External Dependencies for Typescript

References

- jqassistant
- Neo4j Python Driver

External Typescript Module Usage

External Module

An external Typescript module is marked with the label `ExternalModule` and the declarations it provides with `ExternalDeclaration`. In practice, the distinction between internal and external isn't always that clear. When there is a problem following the project configuration like discussed in Missing Interfaces and other elements in the Graph, some internal dependencies might be imported as external ones.

To have a second indicator, the property `isNodeModule` is written with Add_module_properties.cypher in prepareAnalysis.sh. For most package managers this should then be sufficient. As of now (June 2024), it might not work with Yarn Plug'n'Play.

Table 1 - Top 20 most used external packages overall

This table shows the external packages that are used by the most different internal types overall. Additionally, it shows which types of the external modules are actually used. External annotations are also listed.

Only the top 20 entries are shown. The whole table can be found in the following CSV report: `External_module_usage_overall_for_Typescript`

Columns:

- *externalModuleName* is the name of the external module prepended by its namespace if given. Example: "@types/react"
- *numberOfExternalCallerModules* is the number of modules that use that external module

- *numberOfExternalCallerElements* is the number of elements (functions, classes,...) that use that external module
- *numberOfExternalDeclarationCalls* is how often the external declarations of that external module are imported
- *numberOfExternalDeclarationCallsWeighted* is how often the external declarations of that external module are actually used
- *allModules* contains the total count of all analyzed internal modules
- *allInternalElements* contains the total count of all analyzed exported internal elements (function, classes,...)
- *exampleStories* contains a list of sentences that contain concrete examples (for explanation and debugging)

	externalModuleName	numberOfExternalCallerModules	numberOfExternalCallerElements	numberOfExternalDeclarationCalls	numberOfExternalDeclarationCallsWeight
0	@types/react	5	35	195	5
1	@remix-run/router	4	37	390	6
2	@types/react-native	1	5	28	
3	@ungap/url-search-params	1	2	4	

Table 1 Chart 1a - Most called external modules in % by internal elements (more than 0.7% overall)

External modules that are used less than 0.7% are grouped into "others" to get a cleaner chart containing the most significant external modules and how ofter they are called by internal elements in percent.

```
<Figure size 640x480 with 0 Axes>
```

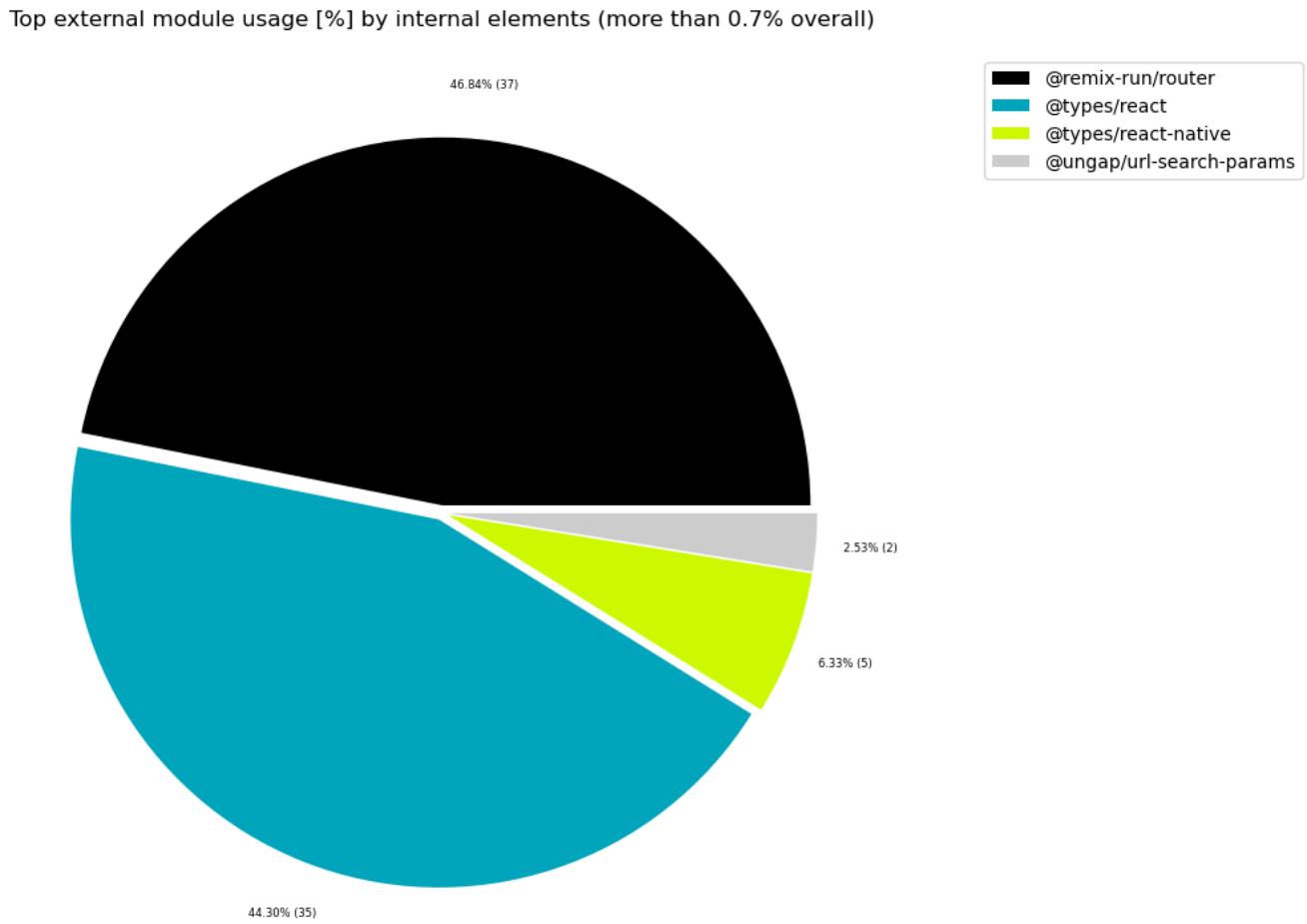


Table 1 Chart 1b - Most called external modules in % by internal elements (less than 0.7% overall "others" drill-down)

Shows the lowest (less than 0.7% overall) most called external modules. Therefore, this plot breaks down the "others" slice of the pie chart above. Values under 0.3% from that will be grouped into "others" to get a cleaner plot.

```
No data to plot for title 'Top external module usage [%] by internal elements  (less than 0.7% overall "others" drill-down)'.
```

Table 1 Chart 2a - Most called external modules in % by internal modules (more than 0.7% overall)

External modules that are used less than 0.7% are grouped into "others" to get a cleaner chart containing the most significant external modules and how ofter they are called by internal modules in percent.

```
<Figure size 640x480 with 0 Axes>
```

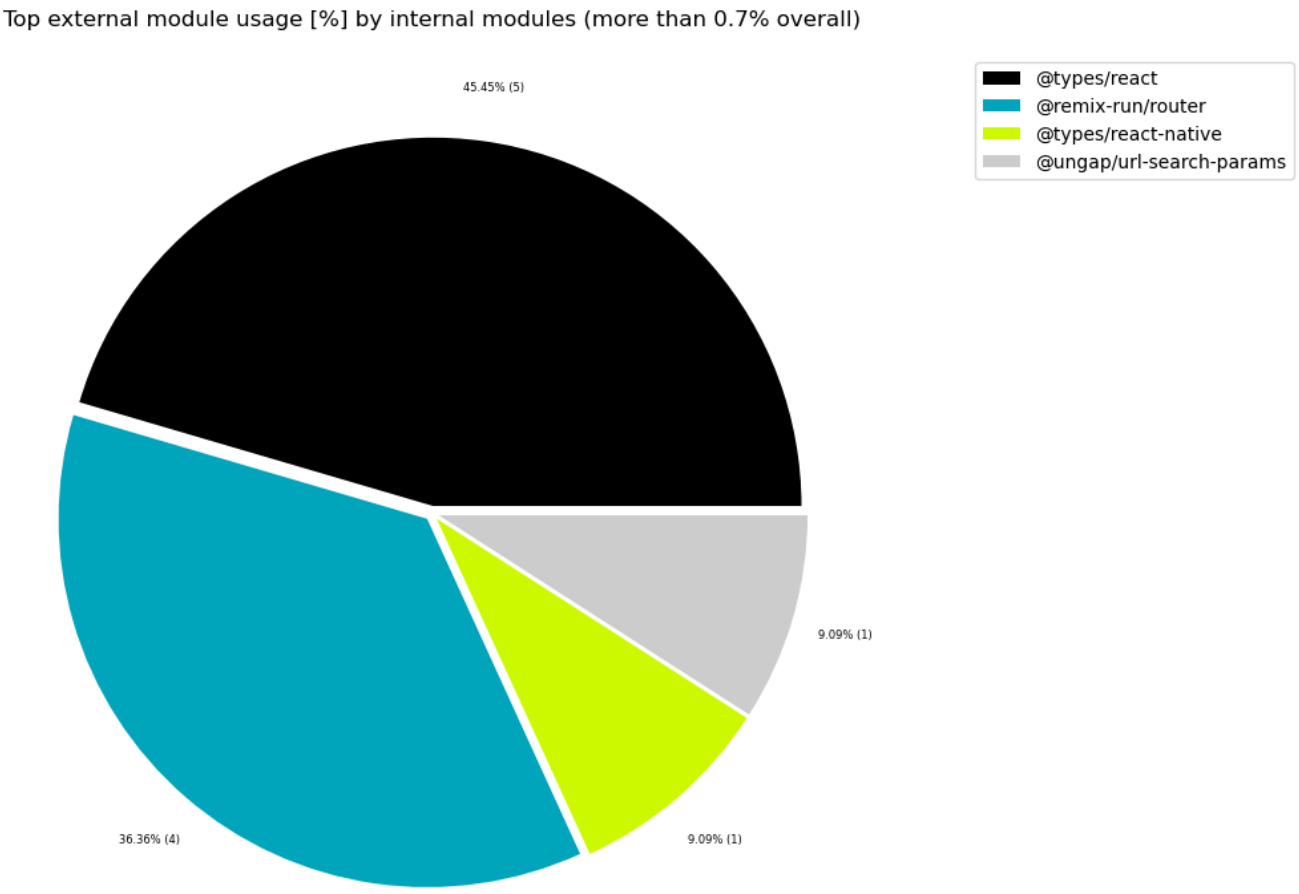


Table 1 Chart 2b - Most called external modules in % by internal modules (less than 0.7% overall "others" drill-down)

