Assignment 2: REST API Backend for Loyalty Program

Last Updated on Oct 26, 2025

- Oct 19: A utorid can be of 7 or 8 characters.
- Oct 20: Auto Tester is available + clarification on where to store the JWT_SECRET
- Oct 24: A reset token cannot be (successfully) consumed more than once and may be "overwritten".
- Oct 26: We should not update an event's capacity to unlimited if the event was created with a limited capacity.

Objective

In this assignment, you will build a REST API backend for a loyalty program using **Express.js** and **Prisma**. This loyalty program will enable users to accumulate points for purchases and redeem them for free items, similar to PC Optimum. The objective is to develop the **backend infrastructure** for the loyalty program, which will serve as the foundation for your term project.

Later in the term project, you and your teammates will add new features into the backend and build the frontend and deploy the web application.

Working version and Postman demo

On this postman workspace: https://www.panchen.ca/csc309-a2-demo, we've provided examples of few requests related to the users, which might be helpful.

In addition, we also deploy a working version of this backend at: https://a2-demo.panchen.ca. You may try a few API requests to get a better understanding of the expectations of the backend. **But please DO NOT give any sensitive personal information**. Everyone can get

the super user privileges with credentials **chenpan** and **ilovecsc309**. Also note that the database will be reset per hour.

Submission

Once you have tested your code and verified that it works according to specification and **committed it locally** (check that by running git status), you can push it back to MarkUs. We will collect and grade the last version pushed to MarkUs after the assignment deadline. Note that whatever you get from the autotester will be your final mark; therefore, you are recommended to run it before your final submission.

Link to submission:
https://markus.teach.cs.toronto.edu/markus/courses/91/assignments/802

Academic Integrity

If you have formed a project team, you may collaborate only with your teammates on Assignment 2. If you collaborate with your teammates, please document how you collaborated with them in `collaboration.txt`. If not, please indicate: “I did not collaborate with any other student”.

Even if you collaborate with other teammates, you must submit the A2 **individually** on MarkUs. You will not be allowed to seek advice from other students who are not your teammates or copy/paste someone else's code. However, you are allowed to look at online resources, tutorials, and Q&A websites to solve the problems. If you need to use any open-source codes from the internet, make sure that you have references for it.

If you use AI to assist you with this assignment, you must indicate how you used AI in `ai.txt`, or indicate: “I did not use any AI tool for this assignment”.

Overview

You will create a REST API that supports the following functionalities:

1. **User Roles**
    a. ***Regular Users (regular)***: Can accumulate and redeem points, view their points balance and transaction history, and manage their account.
    b. ***Cashiers (cashier)***: Can add transactions for users. The system can flag a cashier as suspicious when it detects anomalous activities.

c. ***Managers (manager)***: Responsible for verifying new user accounts and managing transactions, events, and promotions, in addition to performing cashier duties.
d. ***Superusers (superuser)***: Have full database access and all privileges, including those of managers and cashiers.
e. In your code, you should use the following strings to represent the user roles: regular, cashier, manager, and superuser.

2. **User Registration and Authentication**
    a. Cashiers can create an account for a User.
    b. Users can log in and log out.
    c. Users can update their account details and password.
    d. Users must activate their account upon registration.
    e. Users can reset their password if they forget it.
    f. Users cannot redeem points until their student information is verified by a Manager.
    g. Users can transfer their points to another user.
    h. Users cannot delete their accounts, to ensure accountability in case of misuse.
3. **User Management**
    a. Managers can promote a user to a cashier role or revoke their cashier status.
    b. Managers can clear a cashier who is flagged as suspicious.
    c. Superusers can promote a user to a manager role or demote them back to a regular user.
4. **Transaction Logging and Management**
    a. Once created, transactions cannot be deleted.
    b. The following types of transactions are supported:
        i. *Purchase*: created by cashiers on behalf of a customer during checkout. The dollar amount spent is entered, and the customer will receive, by default, 1 point for every 25 cents spent.
            1. Transactions created by a suspicious cashier will require verification by a manager before points are awarded.
        ii. *Adjustment*: created by managers to manually correct any errors (or misuse) relating to a previously made transaction.
        iii. *Redemption*: created by the customer themselves. Once created, a cashier will process the redemption at a rate of 1 cent per point redeemed. For example, if the customer redeems 100 points, then $1 will be taken off the subtotal of their purchase.
        iv. *Transfer*: created by a user to transfer points to another user.
        v. *Event*: created by an event organizer to award points to guests for their participation.
    c. Managers can view all transactions, for auditing purposes.
    d. Users can view their past transactions, including date, transaction total, points earned or redeemed, and any promotions applied.
5. **Point-Earning Events**
    a. Managers can create events and assign organizers to them.

    b. Events include details such as start and end times, description, location, capacity, and the ability to RSVP users.
    c. Managers allocate a set number of points for each event, which organizers can distribute to attendees.
    d. Only users who RSVP and have their attendance confirmed can receive points from the event.
    e. Organizers cannot also be guests at the same time, to prevent organizers from awarding points to themselves.
    f. Organizers can update all event details except for adding/removing organizers and deleting the event.
6. **Promotions**
    a. Managers can create promotional periods that offer increased points per amount spent, with an optional minimum spending requirement.
    b. Managers can create promotional offers, such as "buy one, get extra points." These offers are single-use for each user and must be manually applied by a cashier during checkout when the specified conditions are met.
    c. Managers can monitor and adjust promotions as needed.

Setup

Log in to MarkUs and go to CSC309. Then, navigate to the **A2** assignment. The starter files for this assignment is in **A2** folder:

- **.gitignore**: Specifies files and folders to be ignored, ensuring that files you shouldn't commit aren't checked in.
- **index.js**: The source file for your server.
- **package.json**: contains information about this package. Do not change.
    - You can run the `npm run clean` script to remove all generated files.
- **prisma/schema.prisma**: define your models here.
- **prisma/createsu.js**: a command line script that allows the system administrator to create the first Superuser account.

