Blueprint Games Fall 2026

Your task

Today, you'll work as a Blueprint project team member to plan software for Open Referral. With your team, define the project's goals, deliverables, and features, assuming **14 weeks, or one semester, to build it**.

You won't write code today. Your team will discuss the project's needs, choose which features to prioritize, outline user flows, and sketch the data the system needs.

You don't need to understand every technical term before you start. Ask your teammates or E-Board for help, and talk through your ideas together. We're interested in how you collaborate and break down the problem.

Meet the project partner

Open Referral

Project summary

Open Referral partnered with Stevens Blueprint to upgrade the **HSDS Transformer**, a tool that converts resource directory data into the Human Services Data Specification, or HSDS, format. HSDS provides a shared structure for information about services, including their locations and accessibility, so different systems can exchange that information.

The project's objective was to support HSDS 3.0 and make it easier to convert data from other formats into standardized output. Today, you'll use that project as the basis for your own plan.

The challenge

Different organizations maintain resource directories in separate systems. Those directories often contain overlapping information, but their formats don't match, making it difficult to share data or keep it consistent across systems.

HSDS gives those systems a common format, but adopting it takes work. Someone must understand how their data is organized, map its fields to HSDS, and run the transformation. That process can be especially difficult for people who know the service information well but have limited technical skills or time.

The objective

Plan a process that helps people:

1. Map the structure of their source data to HSDS using a documented, repeatable method, initially through a spreadsheet.
2. Load their source data and the mapping that describes how to convert it.
3. Transform that data into HSDS-compliant output.
4. Validate the output and understand any problems that need attention.

Validation checks whether the output follows the standard. Documentation and examples should help someone repeat the process with their own data.

A user-friendly interface for nontechnical users is a stretch goal. Decide what your team can deliver within 14 weeks and which improvements you would leave for later.

Questions to guide your plan

Use these questions to guide your decisions:

- How could the tool reduce the work needed to convert service data into a shared format?
- How could it help people identify and fix problems before sharing the converted data?
- What would make the process understandable for someone without programming experience?

User personas

Consider three types of users with different skills and needs:

Skyler, a data admin. Skyler manages data flows and databases, has advanced data skills, and can use a command-line interface. He wants to retrieve data through an API and run transformations regularly as part of a data pipeline. He needs a repeatable process that fits into existing systems.

Sasha, a professional content manager. Sasha knows the service information and can organize data and create a mapping in a spreadsheet. She has intermediate data skills but is unfamiliar with GitHub, Python, and command-line tools. She wants a web interface that lets her transform data without installing several programs or learning to use them.

Sheila, a volunteer or part-time content manager. Sheila knows the service information but has basic data skills and limited training in data management. She needs guidance to map her data to HSDS, load it, and run the transformation. The process should let her use her knowledge of the services without requiring software development skills.

Decide which personas your proposed tool will support first and explain how their needs affect your choices.

Getting started

When Blueprint works with a project partner, the partner describes the challenge, and the team works out how to address it. Today, you'll make those decisions. Start with the summary, challenge, objective, and user personas above.

Your team can choose its approach, but be ready to explain your reasoning. Different solutions have different tradeoffs. **Plan for a project your team can complete in 14 weeks.**

Discuss questions such as:

- Would a website, command-line tool, API, or a combination best meet these needs?
- What do you need to know about each user? Consider their comfort with technology, the data they work with, and accessibility needs.
- How would someone supply data, define a mapping, review problems, and retrieve the result?
- Does your approach need accounts or saved work? If so, what should each person be able to view or change?
- Which features would be most useful to Open Referral and the people using the Transformer?
- What can you reuse from the existing Transformer, and what needs to change?
- What can your team realistically build, test, and prepare for use within 14 weeks?
- What belongs in the first version, and what would you leave for later?

When you're ready, make a copy of the team design document template and share it with your teammates. The template includes tips for taking notes.

Deliverables

Your team will submit its design document at the end of the exercise. You won't give a presentation.

Keep your notes as you work. Include ideas you considered, questions you couldn't answer, and reasons for your decisions. Rough sketches and unfinished work are welcome. We care more about your process than polished formatting.

Your document can include:

- User stories that describe what someone needs to do and why.
- A list of key features and their priorities.
- User flow diagrams that show how someone completes a task.
- Wireframes that sketch the main screens.
- A data model that identifies the information the tool processes and how it connects. If your design needs a database, include its schema; if it doesn't, explain how the tool handles data without one.
- API endpoints that describe how the application could request or change data.