Shows the lowest (less than 0.7% overall) most called external modules. Therefore, this plot breaks down the "others" slice of the pie chart above. Values under 0.3% from that will be grouped into "others" to get a cleaner plot.

```
No data to plot for title 'Top external module usage [%] by internal modules (less th
an 0.7% overall "others" drill-down)'.
```

Table 2 - Top 20 most used external namespaces

This table shows external namespaces that are used by the most different internal elements (functions, classes,...) overall.

Additionally, it shows how many of the declarations of the external namespace are actually used.

Only the top 20 entries are shown. The whole table can be found in the following CSV report: `External_namespace_usage_overall_for_Typescript`

Columns:

- *externalNamespaceName* is the name of the external namespace (empty if none). Example: "@types". All other columns are aggregated/grouped by it.
- *numberOfExternalCallerModules* is the number of modules that use that external module
- *numberOfExternalCallerElements* is the number of elements (functions, classes,...) that use that external module
- *numberOfExternalDeclarationCalls* is how often the external declarations of that external module are imported
- *numberOfExternalDeclarationCallsWeighted* is how often the external declarations of that external module are actually used
- *allModules* contains the total count of all analyzed internal modules
- *allInternalElements* contains the total count of all analyzed exported internal elements (function, classes,...)
- *exampleStories* contains a list of sentences that contain concrete examples (for explanation and debugging)

	externalNamespaceName	numberOfExternalCallerModules	numberOfExternalCallerElements	numberOfExternalDeclarationCalls	numberOfExternalDeclarationCallsW
0	@types	5	36	223	
1	@remix-run	4	37	390	
2	@ungap	1	2	4	

Table 2 Chart 1a - Most called external namespaces in % by internal element (more than 0.7% overall)

External namespaces that are used less than 0.7% are grouped into "others" to get a cleaner chart containing the most significant external namespaces and how ofter they are called by internal elements in percent.

```
<Figure size 640x480 with 0 Axes>
```

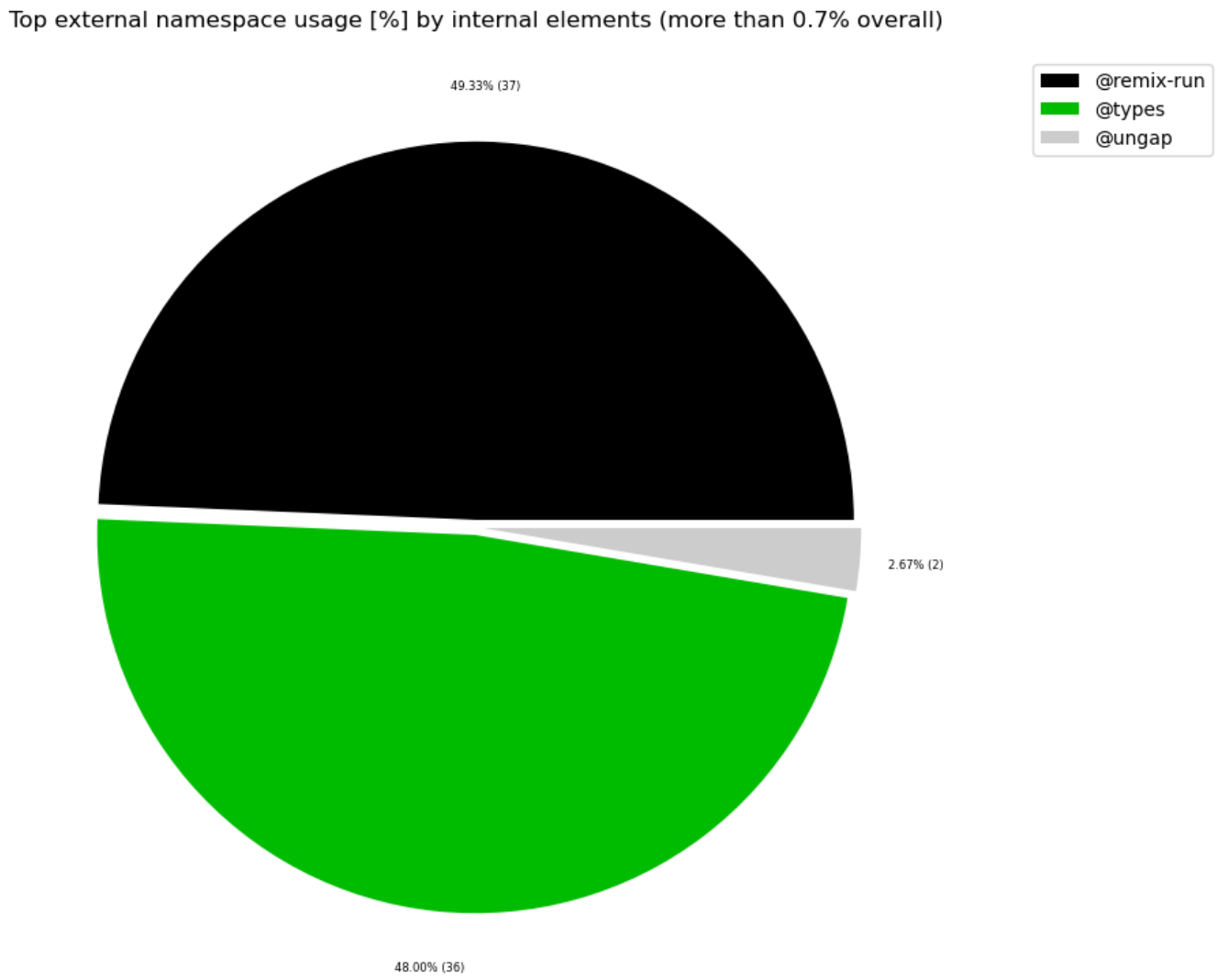


Table 2 Chart 1a - Most called external namespaces in % by internal element (less than 0.7% overall "others" drill-down)

Shows the lowest (less than 0.7% overall) most called external namespaces. Therefore, this plot breaks down the "others" slice of the pie chart above. Values under 0.3% from that will be grouped into "others" to get a cleaner plot.

```
No data to plot for title 'Top external namespace usage [%] by internal elements (les
s than 0.7% overall "others" drill-down)'.
```

Table 2 Chart 2a - Most called external namespaces in % by internal modules (more than 0.7% overall)

External namespaces that are used less than 0.7% are grouped into "others" to get a cleaner chart containing the most significant external namespaces and how ofter they are called by internal modules in percent.

```
<Figure size 640x480 with 0 Axes>
```