For this assignment, you may add any number of files that you want, as long as we can run `node index.js PORT` to start your backend server.

In addition to **express**, **prisma**, and **sqlite3**, the following packages are used for this assignment (they are already listed in package.json):

- **jsonwebtoken**: A library to generate and verify JSON Web Tokens (JWT) for authentication.
- **express-jwt**: Middleware for Express.js to validate JWTs and protect routes.

- **multer**: Middleware for handling multipart/form-data, primarily used for file uploads.
- **cors**: Middleware to enable Cross-Origin Resource Sharing (CORS) in Express.js applications.
- **uuid**: helps you create random uuids (for the reset token).
- **bcrypt**: (optional) allows you to encrypt user password.
- **dotenv:** (optional)

Please be reminded that you are not allowed to add or remove packages from this list. These are the exact set of packages you will use for this assignment.

Creating a JWT_SECRET

We will be loading **JWT_SECRET** from the environment variable. You should not push your own JWT_SECRET to the repo directly. You may load the JWT_SECRET from the environment variable using: **process.env.JWT_SECRET**. If you have a .env file, you can use the dotenv package to load it by **require('dotenv').config();**. (we have this package installed on the auto tester)

For this assignment, we're not checking explicitly whether a JWT_SECRET has been pushed or not, but for the project, pushing a JWT_SECRET might result in a small deduction.

Creating a Superuser

A superuser account is required to create and manage other users. It cannot be created through API endpoints since only cashiers or higher roles can create new users. Hence, we need a way to "seed" our database with a superuser.

Your first task is to complete a script named `createsu.js` in the `prisma` folder so that we can create superusers. The script should take 3 command line arguments: `utorid`, `email`, and `password` (in plaintext). Then, we should be able to create superusers with this command:

node prisma/createsu.js <utorid> <email> <password>

Note: Please ensure that your superuser is flagged as **verified, and all necessary fields are created**.

Note: you may also generate more data in seed.js.

API Specifications

In this section, we will detail the specifications of each API endpoint and the HTTP methods they support. Please adhere to the following guidelines for API descriptions:

- For GET requests, include the payload in the query parameters.
- For all other HTTP methods, send the payload in the request body as JSON.
- Error handling: Return an appropriate HTTP status code along with a JSON response in the following format: { "error": "The error message" }
    - Note that the autoester will not check the error message, so you can customize it however you like to improve error handling.
- If no specific status code is required, default to 200 OK for successful responses.
- The response body must always be in JSON format.
- For each endpoint + method, a minimum clearance is required, ranging from Any (does not require authentication), Regular, Cashier, Event Organizer, Manager, and Superuser.
- If the endpoint + method requires authentication, i.e., clearance is not Any, and the request is not authenticated, return **401 Unauthorized**.
- If the endpoint + method requires a clearance that the user does not have, return **403 Forbidden**.
- If the endpoint has an URL parameter that refers to a non-existent object, return **404 Not Found**.
- If the endpoint does not support a particular method, return **405 Method Not Allowed**.
- If the request body is invalid for any reason, e.g., missing field, extra field (i.e., field not specified in the API), value out of range, value is not of the correct type, etc., return **400 Bad Request**.
- You are suggested to autoincrementing integers as IDs for models, but it is not required (we will use whatever id that is returned to make subsequent requests in the tester).

/users

- **Method:** POST
- **Description:** Register a new user
- **Clearance**: Cashier or higher
- **Payload**:

Field	Required	Type	Description
utorid	Yes	string	Unique, Alphanumeric, 7-8 characters
name	Yes	string	1-50 characters
email	Yes	string	Unique, Valid University of Toronto email

- **Response**:

- **201 Created** on success

```
{
 "id": 1,
 "utorid": "johndoe1",
 "name": "John Doe",
 "email": "john.doe@mail.utoronto.ca",
 "verified": false,
 "expiresAt": "2025-03-10T01:41:47.000Z",
 "resetToken": "ad71d4e1-8614-46aa-b96f-cb894e346506"
}
```

- **409 Conflict** if the user with that utorid already exists

Upon account creation, ~~an email with an activation link will be sent to the provided email address~~ (see `POST /auth/resets/:resetId`). The activation link expires in 7 days, after which, the user can request for a password reset to attempt activation again.

For this assignment, **you are not expected to send emails**, so the response body also contains the `token` that can be used to activate the account.

- **Method:** GET
- **Description:** Retrieve a list of users
- **Clearance**: Manager or higher
- **Payload**:

Field	Required	Type	Description
name	No	string	Filter by utorid or name
role	No	string	Filter by user role
verified	No	boolean	Filter by verified status
activated	No	boolean	Filter by whether the user has ever logged in before
page	No	number	Page number for pagination (default is 1)
limit	No	number	Number of objects per page (default is 10)

- **Response**: `count`, which stores the total number of results (after applying all filters), and `results`, which contains a list of users

```
{
 "count": 51,
```

```
 "results": [
  {
   "id": 1,
   "utorid": "johndoe1",
   "name": "John Doe",
   "email": "john.doe@mail.utoronto.ca",
   "birthday": "2000-01-01",
   "role": "regular",
   "points": 0,
   "createdAt": "2025-02-22T00:00:00.000Z",
   "lastLogin": null,
   "verified": false,
   "avatarUrl": null
  },
  // More user objects...
 ]
}
```


/users/:userId

- **Method:** GET
- **Description:** Retrieve a specific user
- **Clearance**: Cashier or higher
- **Payload:** None
- **Response**:


```
{
 "id": 1,
 "utorid": "johndoe1",
 "name": "John Doe",
 "points": 0,
 "verified": false,
 "promotions": [
  { "id" : 2, "name" : "Buy a pack of Pepsi", "minSpending": null, "rate": null, "points":
20 }
 ]
}
```

Note that the cashier can only see limited information regarding the user. `promotions` should only show one-time promotions that are still available to the user, i.e., they have not used those promotions yet.

