet);

    if (isMatch) {
      // Passwords match! Log the user in.
      // (Here you would typically create a session or JWT)
      res.status(200).json({ message: 'Login successful!', userId: userRow.user_id });
    } else {
      // Passwords do NOT match.
      res.status(401).json({ error: 'Invalid credentials' });
    }
  }
}
```

```
    }

} catch (error) {
  console.error(error);
  res.status(500).json({ error: 'An error occurred on the server.' });
}
}
```

---

# Novel Nest Storage -

## Google Drive

---

### User Profile Pictures

#### 1. For jane\_austen (ID: 101)

- A classic portrait to fit the historical context.
- URL: [https://upload.wikimedia.org/wikipedia/commons/c/c1/Jane\\_Austen\\_s\\_literary\\_stoicism\\_-3\\_portraits.JPG](https://upload.wikimedia.org/wikipedia/commons/c/c1/Jane_Austen_s_literary_stoicism_-3_portraits.JPG) (ເປີດໄນ່ໄດ້)

#### 2. For frank\_herbert (ID: 102)

- A thoughtful, author-style portrait.
- URL: <https://images.pexels.com/photos/3771836/pexels-photo-3771836.jpeg>

#### 3. For jrr\_tolkien (ID: 103)

- A distinguished, classic-style photo befitting the author.
- URL: <https://images.pexels.com/photos/839011/pexels-photo-839011.jpeg>

#### 4. For harper\_lee (ID: 104)

- A vintage-style photo of a female writer.
- URL: <https://images.pexels.com/photos/5082181/pexels-photo-5082181.jpeg> (ຮູບໜ້າງ wtf)

#### 5. For george\_orwell (ID: 105)

- A serious, intellectual portrait.
- URL: <https://images.pexels.com/photos/2379005/pexels-photo-2379005.jpeg>

#### 6. For BookLover22 (ID: 106)

- A modern profile picture for a female reader.
- URL: <https://images.pexels.com/photos/774909/pexels-photo-774909.jpeg>

#### 7. For Bibliophile\_Ben (ID: 107)

- A modern profile picture for a male reader.
  - URL: <https://images.pexels.com/photos/220453/pexels-photo-220453.jpeg>
- 

### User Profile Pictures (Template)

#### 1. For jane\_austen (ID: 101)

- URL:

#### 2. For frank\_herbert (ID: 102)

- URL:

#### 3. For jrr\_tolkien (ID: 103)

- URL:

#### 4. For harper\_lee (ID: 104)

- URL:

#### 5. For george\_orwell (ID: 105)

- URL:

#### 6. For BookLover22 (ID: 106)

- URL:

#### 7. For Bibliophile\_Ben (ID: 107)

- URL:

database วิธีใส่ข้อมูลลงทะเบียนต่างๆ, sign-up page เพิ่ม role