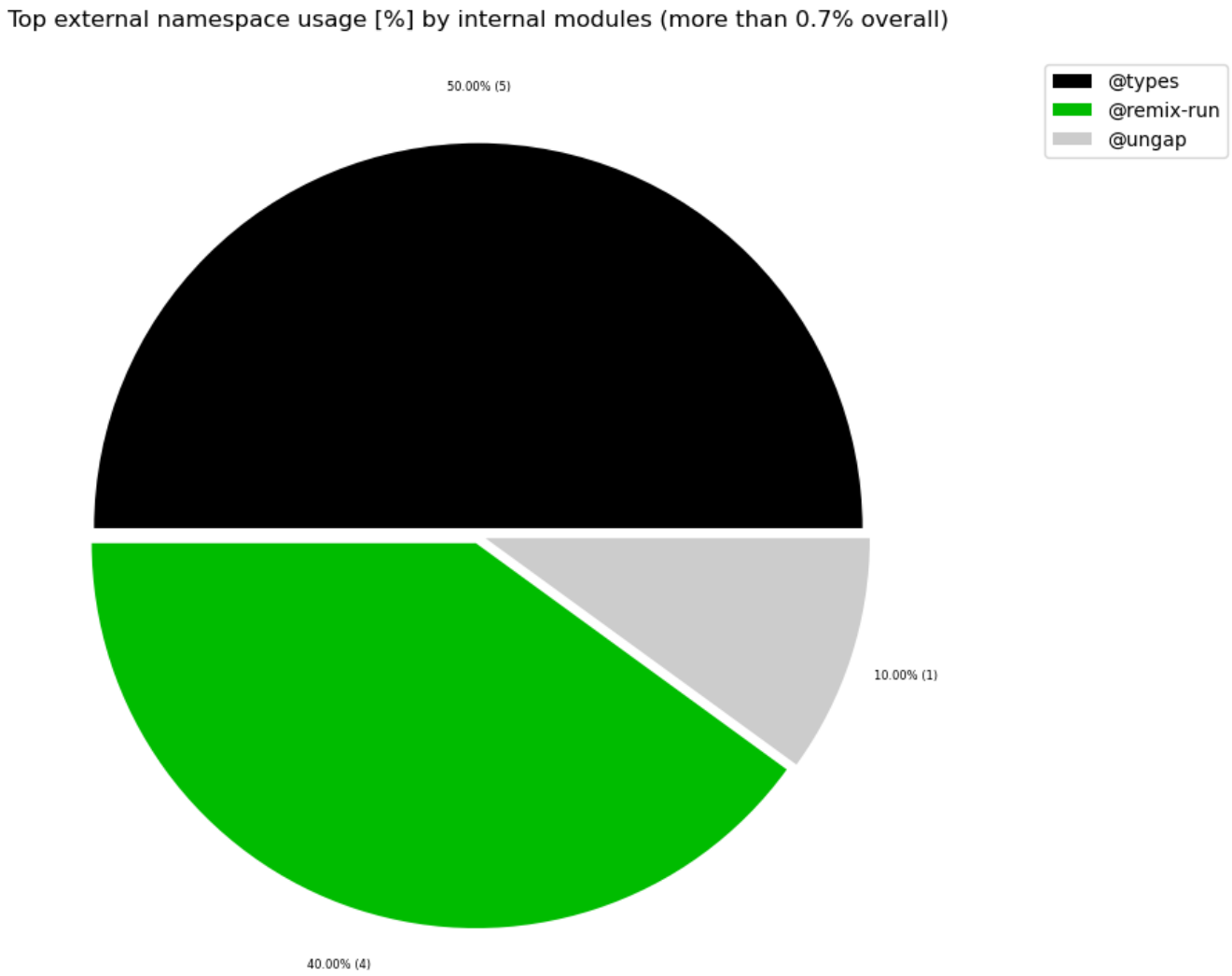


Table 2 Chart 2b - Most called external namespaces in % by internal modules (less than 0.7% overall "others" drill-down)

Shows the lowest (less than 0.7% overall) most called external namespaces. Therefore, this plot breaks down the "others" slice of the pie chart above. Values under 0.3% from that will be grouped into "others" to get a cleaner plot.

```
No data to plot for title 'Top external namespace usage [%] by internal modules (less than 0.7% overall "others" drill-down)'.
```

Table 3 - Top 20 most widely spread external modules

The following tables shows external modules that are used by many different internal modules with the highest number of artifacts first.

Statistics like minimum, maximum, average, median and standard deviation are provided for the number of internally exported elements (function, class, ...) and the external declarations they use for every external module.

The intuition behind that is to find external modules and the external declarations they provide that are used in a widely spread manner. This can help to distinguish widely used libraries and frameworks from external modules that are used for specific tasks. It can also be used to find external modules that are used sparsely regarding internal modules but where many different external declarations are used.

Refactoring with Hexagonal architecture in mind can be considered for non-framework external modules that are used for very specific tasks and that are used in many different internal locations. This makes the internal code more robust against changes of these external modules or it is easier to update and migrate to newer versions of them.

External modules that are only used in very few internal locations overall might be considered for removal if they are easy to replace with a similar library that is already used more often. Or they might also simply be replaced by very few lines of code. Replacing libraries with own code isn't recommended when you need to write a lot of code or for external modules that provide security relevant implementations (encryption, sanitizers, ...), because they will be tracked and maintained globally and security updates need to be adopted fast.

Only the top 20 entries are shown. The whole table can be found in the following CSV report: `External_module_usage_spread_for_Typescript`

Columns:

- *externalModuleName* is the name of the external package prepended by its namespace if given. Example: "@types/react"

external package.

- *numberOfInternalModules* is the number of internal modules that are using that external module
- *[min,max,med,avg,std]NumberOfUsedExternalDeclarations* provide statistics for all internal modules and how their usage of the declarations provided by the external module are distributed. This provides an indicator on how strong the coupling to the external module is. For example, if many (high sum) elements provided by that external module are used constantly (low std), a higher coupling can be presumed. If there is only one (sum) element in use, this could be an indicator for an external module that could get replaced or that there is just one central entry point for it.
- *[min/max/med/avg/std]NumberOfInternalElements* provide statistics for all internal modules and how their usage of the external module is distributed across their internal elements. This provides an indicator on how widely an external module is spread across internal elements and if there are great differences between internal modules (high standard deviation) or not.
- *[min/max/med/avg/std]NumberOfInternalElementsPercentage* is similar to [min/max/med/avg/std]NumberOfUsedExternalDeclarations but provides the value in percent in relation to the total number of internal elements per internal module.
- *internalModuleExamples* some examples of included internal modules for debugging

	externalModuleName	numberOfInternalModules	sumNumberOfUsedExternalDeclarations	minNumberOfUsedExternalDeclarations	maxNumberOfUsedExternalDeclara
0	@types/react	5	38	1	
1	@remix-run/router	4	131	6	
2	@types/react-native	1	10	10	
3	@ungap/url-search-params	1	1	1	

Table 3a - Top 20 most widely spread external packages - number of internal modules

This table shows the top 20 most widely spread external packages focussing on the spread across the number of internal modules.

	externalModuleName	numberOfInternalModules	minNumberOfInternalElements	maxNumberOfInternalElements	medNumberOfInternalElements	avgNumberOfInterna
0	@types/react	5	1	24	3.0	
1	@remix-run/router	4	3	23	5.5	
2	@types/react-native	1	5	5	5.0	
3	@ungap/url-search-params	1	2	2	2.0	

Table 3b - Top 20 most widely spread external packages - percentage of internal modules

This table shows the top 20 most widely spread external packages focussing on the spread across the percentage of internal modules.

	externalModuleName	numberOfInternalModules	minNumberOfInternalElementsPercentage	maxNumberOfInternalElementsPercentage	medNumberOfInternalElements
0	@types/react	5	8.333333	200.000000	
1	@remix-run/router	4	25.000000	191.666667	
2	@types/react-native	1	41.666667	41.666667	
3	@ungap/url-search-params	1	16.666667	16.666667	

Table 3c - Top 20 most widely spread external packages - number of internal elements

This table shows the top 20 most widely spread external packages focussing on the spread across the number of internal elements.

	externalModuleName	numberOfInternalModules	minNumberOfInternalElements	maxNumberOfInternalElements	medNumberOfInternalElements	avgNumberOfInterna
0	@types/react	5	1	24	3.0	
1	@remix-run/router	4	3	23	5.5	
2	@types/react-native	1	5	5	5.0	
3	@ungap/url-search-params	1	2	2	2.0	

Table 3d - Top 20 most widely spread external packages - percentage of internal elements

This table shows the top 20 most widely spread external packages focussing on the spread across the percentage of internal elements.

	externalModuleName	numberOfInternalModules	minNumberOfInternalElementsPercentage	maxNumberOfInternalElementsPercentage	medNumberOfInternalElements
0	@types/react	5	8.333333	200.000000	
1	@remix-run/router	4	25.000000	191.666667	
2	@types/react-native	1	41.666667	41.666667	
3	@ungap/url-search-params	1	16.666667	16.666667	

Table 3 Chart 1a - Most widely spread external module in % by internal elements (more than 0.5% overall)

External modules that are used less than 0.5% are grouped into the name "others" to get a cleaner chart with the most significant external module.

```
<Figure size 640x480 with 0 Axes>
```