- **Method:** GET
- **Description:** Retrieve a specific user
- **Clearance**: Manager or higher
- **Payload:** None

Response:


```
{
 "id": 1,
 "utorid": "johndoe1",
 "name": "John Doe",
 "email": "john.doe@mail.utoronto.ca",
 "birthday": "2000-01-01",
 "role": "regular",
 "points": 0,
 "createdAt": "2025-02-22T00:00:00.000Z",
 "lastLogin": "2025-02-22T00:00:00.000Z",
 "verified": false,
 "avatarUrl": null,
 "promotions": [
   { "id" : 2, "name" : "Buy a pack of Pepsi", "minSpending": null, "rate": null, "points": 20 }
 ]
}
```


- **Method:** PATCH
- **Description:** Update a specific user's various statuses and some information
- **Clearance**: Manager or higher
- **Payload**:

Field	Required	Type	Description
email	No	string	In case it was entered incorrectly during registration

verified	No	boolean	Should always be set to true
suspicious	No	boolean	true or false
role	No	string	**As Manager:** Either "cashier" or "regular" **As Superuser:** Any of "regular", "cashier", "manager", or "superuser"

- **Response**: only the field(s) that were updated will be returned, e.g., when the `suspicious` field is updated:

```
{
 "id": 1,
 "utorid": "johndoe1",
 "name": "John Doe",
 "suspicious": true,
}
```

When promoting a user to a cashier, the initial value for suspicious must be false, meaning that a suspicious user can not be a cashier.

/users/me

- **Method:** PATCH
- **Description:** Update the current logged-in user's information
- **Clearance**: Regular or higher
- **Payload**:

Field	Required	Type	Description
name	No	string	1-50 characters
email	No	string	Unique, Valid UofT email
birthday	No	string	A date in the format of YYYY-MM-DD
avatar	No	file	Image file for the user's avatar

- **Response**:

```
{
 "id": 1,
 "utorid": "johndoe1",
 "name": "John Doe",
 "email": "john.doe@mail.utoronto.ca",
 "birthday": "2000-01-01",
 "role": "regular",
 "points": 0,
 "createdAt": "2025-02-22T00:00:00.000Z",
 "lastLogin": "2025-02-22T00:00:00.000Z",
 "verified": true,
 "avatarUrl": "/uploads/avatars/johndoe1.png"
}
```


- **Method:** GET
- **Description:** Retrieve the current logged-in user's information
- **Clearance**: Regular or higher
- **Payload**: None
- **Response**:


```
{
 "id": 1,
 "utorid": "johndoe1",
 "name": "John Doe",
 "email": "john.doe@mail.utoronto.ca",
 "birthday": "2000-01-01",
 "role": "regular",
 "points": 0,
 "createdAt": "2025-02-22T00:00:00.000Z",
 "lastLogin": "2025-02-22T00:00:00.000Z",
 "verified": true,
 "avatarUrl": "/uploads/avatars/johndoe1.png",
 "promotions": []
}
```

/users/me/password

- **Method:** PATCH
- **Description:** Update the current logged-in user's password
- **Clearance**: Regular or higher
- **Payload**:

Field	Required	Type	Description
old	Yes	string	The user's current password
new	Yes	string	8-20 characters, at least one uppercase, one lowercase, one number, one special character

- **Response**:
  - **200 OK** on success
  - **403 Forbidden** if the provided current password is incorrect

/auth/tokens

- **Method:** POST
- **Description:** Authenticate a user and generate a JWT token
- **Clearance**: Any
- **Payload**:

Field	Required	Type	Description
utorid	Yes	string	The utorid of a user
password	Yes	string	The password of the user with the specified utorid

- **Response**:

```
{
  "token": "jwt_token_here",
  "expiresAt": "2025-03-10T01:41:47.000Z"
}
```

/auth/resets

- **Method:** POST

- **Description:** Request a password reset email.
- **Clearance**: Any
- **Payload**:

Field	Required	Type	Description
utorid	Yes	string	The utorid of a user who forgot their password

- **Response**
  - **202 Accepted** on success

    ```
    {
     "expiresAt": "2025-03-01T01:41:47.000Z",
     "resetToken": "ad71d4e1-8614-46aa-b96f-cb894e346506"
    }
    ```

  - **429 Too Many Requests** if another request is made from the same IP address within 60 seconds. Hint: your rate limiter can be implemented completely in memory. You may not use **express-rate-limit**, since we do not allow you to install additional packages (if you do, the autotester will break).

If an account with the specified `utorid` exists, ~~an email with a password reset link will be sent to the user's email address~~ (see `POST /auth/resets/:resetToken`). The password reset link expires in 1 hour.

For this assignment, **you are not expected to send emails**, so the response body also contains the `token` that can be used to reset password.

/auth/resets/:resetToken

- **Method:** POST
- **Description:** Reset the password of a user given a reset token.
- **Clearance**: Any
- **Payload**:

Field	Required	Type	Description
utorid	Yes	string	The utorid of a user who requested a password reset
password	Yes	string	8-20 characters, at least one uppercase, one lowercase, one number, one special character

- **Response**

- **200 OK** on success
- **404 Not Found** if the reset token does not exist.
- **410 Gone** if the reset token expired.

