Overriding blocks in mod-circulation

Context

Override functionality allows the user to perform an action that should be otherwise blocked.

Currently, there's only one type of check-out override - when the item is not loanable, we can make a call to **POST /circulation/override-check-out-by-barcode** to override this block and create a loan. There is also an endpoint for overriding renewals of items that are not renewable, declared lost, aged to lost etc. (**POST /circulation/override-renewal-by-barcode**).

Requirements

Users need to be able to apply different kinds of overrides in multiple steps. Specifically, override of patron blocks will be added (~~UXPROD-1130~~ - Allow for override of patron blocks CLOSED) as well as override of item blocks (~~UXPROD-2127~~ - Allow for override of item blocks CLOSED). The desired check-out flow should look like this:

- During a check-out, the user should be able to override patron blocks first (right after the patron is selected) and then start checking out items to the selected patron.
- In case when any of the check-outs are blocked for reasons other than patron block (item is not loanable, item limit block etc.), the user makes decisions about overriding those blocks separately, but patron blocks should be overridden automatically (because it has been already decided on the previous step).
- If there are multiple blocks that prevent an item from being borrowed, the whole list of reasons why this item can't be checked out should be displayed. The user should be able to review and override all of them in one go (with the appropriate permissions, of course).

Proposed changes to mod-circulation

Get rid of the "override" endpoints.

These endpoints should be removed:

- **POST /circulation/override-check-out-by-barcode**
- **POST /circulation/override-renewal-by-barcode**

Endpoints that are responsible for regular borrowing, requests and renewals will also be responsible for overrides now:

1. **POST /circulation/check-out-by-barcode**
2. **POST /circulation/renew-by-barcode**
3. **POST /circulation/renew-by-id**
4. **POST /circulation/requests**
5. **PUT /circulation/requests**

UI might need to make two calls to the same endpoint in order to complete the action. First - to get the list of overridable blocks, second - to override these block after user's decision.

Return full list of overridable blocks

Endpoints (1-5) should return a full list of blocks that have to be overridden in order for action to be completed. When checking for blocks these processes shouldn't fail immediately when they find the first block. They should check for all blocks instead and include the results in the response. Typically, in case of an error the response structure looks like this:

Overridable error example

```
{
        "errors": [{
                "message": "Error message",
                "parameters": [{
                        "key": "key-1",
                        "value": "value-1"
                }, {
                        "key": "key-2",
                        "value": "value-2"
                }],
                "overridableBlock": {
                        "name": "patronBlock",
                        "missingPermissions": [
                                "circulation.override-patron-block",
                                "circulation.override-item-limit-block"
                        ]
                }
        }
}
```

An overridable error will contain a new property **overridableBlock** with following properties:

- **name** - the name of the block, currently one of: `"patronBlock", "itemLimitBlock", "itemNotLoanableBlock"`
- **missingPermissions** - array of missing Okapi permissions required to override the block. The user might not have these permissions, but they may want to request permission escalation (UXPROD-2645 - Permission escalation for override OPEN) based on this information.

Non-overridable errors will not contain this new property.

If response contains overridable errors only, the client will know that requested action can be completed by overriding corresponding blocks. Presence of non-overridable errors in the response indicates that there is no point in trying to override the blocks.

Expect a list of blocks that user wants to override

Request bodies of endpoints (1-5) need to include a list of blocks that the requester wants to override. New (optional) field should be added to JSON schemas:

```
"overrideBlocks": {
  "description": "Blocks that need to be overridden",
  "type": "object",
  "properties": {
    "itemNotLoanableBlock" : {
      "description": "Parameters for overriding of the 'item not loanable' block",
      "type": "object",
      "properties": {
        "dueDate": {
          "description": "Due date for a new loan",
          "type": "string",
          "format": "date-time"
        }
      },
      "required": [
        "dueDate"
      ]
    },
    "patronBlock": {
      "description": "Parameters for overriding of the patron block",
      "type": "object",
      "properties": {
        ...
      }
    },
    "itemLimitBlock": {
      "description": "Parameters for overriding of the 'item limit' block",
      "type": "object",
      "properties": {
        ...
      }
    }
  }
}
```

List of the appropriate blocks and their parameters may be dependent on the endpoint.

In case when this parameter is included in the request, the server should override all of the specified blocks if the user has sufficient permissions.

Managing permissions

Each overridable action will be associated with a user permission that is required in order to override this action. These permissions need to be part of **permissionsDesired** rather than **permissionsRequired** which will allow to handle them in the code. For example, new permissions for **POST /circulation/check-out-by-barcode:**

```
"permissionsDesired": [
  "circulation.override-patron-block",
  "circulation.override-item-limit-block",
  "circulation.override-item-not-loanable-block"
]
```