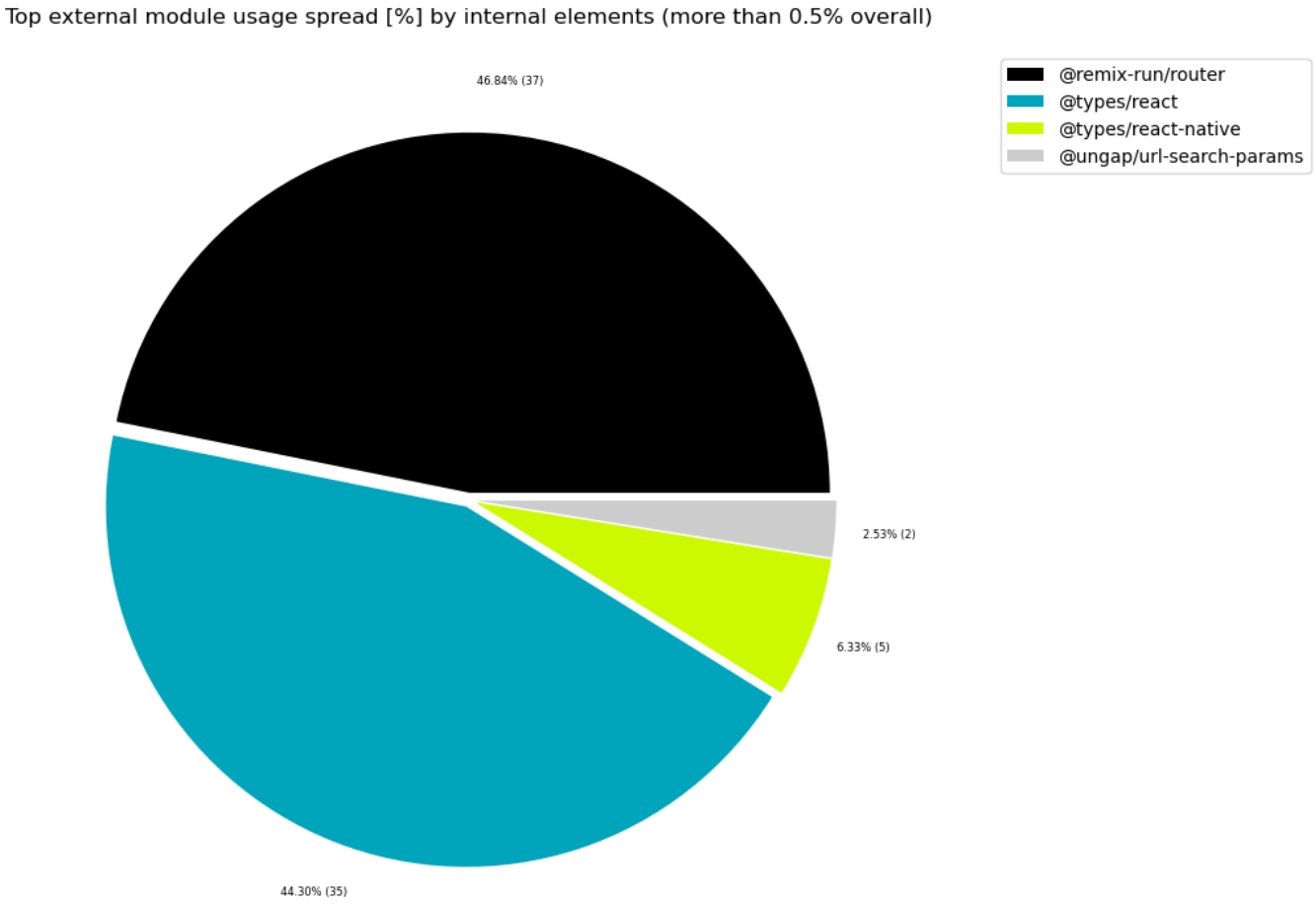


Table 3 Chart 1b - Most widely spread external modules in % by types (less than 0.5% overall "others" drill-down)

Shows the lowest (less than 0.5% overall) most widely spread external modules. Therefore, this plot breaks down the "others" slice of the pie chart above. Values under 0.3% from that will be grouped into "others" to get a cleaner plot.

```
No data to plot for title 'Top external module usage spread [%] by internal elements
(less than 0.7% overall "others" drill-down)'.
```

Table 3 Chart 2a - Most widely spread external modules in % by internal modules (more than 0.5% overall)

External modules that are used less than 0.5% are grouped into "others" to get a cleaner chart containing the most significant external modules.

```
<Figure size 640x480 with 0 Axes>
```

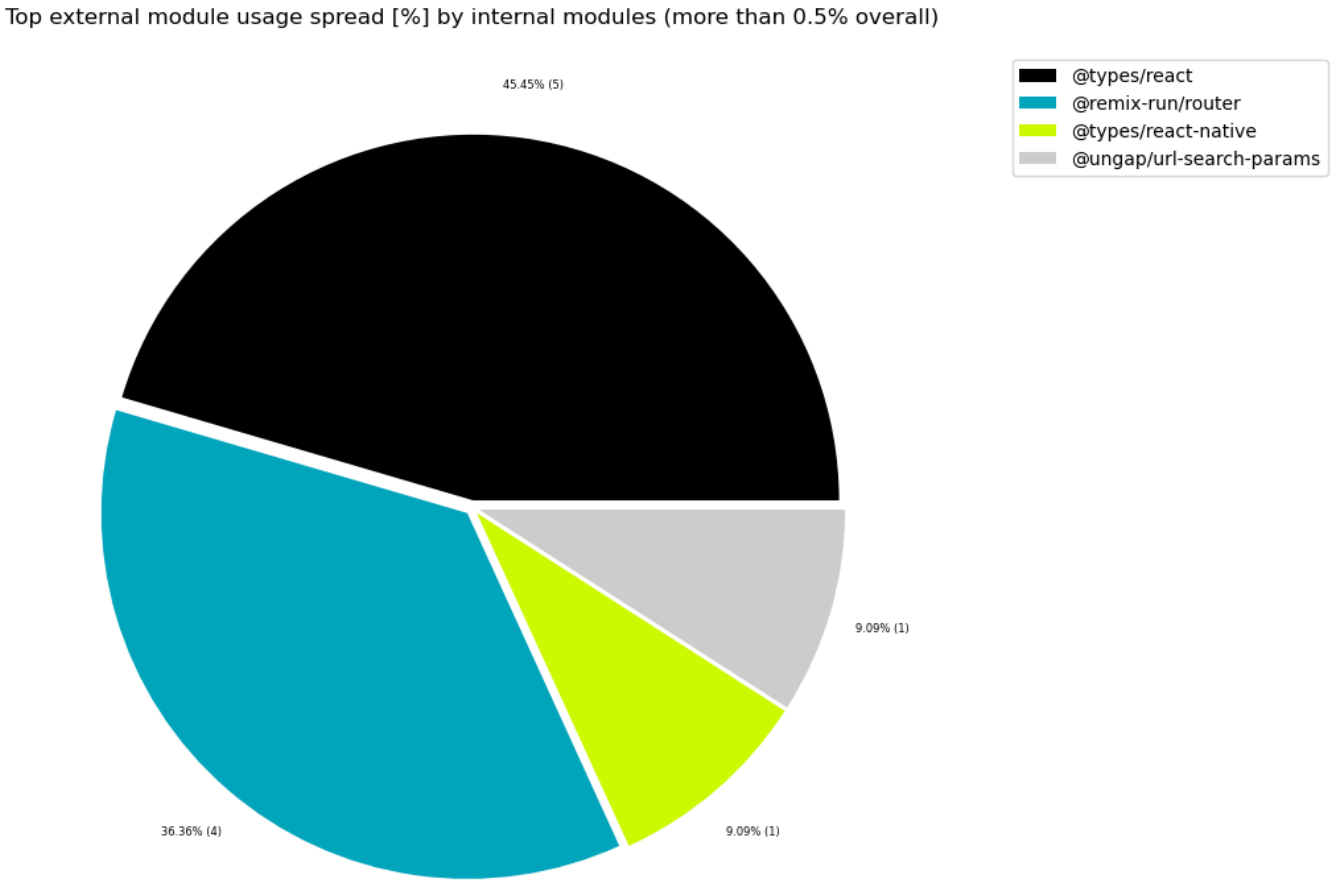


Table 3 Chart 2b - Most widely spread external modules in % by internal modules (less than 0.5% overall "others" drill-down)

Shows the lowest (less than 0.5% overall) most widely spread external modules. Therefore, this plot breaks down the "others" slice of the pie chart above. Values under 0.3% from that will be grouped into "others" to get a cleaner plot.

```
No data to plot for title 'Top external module usage spread [%] by internal modules
(less than 0.7% overall "others" drill-down)'.
```

Table 4 - Top 20 most widely spread external namespaces

This table shows external namespaces that are used by different internal modules with the most used first.

Statistics like minimum, maximum, average, median and standard deviation are provided for the number of internally exported elements (function, class, ...) and the external declarations they use for every external namespace.

The intuition behind that is to find external namespaces that are used in a widely spread manner. This can help to distinguish widely used libraries and frameworks from external modules that are used for specific tasks. It can also be used to find external modules that are used sparsely regarding internal modules but where many different external declarations are used.

Refactoring with a Hexagonal architecture in mind can be considered for non-framework external namespaces that are used for very specific tasks and that are used in many different internal locations. This makes the internal code more robust against changes of these external modules or it is easier to update and migrate to newer versions of them.

External namespaces that are only used in very few internal locations overall might be considered for removal if they are easy to replace with a similar library that is already used more often. Or they might also simply be replaced by very few lines of code. Replacing libraries with own code isn't recommended when you need to write a lot of code or for external modules that provide security relevant implementations (encryption, sanitizers, ...), because they will be tracked and maintained globally and security updates need to be adopted fast.

Only the top 20 entries are shown. The whole table can be found in the following CSV report: `External_namespace_usage_spread_for_Typescript`

Columns:

- *externalModuleNamespace* identifies the external namespace for at least on external module in use. All other columns contain aggregated data for it.
- *numberOfInternalModules* is the number of internal modules that are using that external module
- *[min,max,med,avg,std]NumberOfUsedExternalDeclarations* provide statistics for all internal modules and how their usage of the declarations provided by the external module are distributed. This provides an indicator on how strong the coupling to the external module is. For example, if many (high sum) elements provided by that external module are used constantly (low std), a higher coupling can be presumed. If there is only one (sum) element in use, this could be an indicator for an external module that could get replaced or that there is just one central entry point for it.
- *[min/max/med/avg/std]NumberOfInternalElements* provide statistics for all internal modules and how their usage of the external module is distributed across their internal elements. This provides an indicator on how widely an external module is spread across internal elements and if there are great differences between internal modules (high standard deviation) or not.
- *[min/max/med/avg/std]NumberOfInternalElementsPercentage* is similar to [min/max/med/avg/std]NumberOfUsedExternalDeclarations but provides the value in percent in relation to the total number of internal elements per internal module.
- *internalModuleExamples* some examples of included internal modules for debugging

Table 4 - Top 20 most widely spread external namespaces

This table shows external namespaces that are used by different internal modules with the most used first.