/transactions

- **Method:** POST
- **Description:** Create a new purchase transaction.
- **Clearance**: Cashier or higher
- **Payload**:

Field	Required	Type	Description
utorid	Yes	string	The utorid of the customer making a purchase
type	Yes	string	Must be "purchase"
spent	Yes	number	The dollar amount spent in this transaction. Must be a positive numeric value.
promotionIds	No	array	The IDs of promotions to apply to this transaction
remark	No	string	Any remark regarding this transaction

- **Response**
  - **201 Created** on success

```
{
 "id": 123,
 "utorid": "johndoe1",
 "type": "purchase",
 "spent": 19.99,
 "earned": 80,
 "remark": "",
 "promotionIds": [42],
 "createdBy": "alice666"
}
```

  - **400 Bad Request** when any of the specified promotion IDs are invalid for any reason, e.g., does not exist, expired, or have been used already.

After a purchase is made, the earned amount is automatically added to the user's points balance, unless the cashier processing the transaction is flagged as suspicious. For a regular

purchase transaction without additional promotions, the rate of earning points is 1 point per 25 cents spent (rounded to nearest integer).

- **Method:** POST
- **Description:** Create a new adjustment transaction.
- **Clearance**: Manager or higher
- **Payload**:

Field	Required	Type	Description
utorid	Yes	string	The utorid of the user whose previous transaction is being adjusted
type	Yes	string	Must be "adjustment"
amount	Yes	number	The point amount adjusted in this transaction
relatedId	Yes	number	The ID of the related transaction
promotionIds	No	array	The IDs of promotions to apply to this transaction
remark	No	string	Any remark regarding this transaction

- **Response**
  - **201 Created** on success

```
{
 "id": 125,
 "utorid": "johndoe1",
 "amount": -40,
 "type": "adjustment",
 "relatedId": 123,
 "remark": "",
 "promotionIds": [],
 "createdBy": "smithw42"
}
```

Once an adjustment is made, the amount is automatically reflected in the user's points balance.

- **Method:** GET
- **Description:** Retrieve a list of transactions
- **Clearance**: Manager or higher

- **Payload**:

Field	Required	Type	Description
name	No	string	Filter by utorid or name
createdBy	No	string	Filter by the user who created the transaction
suspicious	No	boolean	Filter by whether the transaction is flagged as suspicious
promotionId	No	number	Filter by a promotion applied to the transaction
type	No	string	Filter by transaction type (can be used without relatedId)
relatedId	No	number	Filter by related ID (must be used with type)
amount	No	number	Filter by point amount (must be used with operator)
operator	No	string	One of "gte" (greater than or equal) or "lte" (less than or equal)
page	No	number	Page number for pagination (default is 1)
limit	No	number	Number of objects per page (default is 10)

- **Response**: `count`, which stores the total number of results (after applying all filters), and `results`, which contains a list of transactions

```
{
 "count": 21,
 "results": [
  {
    "id": 123,
    "utorid": "johndoe1",
    "amount": 80,
    "type": "purchase",
    "spent": 19.99,
    "promotionIds": [],
    "suspicious": false,
    "remark": "",
    "createdBy": "alice666"
  },
  {
    "id": 124,
    "utorid": "johndoe1",
    "amount": -1000,
    "type": "redemption",       // see POST /users/me/transactions for redemption transactions
```

```
      "relatedId": 666,
      "promotionIds": [],
      "redeemed": 1000,
      "remark": "",
      "createdBy": "johndoe1"
    },
    {
      "id": 125,
      "utorid": "johndoe1",
      "amount": -40,
      "type": "adjustment",
      "relatedId": 123,
      "promotionIds": [],
      "suspicious": false,
      "remark": "",
      "createdBy": "smithw42"
    },
    // More transaction objects...
  ]
}
```


For the `relatedId` field, its value will be dependent on the type of the transaction:

- *Adjustment*: the ID of the transaction for which the adjustment is being made to.
- *Transfer*: the ID of the other user, i.e., for the sender's transaction, relatedId is the ID of the receiver; for the receiver's transaction, relatedId is the ID of the sender.
- *Redemption*: the user ID of the cashier who processed the redemption -- can be null if the redemption has not been processed yet.
- *Event*: the ID of the event from which points were disbursed.

/transactions/:transactionId

- **Method:** GET
- **Description:** Retrieve a single transaction
- **Clearance**: Manager or higher
- **Payload**: None
- **Response**:


```
{
```

```
  "id": 123,
  "utorid": "johndoe1",
  "type": "purchase",
  "spent": 19.99,
  "amount": 80,
  "promotionIds": [],
  "suspicious": false,
  "remark": "",
  "createdBy": "alice666"
}
```


/transactions/:transactionId/suspicious

- **Method:** PATCH
- **Description:** Set or unset a transaction as being suspicious
- **Clearance**: Manager or higher
- **Payload**:

Field	Required	Type	Description
suspicious	Yes	boolean	true or false

- **Response**:


```
{
  "id": 123,
  "utorid": "johndoe1",
  "type": "purchase",
  "spent": 19.99,
  "amount": 80,
  "promotionIds": [],
  "suspicious": true,
  "remark": "",
  "createdBy": "alice666"
}
```


When marking a transaction as suspicious (changing the flag from false to true), the amount should be immediately deducted from the user's points balance, which may result in a negative

balance. Conversely, when verifying a transaction as not suspicious (changing the flag from true to false), the amount should be immediately credited to the user's points balance.