If you create diagrams or wireframes in another tool, include screenshots in your document.

These are examples of useful artifacts, not a requirement to finish every item. Submit what your team develops, including work in progress.

Submission Link

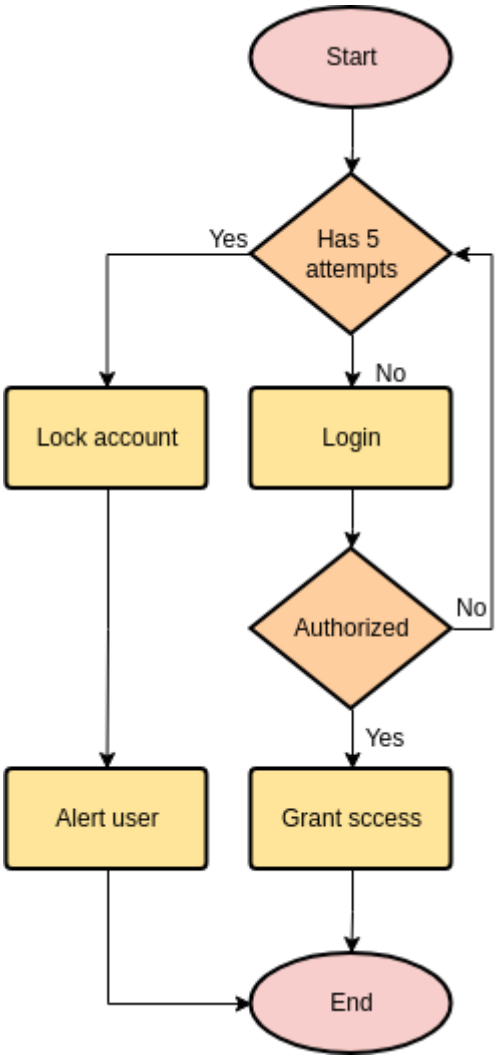
Appendix

Use these explanations if your team needs help getting started.

User flow diagrams

A user flow diagram shows the steps someone takes to complete a task and how the system responds. For example, a login flow might show someone entering their credentials, the system checking them, and the person either reaching their account or receiving an error.

You may need separate diagrams for different tasks or account types.



Common symbols include:

- **Ovals** for the start and end of a flow.
- **Rectangles** for actions taken by a person or the system.
- **Diamonds** for decisions, with labeled branches such as "Yes" and "No."
- **Arrows** to show the order of steps.

Application programming interfaces

An application programming interface, or API, lets software communicate with other software. In a web application, the frontend can use API endpoints to request data from the server or ask it to make changes.

An endpoint combines an address with an HTTP method that describes the operation.

For example, an application that manages members might use:

Method and endpoint	Purpose
`GET /api/v1/members`	Retrieve a list of members.

`GET /api/v1/members/{id}`	Retrieve one member by ID.
`POST /api/v1/members`	Create a member.
`PUT /api/v1/members/{id}`	Replace a member's editable information.
`PATCH /api/v1/members/{id}`	Update selected fields for a member.
`DELETE /api/v1/members/{id}`	Delete a member.

The `{id}` placeholder represents the ID of a particular member.

To create a member, the frontend could send a `POST` request with a JSON body:

```
POST /api/v1/members
```

```
{
  "name": "John Doe",
  "teamId": "team-123"
}
```

The server could assign the new member an ID when it saves the record.

To change only the member's team, the frontend could send:

```
PATCH /api/v1/members/{id}
```

```
{
  "teamId": "team-456"
}
```

To retrieve members of a particular team, the application could use:

```
GET /api/v1/teams/{teamId}/members
```

`GET` requests read data. Methods such as `POST`, `PUT`, `PATCH`, and `DELETE` request changes.

For this exercise, you only need to describe the endpoints your system might need and what they do. You don't need to implement them.

Database schemas

A database schema describes the information a system stores and the relationships between records.