Statistics like minimum, maximum, average, median and standard deviation are provided for the number of internally exported elements (function, class, ...) and the external declarations they use for every external namespace.

The intuition behind that is to find external namespaces that are used in a widely spread manner. This can help to distinguish widely used libraries and frameworks from external modules that are used for specific tasks. It can also be used to find external modules that are used sparsely regarding internal modules but where many different external declarations are used.

Refactoring with a Hexagonal architecture in mind can be considered for non-framework external namespaces that are used for very specific tasks and that are used in many different internal locations. This makes the internal code more robust against changes of these external modules or it is easier to update and migrate to newer versions of them.

External namespaces that are only used in very few internal locations overall might be considered for removal if they are easy to replace with a similar library that is already used more often. Or they might also simply be replaced by very few lines of code. Replacing libraries with own code isn't recommended when you need to write a lot of code or for external modules that provide security relevant implementations (encryption, sanitizers, ...), because they will be tracked and maintained globally and security updates need to be adopted fast.

Only the top 20 entries are shown. The whole table can be found in the following CSV report: `External_namespace_usage_spread_for_Typescript`

Columns:

- *externalModuleNamespace* identifies the external namespace for at least on external module in use. All other columns contain aggregated data for it.
- *numberOfInternalModules* is the number of internal modules that are using that external module
- *[min,max,med,avg,std]NumberOfUsedExternalDeclarations* provide statistics for all internal modules and how their usage of the declarations provided by the external module are distributed. This provides an indicator on how strong the coupling to the external module is. For example, if many (high sum) elements provided by that external module are used constantly (low std), a higher coupling can be presumed. If there is only one (sum) element in use, this could be an indicator for an external module that could get replaced or that there is just one central entry point for it.
- *[min/max/med/avg/std]NumberOfInternalElements* provide statistics for all internal modules and how their usage of the external module is distributed across their internal elements. This provides an indicator on how widely an external module is spread across internal elements and if there are great differences between internal modules (high standard deviation) or not.

- *[min/max/med/avg/std]NumberOfInternalElementsPercentage* is similar to [min/max/med/avg/std]NumberOfUsedExternalDeclarations but provides the value in percent in relation to the total number of internal elements per internal module.
- *internalModuleExamples* some examples of included internal modules for debugging

	externalModuleNamespace	numberOfInternalModules	sumNumberOfUsedExternalDeclarations	minNumberOfUsedExternalDeclarations	maxNumberOfUsedExternalD
0	@types	5	48	1	
1	@remix-run	4	131	6	
2	@ungap	1	1	1	

Table 4 Chart 1a - Most widely spread external namespaces in % by internal element (less than 0.5% overall)

External namespaces that are used less than 0.5% are grouped into "others" to get a cleaner chart containing the most significant external namespaces and how ofter they are called in percent.

```
<Figure size 640x480 with 0 Axes>
```

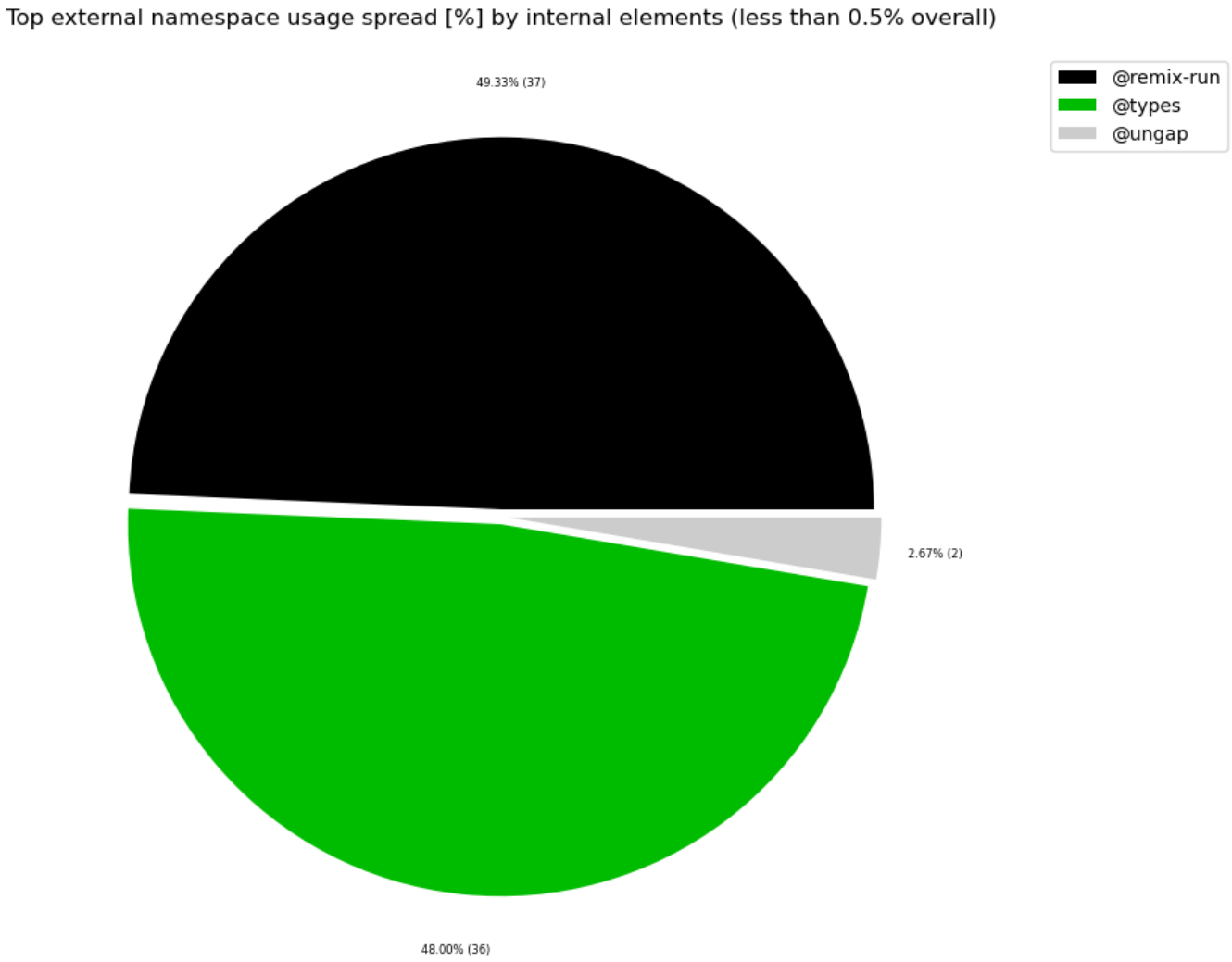


Table 4 Chart 1b - Most widely spread external namespaces in % by internal element (less than 0.5% overall "others" drill-down)

Shows the lowest (less than 0.5% overall) most widely spread external namespaces. Therefore, this plot breaks down the "others" slice of the pie chart above. Values under 0.3% from that will be grouped into "others" to get a cleaner plot.

```
No data to plot for title 'Top external namespace usage spread [%] by internal elemen
ts (less than 0.7% overall "others" drill-down)'.
```

Table 4 Chart 2a - Most widely spread external namespace in % by internal modules (more than 0.5% overall)

External namespaces that are used less than 0.5% are grouped into "others" to get a cleaner chart containing the most significant external namespaces and how ofter they are called in percent.

```
<Figure size 640x480 with 0 Axes>
```