/users/:userId/transactions

- **Method:** POST
- **Description:** Create a new transfer transaction between the current logged-in user (sender) and the user specified by userId (the recipient)
- **Clearance**: Regular or higher
- **Payload**:

Field	Required	Type	Description
type	Yes	string	Must be "transfer"
amount	Yes	number	The points amount to be transferred. Must be a positive integer value.
remark	No	string	Any remark regarding this transaction

- **Response**

201 Created on success

{

- "id": 127,
  "sender": "johndoe1",
  "recipient": "friend69",
  "type": "transfer",
  "sent": 500,
  "remark": "Poker night",
  "createdBy": "johndoe1"
  }

- **400 Bad Request** if the sender does not have enough points
- **403 Forbidden** if the sender is not verified.

Upon success, two transactions should be created: one for sending the amount and another for receiving it. For the sender, `relatedId` should be the user id of the recipient, For the receiver, `relatedId` should be the user id of the sender.

/users/me/transactions

- **Method:** POST
- **Description:** Create a new redemption transaction.
- **Clearance**: Regular or higher
- **Payload**:

Field	Required	Type	Description
type	Yes	string	Must be "redemption"
amount	Yes	number	The amount to redeem in this transaction. Must be a positive integer value.
remark	No	string	Any remark regarding this transaction

- **Response**
  - **201 Created** on success

```
{
 "id": 124,
 "utorid": "johndoe1",
 "type": "redemption",
 "processedBy": null,
 "amount": 1000,
 "remark": "",
 "createdBy": "johndoe1"
}
```

  - **400 Bad Request** if the requested amount to redeem exceed the user's point balance.
  - **403 Forbidden** if the logged-in user is not verified.

A redemption transaction does not immediately deduct from the user's point balance. Instead, a cashier must process the redemption through `PATCH /transactions/:transactionId/processed`.

- **Method:** GET
- **Description:** Retrieve a list of transactions owned by the currently logged in user
- **Clearance**: Regular or higher
- **Payload**:

Field	Required	Type	Description
type	No	string	Filter by transaction type

relatedId	No	number	Filter by related ID (must be used with type)
promotionId	No	number	Filter by promotion applied to the transaction
amount	No	number	Filter by point amount (must be used with operator)
operator	No	string	One of "gte" (greater than or equal) or "lte" (less than or equal)
page	No	number	Page number for pagination (default is 1)
limit	No	number	Number of objects per page (default is 10)

- **Response**: `count`, which stores the total number of results (after applying all filters), and `results`, which contains a list of transactions


```
{
 "count": 21,
 "results": [
  {
   "id": 123,
   "type": "purchase",
   "spent": 19.99,
   "amount": 80,
   "promotionIds": [],
   "remark": "",
   "createdBy": "alice666"
  },
  {
   "id": 125,
   "amount": -40,
   "type": "adjustment",
   "relatedId": 123,
   "promotionIds": [],
   "remark": "",
   "createdBy": "smithw42"
  },
  {
   "id": 127,
   "amount": -500,
   "type": "transfer",
   "relatedId": 35,
   "promotionIds": [],
   "remark": "Poker night",
   "createdBy": "johndoe1"
  }
```

```
    // More transaction objects...
  ]
}
```

/transactions/:transactionId/processed

- **Method:** PATCH
- **Description:** Set a redemption transaction as being completed
- **Clearance**: Cashier or higher
- **Payload**:

Field	Required	Type	Description
processed	Yes	boolean	Can only be true

- **Response**:
    - **200 OK** on success

```
{
  "id": 124,
  "utorid": "johndoe1",
  "type": "redemption",
  "processedBy": "alice666",
  "redeemed": 1000,
  "remark": "",
  "createdBy": "johndoe1"
}
```

    - **400 Bad Request**
        - If the transaction is not of type "redemption"
        - If the transaction has already been processed

When marking a redemption transaction as processed (changing the flag from false to true), the amount should then be deducted from the user's points balance.

/events

- **Method:** POST
- **Description:** Create a new point-earning event.
- **Clearance**: Manager or higher
- **Payload**:

Field	Required	Type	Description
name	Yes	string	The name of the event
description	Yes	string	The description of the event
location	Yes	string	The location of the event
startTime	Yes	string	ISO 8601 format
endTime	Yes	string	ISO 8601 format -- must be after startTime
capacity	No	number	Must be a positive number, or `null` if there is no limit to the number of attendees
points	Yes	number	Points allocated such that the organizers can distribute freely during the event. Must be a positive integer.

- **Response**
  - **201 Created** on success

```
{
 "id": 1,
 "name": "Event 1",
 "description": "A simple event",
 "location": "BA 2250",
 "startTime": "2025-11-10T09:00:00Z",
 "endTime": "2025-11-10T17:00:00Z",
 "capacity": 200,
 "pointsRemain": 500,
 "pointsAwarded": 0,
 "published": false,
 "organizers": [],
 "guests": []
}
```

Please see `PATCH /events/:eventId` for a list of possible errors.

- **Method:** GET

- **Description:** Retrieve a list of events
- **Clearance**: Regular or higher
- **Payload**:

Field	Required	Type	Description
name	No	string	Filter by name of the event
location	No	string	Filter by location
started	No	boolean	Filter events that have started already (false would mean that the event has not started)
ended	No	boolean	Filter events that have ended already (false would mean that the event has not ended)
showFull	No	boolean	Show events that are full (default is false)
page	No	number	Page number for pagination (default is 1)
limit	No	number	Number of objects per page (default is 10)

- **Response**: `count`, which stores the total number of results (after applying all filters), and `results`, which contains a list of events
    - **200 OK** on success

```
{
 "count": 2,
 "results": [
  {
   "id": 1,
   "name": "Event 1",
   "location": "BA 2250",
   "startTime": "2025-11-10T09:00:00Z",
   "endTime": "2025-11-10T17:00:00Z",
   "capacity": 200,
   "numGuests": 0
  }
  // More event objects...
 ]
}
```

    - **400 Bad Request** when both started and ended are specified (it is never necessary to specify both started and ended).

Note that regular users cannot see unpublished events.

