Description: Implement as many items as possible, extra features are welcomed but a not required condition. Any implementation will be considered correct if it doesn't conflict with the requirements. The order of performing tasks remains for the developer's consideration if the implementation allows to reuse or extend any part of code (block, service, etc.) and if this doesn't contradict the requirements, then this is not prohibited. If the task does not specify the details or UI elements then this is not required in context and remains for the developer's consideration.

Result: Source code(including the local history of git changes) and database dump.

1. Create a Custom Block plugin (programmatically), the block should contain a node listing grouped by type. The data should be displayed as a table. The data is displayed only for users with "View published content". Users with the "Authenticated" role have permissions to "View published content".

2. Create a service. The service class implements a single public method: "fetchUserByRoute". The fetchUserByRoute method gets the user from the current route and, if it is available, returns a label. If we assume that the given service is "contrib" solution and we cannot modify the existing class code. Extend the functionality of the service so that it is possible to call two public methods: fetchUserByRoute and fetchNodeByRoute. The fetchNodeByRoute method implements similar functionality, which gets a node from the current path and, if it is available, it returns a title.

3. Add a field to the user entity with the ability to save one of the two values “Keep private” and “No restriction”. Add a custom check to the “entity.user.canonical” route (/user/{user}). The other users should not have access to the chosen user's page if the “Keep private” option is selected. The task should affect the route only and should not affect the access to the entity. We can also ignore the user "superadmin" (id: 1) checks, because many checks for it can be ignored by core functionality.

4. Create a custom entity (programmatically). The custom entity should have an entity reference field definition with an allowed node bundle of bacis_page type. Create views with an entity list as a table, the table should output a link to the referenced node. The task does not include a specific requirement for annotation of the entity and all available options, but attention will be paid to the quality of the approach.

5. Implement an ImprovedNode class.The class should extend "Node" which an implementation of NodeInterface. Add an implementation of the getTrimmedTitle method the the new class that returns the first 10 characters of the title, if available. Make sure the new class overrides the existing one and the core entity load implementation returns the new class. Thus, the method can be called as "$node-> getTrimmedTitle()".

6. Add two fields to the "ArticlePage" bundle of the node content type (you can use the UI and any type of the fields that you would like). Implement a multi-step form to override the "ArticlePage" form, splitting each of the two fields, the first field on the

first step and the second field on the second step, respectively. The main criteria is to demonstrate the approach to creating a multistep form.

7. Create Custom Block plugin (programmatically) for anonymous users. Since we want to return actual information from the backend of the user, the block will return one of two options for the text:
    - “Server time contains an even number” - if the current timestamp of the server is even.
    - “Server time contains an odd number” - if the current timestamp of the server is odd.

    Since we use time as some random unknown value. Make sure that the block receives up-to-date information from the backend. Implementation should consider page caching for an anonymous user and should not affect performance.