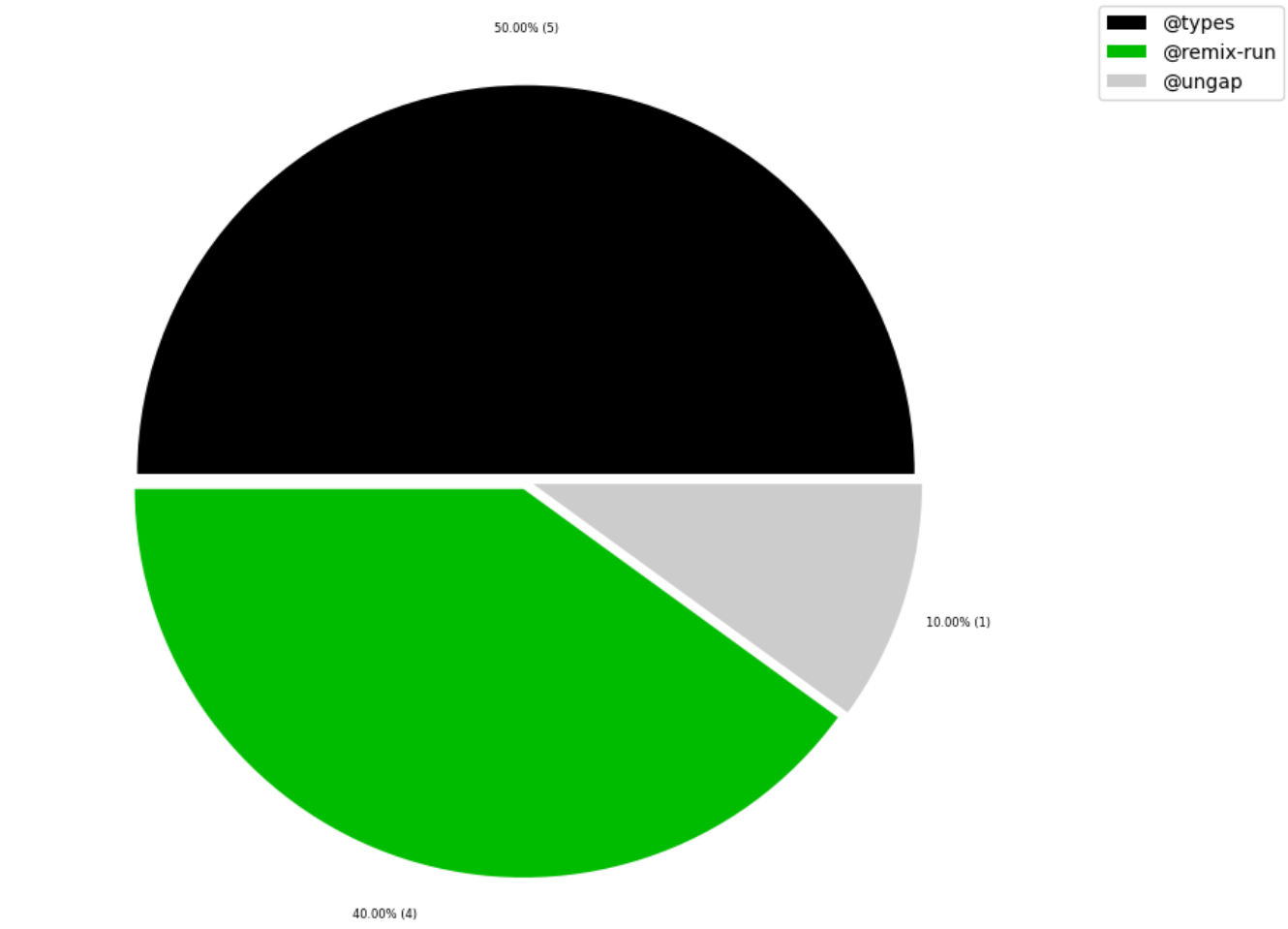


Table 4 Chart 2b - Most widely spread external namespace in % by internal modules (less than 0.5% overall "others" drill-down)

Shows the lowest (less than 0.5% overall) most widely spread external namespaces. Therefore, this plot breaks down the "others" slice of the pie chart above. Values under 0.3% from that will be grouped into "others" to get a cleaner plot.

```
No data to plot for title 'Top external namespace usage spread [%] by internal module
s (less than 0.7% overall "others" drill-down)'.
```

Table 4 Chart 3a - External namespaces with the most used declarations in % (more than 0.5% overall)

External namespaces that are used less than 0.5% are grouped into "others" to get a cleaner chart containing the most significant external namespaces and how ofter they are called in percent.

```
<Figure size 640x480 with 0 Axes>
```

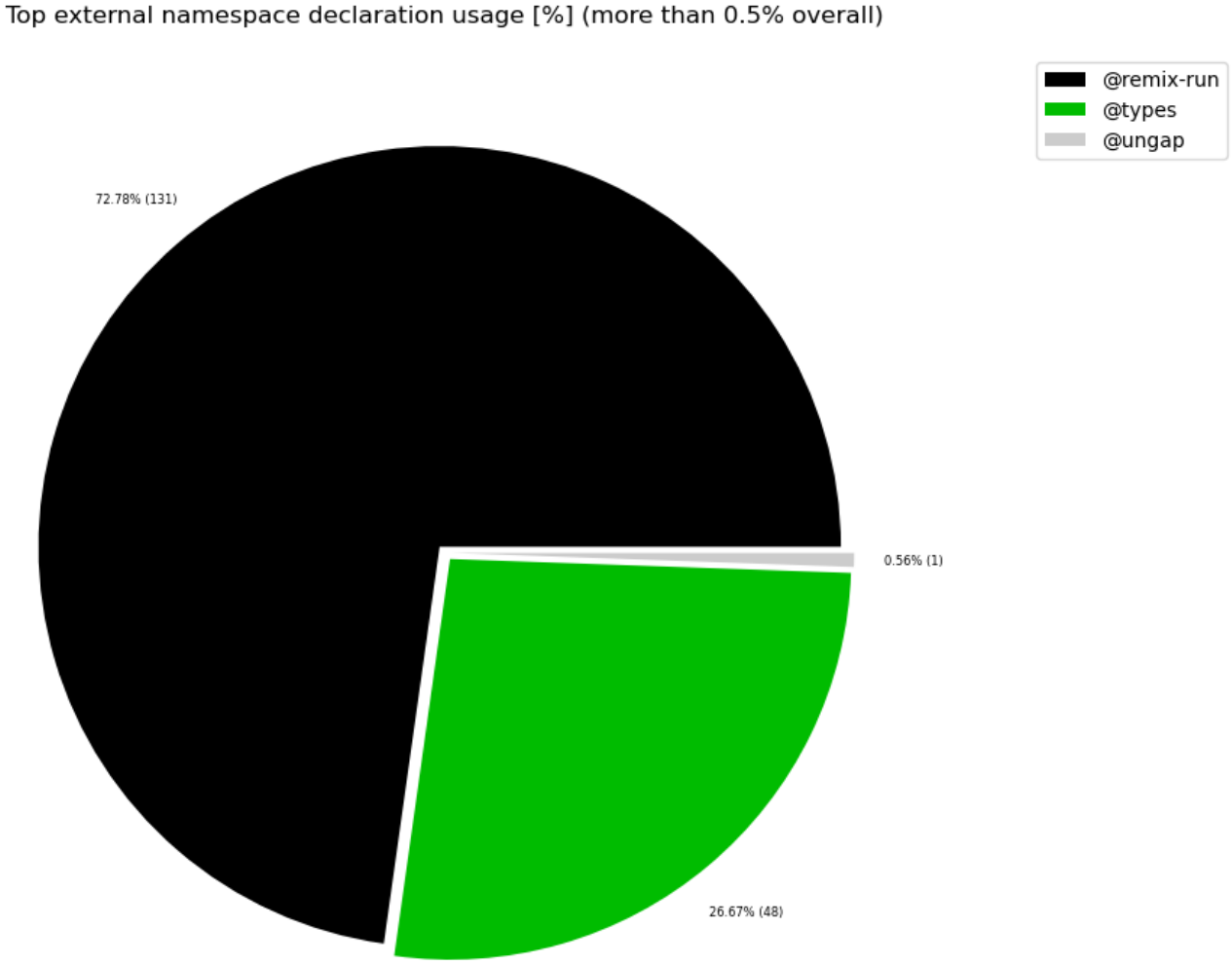


Table 4 Chart 3b - External namespaces with the most used declarations in % (less than 0.5% overall "others" drill-down)

Shows the lowest (less than 0.5% overall) external namespaces with the most used declarations. Therefore, this plot breaks down the "others" slice of the pie chart above. Values under 0.3% from that will be grouped into "others" to get a cleaner plot.

```
No data to plot for title 'Top external namespace declaration usage (less than 0.7% o
verall "others" drill-down)'.
```

Table 5 - Top 20 least used external modules overall

This table identifies external modules that aren't used very often. This could help to find libraries that aren't actually needed or maybe easily replaceable. Some of them might be used sparsely on purpose for example as an adapter to an external library that is actually important. Thus, decisions need to be made on a case-by-case basis.

Only the last 20 entries are shown. The whole table can be found in the following CSV report: `External_module_usage_overall_for_Typescript`

Columns:

- *externalModuleName* identifies the external package as described above
- *numberOfExternalDeclarationCalls* includes every invocation or reference to the declarations in the external module

	externalModuleName	numberOfExternalDeclarationCalls
0	@ungap/url-search-params	4
1	@types/react-native	28
2	@types/react	195
3	@remix-run/router	390

Table 6 - External usage per internal module sorted by highest external element usage rate descending

The following table shows the most used external packages separately for each artifact including external annotations. The results are sorted by the artifacts with the highest external type usage rate descending.

The intention of this table is to find artifacts that use a lot of external dependencies in relation to their size and get all the external packages and their usage.