- **Method:** GET
- **Description:** Retrieve a list of events
- **Clearance**: Manager or higher
- **Payload**: Same as above, with these additional fields added

Field	Required	Type	Description
published	No	boolean	Filter events that are published (or not)

- **Response**: `count`, which stores the total number of results (after applying all filters), and `results`, which contains a list of events
    - **200 OK** on success

```
{
 "count": 5,
 "results": [
  {
   "id": 1,
   "name": "Event 1",
   "location": "BA 2250",
   "startTime": "2025-11-10T09:00:00Z",
   "endTime": "2025-11-10T17:00:00Z",
   "capacity": 200,
   "pointsRemain": 500,
   "pointsAwarded": 0,
   "published": false,
   "numGuests": 0
  }
  // More event objects...
 ]
}
```

Note that for both versions of `GET /events`, the descriptions of the returned events are omitted.

/events/:eventId

- **Method:** GET
- **Description:** Retrieve a single event
- **Clearance**: Regular or higher
- **Payload**: None

- **Response**
  - **200 OK** on success

```
{
 "id": 1,
 "name": "Event 1",
 "description": "A simple event",
 "location": "BA 2250",
 "startTime": "2025-11-10T09:00:00Z",
 "endTime": "2025-11-10T17:00:00Z",
 "capacity": 200,
 "organizers": [
  { "id": 1, "utorid": "johndoe1", "name": "John Doe" }
 ],
 "numGuests": 0
}
```

  - **404 Not Found** if the event is not published.

Note that a regular user cannot see all the information regarding an event, such as the points allocated to the event, or the list of guests, but can see the current number of guests that have RSVPed.

- **Method:** GET
- **Description:** Retrieve a single event
- **Clearance**: Manager or higher, or an organizer for this event
- **Payload**: None
- **Response**:

```
{
 "id": 1,
 "name": "Event 1",
 "description": "A simple event",
 "location": "BA 2250",
 "startTime": "2025-11-10T09:00:00Z",
 "endTime": "2025-11-10T17:00:00Z",
 "capacity": 200,
 "pointsRemain": 500,
 "pointsAwarded": 0,
 "published": false,
 "organizers": [
```

```
    { "id": 1, "utorid": "johndoe1", "name": "John Doe" }
  ],
  "guests": []
}
```

- **Method:** PATCH
- **Description:** Update an existing event.
- **Clearance**: Manager or higher, or an organizer for this event
- **Payload**:

Field	Required	Type	Description
name	No	string	The name of the event
description	No	string	The description of the event
location	No	string	The location of the event
startTime	No	string	ISO 8601 format
endTime	No	string	ISO 8601 format -- must be after startTime
capacity	No	number	Must be a positive number~~, or `null` if there is no limit to the number of attendees~~
points	No	number	**Can only be set by managers:** Points allocated such that the organizers can distribute freely during the event. Must be a positive integer.
published	No	boolean	**Can only be set by managers:** Make the event visible to others (including its organizers). Can only be set to true

- **Response:** The id, name and location, shall always be returned. For others, only the field(s) updated will be returned, e.g., when the published field is updated:
  - **200 OK** on success

```
{
  "id": 1,
  "name": "Event 1",
  "location": "BA 2250",
  "published": true
}
```

  - **400 Bad Request**

- If start time or end time (or both) is in the past.
- If capacity is reduced, but the number of confirmed guests exceeds the new capacity.
- If the total amount of points is reduced, resulting in the remaining points allocated to the event falling below zero. Points already awarded to guests cannot be retracted through this API.
- If update(s) to name, description, location, startTime, or capacity is made after the original start time has passed.
- In addition to the above, if update to endTime is made after the original end time has passed.

- **Method:** DELETE
- **Description:** Remove the specified event.
- **Clearance**: Manager or higher
- **Payload**: None
- **Response**:

  - **204 No Content** on success
  - **400 Bad Request** if the event has already been published

/events/:eventId/organizers

- **Method:** POST
- **Description:** Add an organizer to this event.
- **Clearance**: Manager or higher
- **Payload**:

Field	Required	Type	Description
utorid	Yes	string	The utorid of the organizer (he or she must have an account with us)

- **Response**:

  - **201 Created** on success

    {
     "id": 1,
     "name": "Event 1",

```
  "location": "BA 2250",
  "organizers": [
    { "id": 1, "utorid": "johndoe1", "name": "John Doe" },
    { "id": 2, "utorid": "alice666", "name": "Alice Liddell" }
  ]
}
```


- **400 Bad Request** if the user is registered as a guest to the event (remove user as guest first, then retry)
- **410 Gone** if the event has ended

/events/:eventId/organizers/:userId

- **Method:** DELETE
- **Description:** Remove an organizer from this event.
- **Clearance**: Manager or higher
- **Payload**: None
- **Response**:
    - **204 No Content** on success

/events/:eventId/guests

- **Method:** POST
- **Description:** Add a guest to this event.
- **Clearance**: Manager or higher, or an organizer for this event
- **Payload**:

Field	Required	Type	Description
utorid	Yes	string	The utorid of the guest to be added

- **Response**:
    - **201 Created** on success


```
{
  "id": 1,
```

```
 "name": "Event 1",
 "location": "BA 2250",
 "guestAdded": { "id": 3, "utorid": "jacksun0", "name": "Jack Sun" },
 "numGuests": 1
}
```


- **400 Bad Request** if the user is registered as an organizer (remove user as organizer first, then retry)
- **404 Not Found** if the event is not visible to the organizer yet
- **410 Gone** if the event is full or has ended

/events/:eventId/guests/:userId

- **Method:** DELETE
- **Description:** Remove a guest from this event.
- **Clearance**: Manager or higher (not organizers for this event)
- **Payload**: None
- **Response**:

    - **204 No Content** on success

/events/:eventId/guests/me

- **Method:** POST
- **Description:** Add the logged-in user to the event
- **Clearance**: Regular
- **Payload**: None
- **Response**:

    - **201 Created** on success


```
{
 "id": 1,
 "name": "Event 1",
 "location": "BA 2250",
 "guestAdded": { "id": 4, "utorid": "kian1234", "name": "Mo Kian" },
 "numGuests": 1
```

```
}
```

- **400 Bad Request** if the user is already on the guest list
- **410 Gone** if the event is full or has ended

Only the currently logged-in user should appear in the array.

