Creating an entire virtual cinema platform involves several components and steps, including setting up the IBM Cloud Video Streaming service, creating a user interface, integrating payment gateways, and managing video content. Due to the complexity and length of such a project, it is not feasible to provide the entire implementation in one response. However, I can guide you on how to start setting up the IBM Cloud Video Streaming service using code.

Before you start, ensure you have an IBM Cloud account and the necessary permissions to create resources. Additionally, you need to install the required libraries. For this example, we'll use the Python programming language.

First, install the required package for IBM Cloud Video Streaming using pip:

```
pip install ibm-watson
```

Then, you can use the following Python code to get started with the IBM Cloud Video Streaming service:

```
from ibm_watson import IAMTokenManager

from ibm_watson import ApiException

from ibm_watson import IBMCredentials

from ibm_watson import IBMService

# Replace these variables with your IBM Cloud Video Streaming service credentials

api_key = 'YOUR_API_KEY'

url = 'YOUR_URL'

# Authenticate using IAMTokenManager

iam_token_manager = IAMTokenManager(

   apikey=api_key

)

# Set up the IBM Cloud Video Streaming Service

service = IBMService(

   authenticator=iam_token_manager

)

service.set_service_url(url)

# Now, you can start using the IBM Cloud Video Streaming service

# For example, you can create a channel or manage videos
# Example of creating a channel
try:
    response = service.create_channel(name='example_channel')
    print(response.get_result())
except ApiException as ex:
    print("Method failed with status code " + str(ex.code) + ": " + ex.message)
```

Make sure to replace `YOUR_API_KEY` and `YOUR_URL` with your actual IBM Cloud Video Streaming service credentials.

Platform Features:

1. User Authentication: Allow users to create accounts and log in to access the virtual cinema.
2. Video Catalog: Display a collection of available movies with details such as title, description, and genre.
3. Streaming Functionality: Enable users to stream movies directly from the platform.
4. User Profiles: Allow users to manage their profiles, view their watch history, and customize preferences.
5. Payment Integration: Incorporate a payment system to allow users to purchase or rent movies.
6. Search and Filter: Implement a search and filter function to help users find movies based on genres, actors, or directors.

Here's a simple example of an intuitive user interface for the virtual cinema platform using HTML and CSS:

```
<!DOCTYPE html>
<html>
<head>
    <title>Virtual Cinema</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
    <header>
        <h1>Welcome to Virtual Cinema</h1>
```

```html
        <nav>
            <ul>
                <li><a href="#">Home</a></li>
                <li><a href="#">Movies</a></li>
                <li><a href="#">Profile</a></li>
                <li><a href="#">Sign Out</a></li>
            </ul>
        </nav>
    </header>
    <section class="movies">
        <h2>Featured Movies</h2>
        <div class="movie">
            <img src="movie1.jpg" alt="Movie 1">
            <h3>Movie 1 Title</h3>
            <p>Description of the movie.</p>
            <button>Watch Now</button>
        </div>
        <!-- More movie entries here -->
    </section>
    <footer>
        <p>&copy; 2023 Virtual Cinema. All rights reserved.</p>
    </footer>
</body>
</html>
```

CSS (styles.css):

```css
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
}
```

```css
header {
   background-color: #333;
   color: #fff;
   padding: 1em;
   text-align: center;
}

nav ul {
   list-style-type: none;
   padding: 0;
}

nav ul li {
   display: inline;
   margin: 0 10px;
}

section.movies {
   padding: 20px;
}

.movie {
   display: inline-block;
   width: 30%;
   margin: 10px;
   text-align: center;
}

footer {
   background-color: #333;
   color: #fff;
```

```
  text-align: center;
  padding: 1em;
  position: fixed;
  bottom: 0;
  width: 100%;
}
```

To implement user registration and authentication mechanisms in your virtual cinema platform, you can use various technologies such as HTML, CSS, JavaScript, and a back-end framework like Node.js with Express. For user authentication, you can use libraries like Passport.js and bcrypt for secure password hashing. Here's a basic example of how you can set up user registration and authentication using Node.js and Express:

First, install the necessary dependencies:

```
npm install express body-parser express-session passport passport-local bcrypt
```

Create a file named `server.js` for the Node.js application:

```
const express = require('express');
const session = require('express-session');
const bodyParser = require('body-parser');
const passport = require('passport');
const LocalStrategy = require('passport-local').Strategy;
const bcrypt = require('bcrypt');
const app = express();

// Sample database, you should use a proper database in your application
const users = [];

// Middleware
app.use(bodyParser.urlencoded({ extended: true }));
app.use(session({ secret: 'secret', resave: false, saveUninitialized: false }));
app.use(passport.initialize());
app.use(passport.session());
```

```
// Passport local strategy
passport.use(new LocalStrategy((username, password, done) => {
  const user = users.find(user => user.username === username);
  if (!user) {
    return done(null, false, { message: 'Incorrect username.' });
  }
  if (!bcrypt.compareSync(password, user.password)) {
    return done(null, false, { message: 'Incorrect password.' });
  }
  return done(null, user);
}));

// Serialize and deserialize user
passport.serializeUser((user, done) => {
  done(null, user.username);
});

passport.deserializeUser((username, done) => {
  const user = users.find(user => user.username === username);
  done(null, user);
});

// Routes
app.get('/', (req, res) => {
  res.send('Welcome to Virtual Cinema!');
});

app.get('/login', (req, res) => {
  res.sendFile(__dirname + '/login.html');
});
```

```
app.post('/login', passport.authenticate('local', { successRedirect: '/', failureRedirect: '/login' }));


app.get('/register', (req, res) => {
  res.sendFile(__dirname + '/register.html');
});


app.post('/register', (req, res) => {
  const hashedPassword = bcrypt.hashSync(req.body.password, 10);
  users.push({ username: req.body.username, password: hashedPassword });
  res.redirect('/login');
});


// Server
const port = 3000;
app.listen(port, () => {
  console.log(`Server is running on http://localhost:${port}`);
});
```

Create `login.html` and `register.html` files in the same directory with appropriate form fields for login and registration.

`login.html`:

```
<!DOCTYPE html>
<html>
<head>
  <title>Login</title>
</head>
<body>
  <h2>Login</h2>
  <form action="/login" method="post">
    <div>
```

```
            <label>Username:</label>
            <input type="text" name="username" required>
        </div>
        <div>
            <label>Password:</label>
            <input type="password" name="password" required>
        </div>
        <div>
            <button type="submit">Login</button>
        </div>
    </form>
</body>
</html>
```

`register.html`:

```
<!DOCTYPE html>
<html>
<head>
    <title>Register</title>
</head>
<body>
    <h2>Register</h2>
    <form action="/register" method="post">
        <div>
            <label>Username:</label>
            <input type="text" name="username" required>
        </div>
        <div>
            <label>Password:</label>
            <input type="password" name="password" required>
        </div>
        <div>
            <button type="submit">Register</button>
        </div>
    </form>
</body>
</html>
```

You can further customize and enhance this basic authentication system to fit your specific requirements and integrate it into your virtual cinema platform.