Only the first 40 entries are shown. The whole table can be found in the following CSV report: `External_module_usage_per_internal_module_sorted_for_Typescript`

Columns:

- *internalModuleName* is the internal module that uses the external one. Both are used here as a group for a more detailed analysis.
- *externalModuleName* is the external module prepended by its namespace if given. Example: "@types/react"
- *numberOfExternalDeclarationCaller* is the count of distinct internal elements in the internal module that call the external module
- *numberOfExternalDeclarationCalls* is the count of how often the external module is called within the internal module
- *numberOfAllElementsInInternalModule* is the total count of all exported elements of the internal module

- *numberOfAllExternalDeclarationsUsedInInternalModule* is the total count of all distinct external declarations used in the internal module
- *numberOfAllExternalModulesUsedInInternalModule* is the total count of all distinct external modules used in the internal module
- *externalDeclarationRate* is the numberOfAllExternalDeclarationsUsedInInternalModule / numberOfAllElementsInInternalModule * 100 of the internal module for all external modules
- *externalDeclarationNames* contains a list of actually used external declarations

Table 6a - External module usage per internal module sorted by highest external element usage rate descending

	internalModuleName	externalModuleName	numberOfExternalDeclarationCaller	numberOfExternalDeclarationCalls	numberOfAllElementsInInternalModule	numberO
0	server	@remix-run/router	74	128	6	
1	server	@types/react	6	30	6	
2	react-router-dom	@remix-run/router	282	498	63	
3	react-router-dom	@types/react	166	496	63	
4	react-router-native	@types/react-native	22	49	17	
5	react-router-native	@remix-run/router	18	41	17	
6	react-router-native	@types/react	18	49	17	
7	react-router-native	@ungap/url-search-params	4	8	17	
8	react-router	@remix-run/router	16	20	7	
9	react-router	@types/react	2	6	7	
10	App	@types/react	1	1	1	

Table 6b - External namespace usage per internal module sorted by highest external element usage rate descending

	internalModuleName	externalNamespaceName	numberOfExternalDeclarationCaller	numberOfExternalDeclarationCalls	numberOfAllElementsInInternalModule	numb
0	server	@remix-run	74	128	6	
1	server	@types	6	30	6	
2	react-router-dom	@remix-run	282	498	63	
3	react-router-dom	@types	166	496	63	
4	react-router-native	@types	40	98	17	
5	react-router-native	@remix-run	18	41	17	
6	react-router-native	@ungap	4	8	17	
7	react-router	@remix-run	16	20	7	
8	react-router	@types	2	6	7	
9	App	@types	1	1	1	

Table 6c - Top 15 used external modules with the internal modules that use them the most

The following table uses pivot to show the internal modules in columns, the external modules in rows and the number of internal elements using them as values.

internalModuleName externalModuleName	react-router-dom	server	react-router-native	react-router	App
@remix-run/router	282	74	18	16	0
@types/react	166	6	18	2	1
@types/react-native	0	0	22	0	0
@ungap/url-search-params	0	0	4	0	0

Table 6d - Top 15 used external namespaces with the internal modules that use them the most

The following table uses pivot to show the internal modules in columns, the external namespaces in rows and the number of internal elements using them as values.

internalModuleName externalNamespaceName	react-router-dom	server	react-router-native	react-router	App
@remix-run	282	74	18	16	0
@types	166	6	40	2	1
@ungap	0	0	4	0	0

Table 6e - External usage per internal module and its elements

This table lists internal elements and the modules they belong to that use many different external declarations of a specific external module.

	internalModuleName	numberOfAllExternalDeclarationsUsedInInternalModule	numberOfAllElementsInInternalModule
0	server	29	6
2	react-router-dom	117	63
4	react-router-native	24	17
8	react-router	9	7
10	App	1	1

Table 6 Chart 1 - Top 15 external dependency using artifacts and their external packages stacked

The following chart shows the top 15 external package using artifacts and breaks down which external packages they use in how many different internal packages with stacked bars.

Note that every external dependency is counted separately so that if on internal package uses two external packages it will be displayed for both and so stacked twice.

```
<Figure size 640x480 with 0 Axes>
```

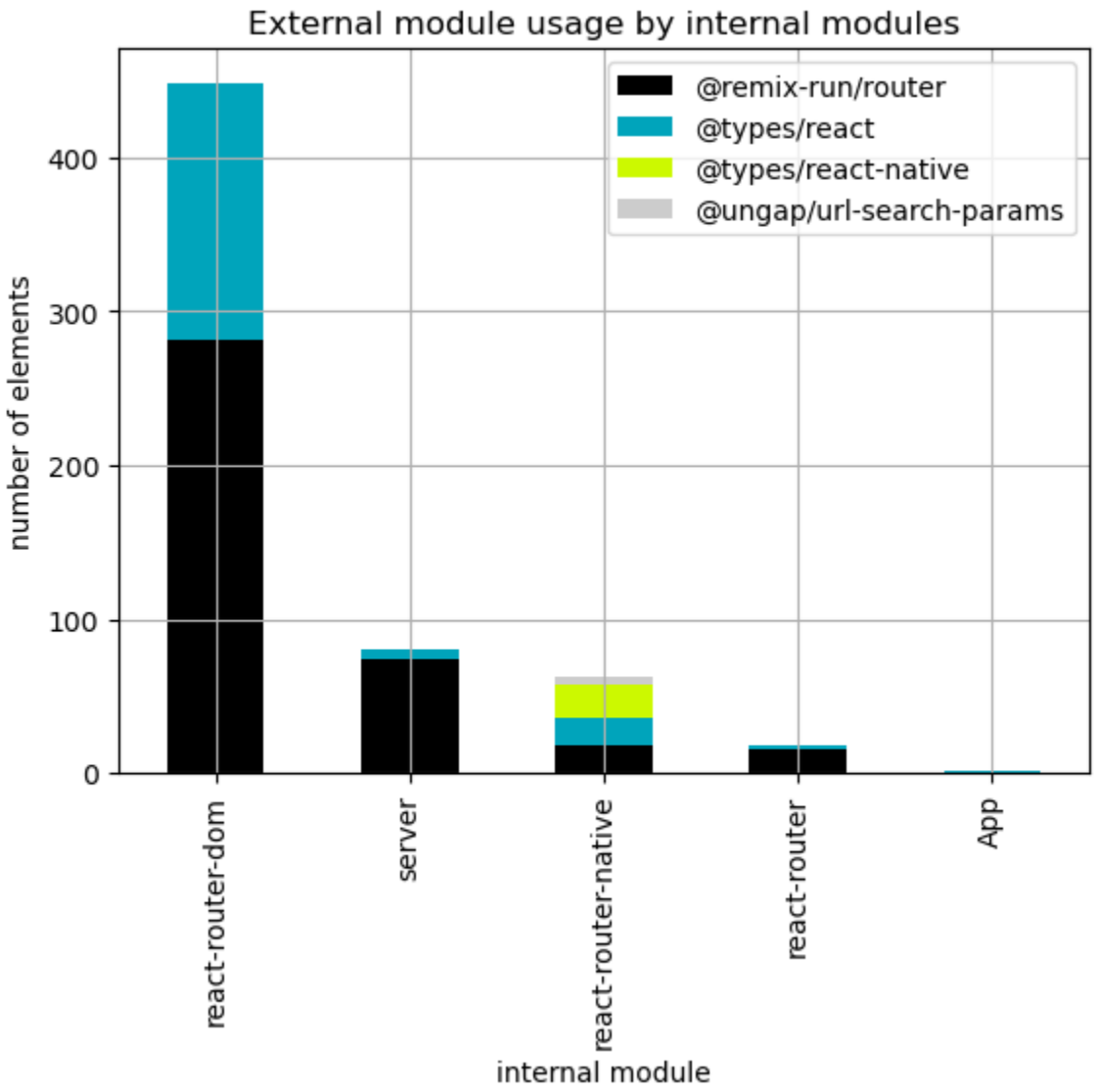


Table 6 Chart 2 - Top 15 external dependency using artifacts and their external packages (first 2 levels) stacked

The following chart shows the top 15 external package using artifacts and breaks down which external packages (first 2 levels) are used in how many different internal packages with stacked bars.

Note that every external dependency is counted separately so that if on internal package uses two external packages it will be displayed for both and so stacked twice.

```
<Figure size 640x480 with 0 Axes>
```