/events/:eventId/guests/me

- **Method:** DELETE
- **Description:** Remove the logged-in user from this event.
- **Clearance**: Regular
- **Payload**: None
- **Response**:

  - **204 No Content** on success
  - **404 Not Found** if the user did not RSVP to this event
  - **410 Gone** if the event has ended

/events/:eventId/transactions

- **Method:** POST
- **Description:** Create a new reward transaction
- **Clearance**: Manager or higher, or an organizer for this event
- **Payload**:

Field	Required	Type	Description
type	Yes	string	Must be "event"
utorid	No	string	The utorid of the guest to award the points. If utorid is not specified, amount is awarded to **all guests**
amount	Yes	number	Points to award to the guest. Must be a positive integer value.

- **Response**:

- **201 Created** on success (when utorid is specified)

```
{
 "id": 132,
 "recipient": "johndoe1",
 "awarded": 200,
 "type": "event",
 "relatedId": 1,
 "remark": "Trivia winner",
 "createdBy": "alice666"
}
```

- **201 Created** on success (when utorid is not specified)

```
[
 {
  "id": 201,
  "recipient": "reidk129",
  "awarded": 100,
  "type": "event",
  "relatedId": 3,
  "remark": "meditation session",
  "createdBy": "alice666"
 },
 {
  "id": 202,
  "recipient": "craigm34",
  "awarded": 100,
  "type": "event",
  "relatedId": 3,
  "remark": "meditation session",
  "createdBy": "alice666"
 },
 {
  "id": 203,
  "recipient": "campj768",
  "awarded": 100,
  "type": "event",
  "relatedId": 3,
  "remark": "meditation session",
  "createdBy": "alice666"
 }
]
```

- **400 Bad Request**
  - If the user is not on the guest list (even if the capacity is unlimited)
  - If the remaining points is less than the requested amount.

Awarding points to guests can be done after an event has ended. After this transaction is created, the points are awarded to the user immediately.

Points can be awarded to the same guest multiple times, i.e., without restriction. For example, the event organizer can first award johndoe1 500 points, then award all guests (including johndoe1) 50 points each.

/promotions

- **Method:** POST
- **Description:** Create a new promotion.
- **Clearance**: Manager or higher
- **Payload**:

Field	Required	Type	Description
name	Yes	string	The name of the promotion
description	Yes	string	The description of the promotion
type	Yes	string	Either "automatic" or "one-time"
startTime	Yes	string	ISO 8601 format. Must not be in the past.
endTime	Yes	string	ISO 8601 format. Must be after startTime
minSpending	No	number	The minimum spending required to trigger the promotion.. Must be a positive numeric value.
rate	No	number	The promotional rate (on top of the existing rate). Must be a positive numeric value.
points	No	number	The promotional points, added to qualifying purchase transaction. Must be a positive integer value.

- **Response**

201 Created on success


```
{
 "id": 3,
 "name": "Start of Summer Celebration",
```

"description": "A simple promotion",
"type": "automatic",
"startTime": "2025-11-10T09:00:00Z",
"endTime": "2025-11-10T17:00:00Z",
"minSpending": 50,
"rate": 0.01, // for every dollar spent, 1 extra point is added
"points": 0

- }

- **Method:** GET
- **Description:** Retrieve a list of promotions
- **Clearance**: Regular or higher
- **Payload**:

Field	Required	Type	Description
name	No	string	Filter by name of the promotion
type	No	string	Filter by type (either "automatic" or "one-time")
page	No	number	Page number for pagination (default is 1)
limit	No	number	Number of objects per page (default is 10)

- **Response**: `count`, which stores the total number of results (after applying all filters), and `results`, which contains a list of promotions
  - **200 OK** on success

```
{
 "count": 3,
 "results": [
  {
   "id": 3,
   "name": "Start of Summer Celebration",
   "type": "automatic",
   "endTime": "2025-11-10T17:00:00Z",
   "minSpending": 50,
   "rate": 0.01,  // for every dollar spent, 1 extra point is added
   "points": 0
  }
  // More event objects...
 ]
```

```
}
```

A regular user may only see available promotions, i.e., active promotions that they have not used. An active promotion is one that has started, but not ended.

- **Method:** GET
- **Description:** Retrieve a list of promotions
- **Clearance**: Manager or higher
- **Payload**: on top of the fields above, these addition fields are available to managers:

Field	Required	Type	Description
started	No	boolean	Filter promotions that have started already (false would mean that the event has not started)
ended	No	boolean	Filter promotions that have ended already (false would mean that the event has not ended)

- **Response**: `count`, which stores the total number of results (after applying all filters), and `results`, which contains a list of promotions
  - **200 OK** on success

```
{
 "count": 3,
 "results": [
  {
   "id": 3,
   "name": "Start of Summer Celebration",
   "type": "automatic",
   "startTime": "2025-11-10T09:00:00Z",
   "endTime": "2025-11-10T17:00:00Z",
   "minSpending": 50,
   "rate": 0.01,       // for every dollar spent, 1 extra point is added
   "points": 0
  }
  // More event objects...
 ]
}
```

  - **400 Bad Request** when both started and ended are specified (it is never necessary to specify both started and ended).