For example, an application might store teams and members. If each member belongs to one team, the schema could look like this:

```
Team
- id: primary key
- name

Member
- id: primary key
- name
- team_id: foreign key referencing Team.id
```

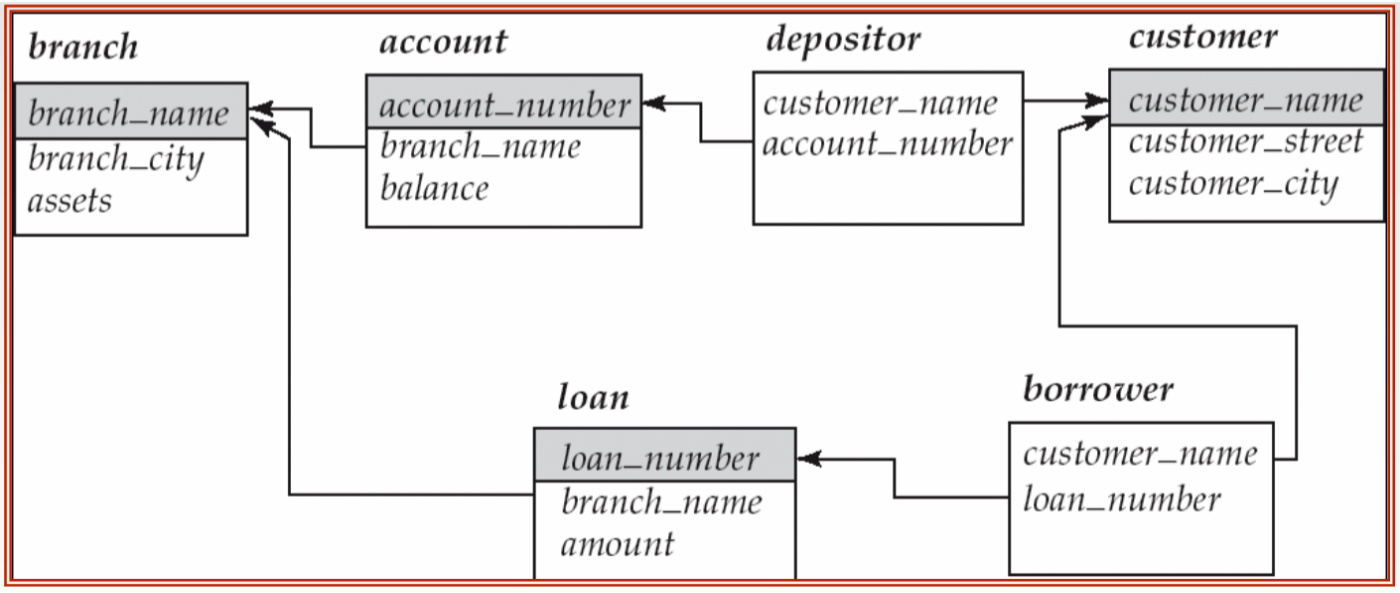
A **primary key** uniquely identifies a record. A **foreign key** references a record in another table.

In this example, one team can have many members, and each member belongs to one team. That's a one-to-many relationship when viewed from the team, and a many-to-one relationship when viewed from a member.

Other relationships include:

- **One-to-one:** Each record relates to at most one record on the other side.
- **Many-to-many:** Records on both sides can relate to multiple records. For example, a service can be available at multiple locations, and a location can offer multiple services.

In a relational database, a many-to-many relationship usually uses a connecting table. A table called `ServiceLocation`, for example, could connect a service ID to a location ID.



For this exercise, list the records, fields, and relationships your system needs. A diagram or a plain-text outline is enough. You don't need to write SQL, and your team can decide whether the Transformer needs a database or can process files without storing them.

Tips

A process to try

1. Identify the most important tasks each type of user needs to complete.
2. Choose the features that support those tasks.
3. List the information those features need.
4. Decide how to organize that information, whether to store it, and who can view or change it.
5. Map the steps someone takes to complete each task.
6. Sketch the screens that support those steps.
7. Check whether your plan fits within 14 weeks, and adjust the scope if needed.

Keep the users in mind

As you make decisions, consider:

- Can someone understand what to do without training?
- Will the system work on the devices they have?
- Can people with different accessibility needs use it?
- If your design includes accounts, does each account have access only to the information it needs?

Talk through disagreements, record your assumptions, and ask for help when you get stuck. Your notes should show how your team reached its decisions.

Revision #6
Created 2026-03-01 23:46:27 UTC by Lucas Ha
Updated 2026-09-30 00:23:02 UTC by Nishit Sharma