Note that for both versions of `GET /promotions`, the descriptions of the returned promotions are omitted.

/promotions/:promotionId

- **Method:** GET
- **Description:** Retrieve a single promotion
- **Clearance**: Regular or higher
- **Payload**: None
- **Response**
  - **200 OK** on success

```
{
 "id": 3,
 "name": "Start of Summer Celebration",
 "description": "A simple promotion",
 "type": "automatic",
 "endTime": "2025-11-10T17:00:00Z",
 "minSpending": 50,
"rate": 0.01,
 "points": 0
}
```

  - **404 Not Found** if the promotion is currently inactive (not started yet, or have ended).

- **Method:** PATCH
- **Description:** Update an existing promotion.
- **Clearance**: Manager or higher
- **Payload**:

Field	Required	Type	Description
name	No	string	The name of the event
description	No	string	The description of the promotion
type	No	string	Either "automatic" or "one-time"
startTime	No	string	ISO 8601 format

endTime	No	string	ISO 8601 format. Must be after startTime
minSpending	No	number	The minimum spending required to trigger the promotion. Must be a positive numeric value.
rate	No	number	The promotional rate (on top of the existing rate). Must be a positive numeric value.
points	No	number	The promotional points, added to qualifying purchase transaction. Must be a positive integer value.

- **Response:** The id, name and type, shall always be returned. For others, only the field(s) updated will be returned, e.g., when the endTime field is updated:
    - **200 OK** on success

```
{
 "id": 3,
 "name": "Start of Summer Celebration",
 "type": "automatic",
"endTime": "2025-11-20T17:00:00Z",
}
```

    - **400 Bad Request**
        - If start time or end time (or both) is in the past.
        - If update(s) to `name`, `description`, `type`, `startTime`, `minSpending`, `rate`, or `points` is made after the original start time has passed.
        - In addition to the above, if update to `endTime` is made after the original end time has passed.

- **Method:** DELETE
- **Description:** Remove the specified promotion.
- **Clearance**: Manager or higher
- **Payload**: None
- **Response**:
    - **204 No Content** on success
    - **403 Forbidden** if the promotion has already started.

Getting Started: Suggested Approach

This is a **challenging assignment**, so we strongly recommend that you start early. The project consists of multiple components and tackling them step by step will make the workload more manageable. Below is a structured approach to help you get started and progress efficiently.

1. Understand the Requirements

Before writing any code, thoroughly read the entire handout and make sure you **fully understand**:

- The problem statement and the expected system behavior.
- The **user stories** outlined in the assignment -- these define how the system should function from the user's perspective.
- The **API requirements** and expected responses.
- Any constraints or business rules that must be enforced in your implementation.

Taking time to understand the specifications upfront will save you significant time debugging later.

2. Plan Your Database Schema

- **Start with users**: Identify the key fields each user should have (e.g., `id`, `email`, `password`, `role`).
    - What constraints should be enforced? (e.g., unique utorid, valid UofT email)
- **Define the main entities**: Think about how to structure key components such as:
    - **Transactions**: Different types, relationships, and necessary fields.
    - **Events & Promotions**: How are they linked to users and transactions?
- **Model relationships properly**:
    - How do users **attend** events? Should there be a many-to-many relationship?
    - How do transactions relate to users and events?
- **Draft your schema in** `schema.prisma`:
    - Before coding, consider drawing a schema diagram (use tools like dbdiagram.io) to visualize relationships.
    - Expect **iterations** -- your schema will likely evolve as you implement more features.

3. Implement the Backend Incrementally

Once you have a basic database schema, you can start implementing the backend in a **layered approach**:

- **Begin with foundational layers (Services & Repositories)**
    - Implement **services** that handle business logic.

    - Implement **repositories** that interact with Prisma to perform database operations.
    - For simplicity, you may combine services and repositories in the service layer.
- **Move to API Controllers**
    - Start with simpler CRUD (Create, Read, Update, Delete) operations before handling complex business logic.
    - Implement and test endpoints **one by one**.
- **Prioritize user-related features first**:
    - User authentication (e.g., registration, login) will help you become familiar with Prisma and database interactions.
    - Implementing user-related endpoints first will establish a base for handling authentication and authorization in later stages.
    - These endpoints are also simpler to implement and test, providing a good starting point to understand the system flow.

4. Test as You Go

- **Use Postman or similar tools** to test your API endpoints.
- Test for both **valid inputs** (happy paths) and **edge cases** (e.g., invalid data, unauthorized access, incorrect tokens).
- **Validate database interactions**: After each major change, inspect your database to ensure data is stored and retrieved correctly.

5. Work on More Complex Features

After completing basic authentication and CRUD operations:

- Implement **event and promotion management**.
- Handle **transactions**, which will likely be the most complex part due to business logic. Consider deferring these endpoints until you're comfortable with Prisma and Express.

6. Test End-to-End

Once you have implemented all required features:

- Test **the entire system workflow** by simulating real-world scenarios.
- Ensure that all API endpoints **work together as expected**.
- Cross-check against the original **user stories** -- did you implement everything correctly?

7. Debugging and Refinement

- Review **error handling**: Are errors properly logged and returned to clients with meaningful messages?
- Implement proper **input validation** to prevent invalid data from breaking the system.

- Check your **Prisma queries** -- are they efficient? Do they handle edge cases like missing records?

Final Tips

- **Do not hesitate to ask for help!** If you're stuck:
  - Post on **Piazza**.
  - Attend **office hours** or tutorials.
- **Start early!** This project requires **iteration and testing**, so procrastination will make it much harder.
- **Follow best practices** for Express.js and Prisma to avoid common pitfalls.

Good luck, and happy coding!