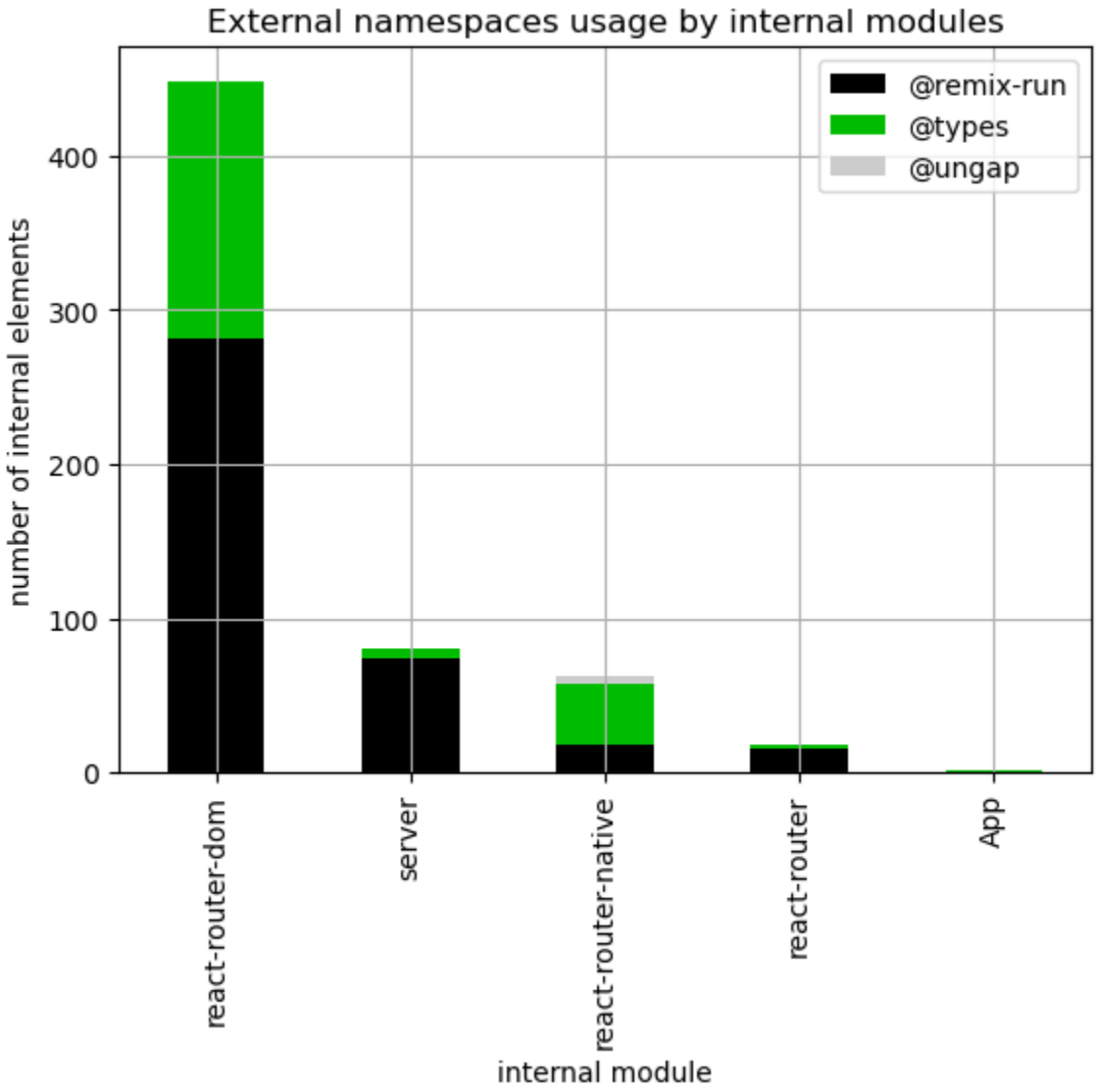


Table 7 - External module usage distribution per internal element

This table shows how many internal elements use one external module, how many use two, etc. . This gives an overview of the distribution of external module calls and the overall coupling to external libraries. The higher the count of distinct external modules the lower should be the count of internal elements that use them. More details about which types have the highest external package dependency usage can be in the tables 4 and 5 above.

Only the last 40 entries are shown. The whole table can be found in the following CSV report: `External_module_usage_per_internal_module_distribution_for_Typescript`

Columns:

- *internalModuleName* is the internal module that uses at least one external module. All other columns refer to it.
- *numberOfAllInternalElements* the total number of all elements that are exported by the internal module
- *externalModuleCount* is the number of distinct external modules used by the internal module

- *internalElementCount* is the number of distinct internal elements that use at least one external one
- *internalElementsCallingExternalRate* is internalElementCount / numberOfAllInternalElements * 100 (in %)

	internalModuleName	numberOfAllInternalElements	externalModuleCount	internalElementCount	internalElementsCallingExternalRate
0	react-router-dom	63	2	29	46.031746
1	react-router-native	17	4	10	58.823529
2	server	6	2	6	100.000000
3	react-router	7	2	3	42.857143
4	App	1	1	1	100.000000

Table 8 - External package usage aggregated

This table lists all artifacts and their external package dependencies usage aggregated over internal packages.

The intention behind this is to find artifacts that use an external dependency across multiple internal packages. This might be intended for frameworks and standardized libraries and helps to quantify how widely those are used. For some external dependencies it might be beneficial to only access it from one package and provide an abstraction for internal usage following a Hexagonal architecture. Thus, this table may also help in finding application for the Hexagonal architecture or similar approaches (Domain Driven Design Anti Corruption Layer). After all it is easier to update or replace such external dependencies when they are used in specific areas and not all over the code.

Only the last 40 entries are shown. The whole table can be found in the following CSV report: `External_module_usage_per_internal_module_aggregated_for_Typescript`

Columns:

- *internalModuleName* that contains the type that calls the external package
- *internalModuleElementsCount* is the total count of packages in the internal module
- *numberOfExternalModules* the number of distinct external packages used
- *[min,max,med,avg,std]NumberOfInternalModules* provide statistics based on each external package and its package usage within the internal module
- *[min,max,med,avg,std]NumberOfInternalElements* provide statistics based on each external package and its type usage within the internal module
- *[min,max,med,avg,std]NumberOfTypePercentage* provide statistics in % based on each external package and its type usage within the internal module in respect to the overall count of packages in the internal module
- *numberOfinternalElements* in the internal module where the *numberOfExternalModules* applies
- *numberOfTypesPercentage* in the internal module where the *numberOfExternalModules* applies in %

Table 8a - External module usage aggregated - count of internal modules

	internalModuleName	internalModuleElementsCount	numberOfExternalModules	minNumberOfInternalModules	medNumberOfInternalModules	avgNumberOfInternalM
0	App	1	1	1	1.0	
1	react-router	7	2	1	1.0	
2	react-router-dom	63	2	1	1.0	
3	react-router-native	17	4	1	1.0	
4	server	6	2	1	1.0	

Table 8b - External module usage aggregated - count of internal elements

	internalModuleName	internalModuleElementsCount	numberOfExternalModules	minNumberOfInternalElements	medNumberOfInternalElements	avgNumberOfInterna
0	App	1	1	1	1.0	
1	react-router	7	2	1	2.0	
2	react-router-dom	63	2	23	23.5	
3	react-router-native	17	4	2	5.5	
4	server	6	2	3	4.0	

Table 8c - External module usage aggregated - percentage of internal elements

	internalModuleName	internalModuleElementsCount	numberOfExternalModules	minNumberOfInternalElementsPercentage	medNumberOfInternalElementsPercentage
0	App	1	1	100.000000	100.000000
1	react-router	7	2	14.285714	28.571429
2	react-router-dom	63	2	36.507937	37.301587
3	react-router-native	17	4	11.764706	32.352941
4	server	6	2	50.000000	66.666667

Table 8 Chart 1 - External module usage - max percentage of internal types

This chart shows per internal module the maximum percentage of internal packages (compared to all packages in that internal module) that use one specific external package.

Example: One internal module might use 10 external packages where 7 of them are used in one internal package, 2 of them are used in two packages and one external dependency is used in 5 packages. So for this internal module there will be a point at x = 10 (external packages used by the internal module) and 5 (max internal packages). Instead of the count the percentage of internal packages compared to all packages in that internal module is used to get a normalized plot.

```
<Figure size 640x480 with 0 Axes>
```

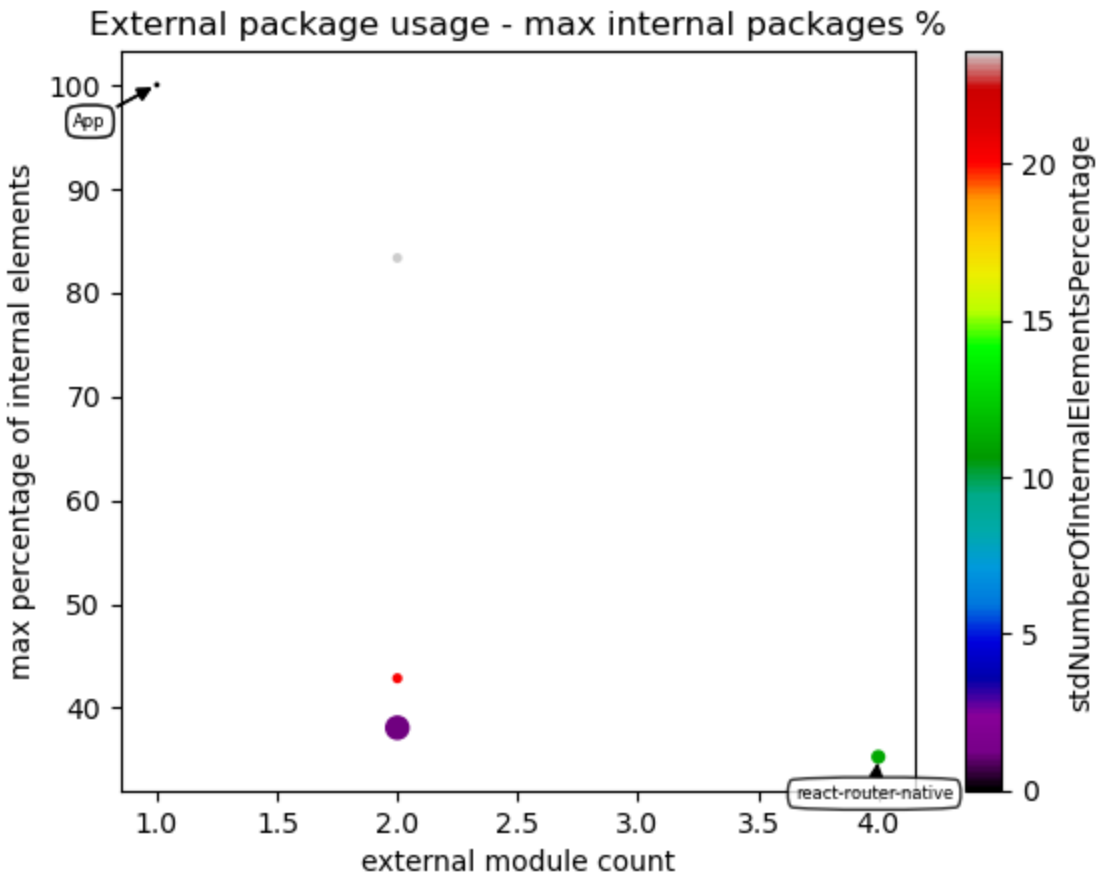
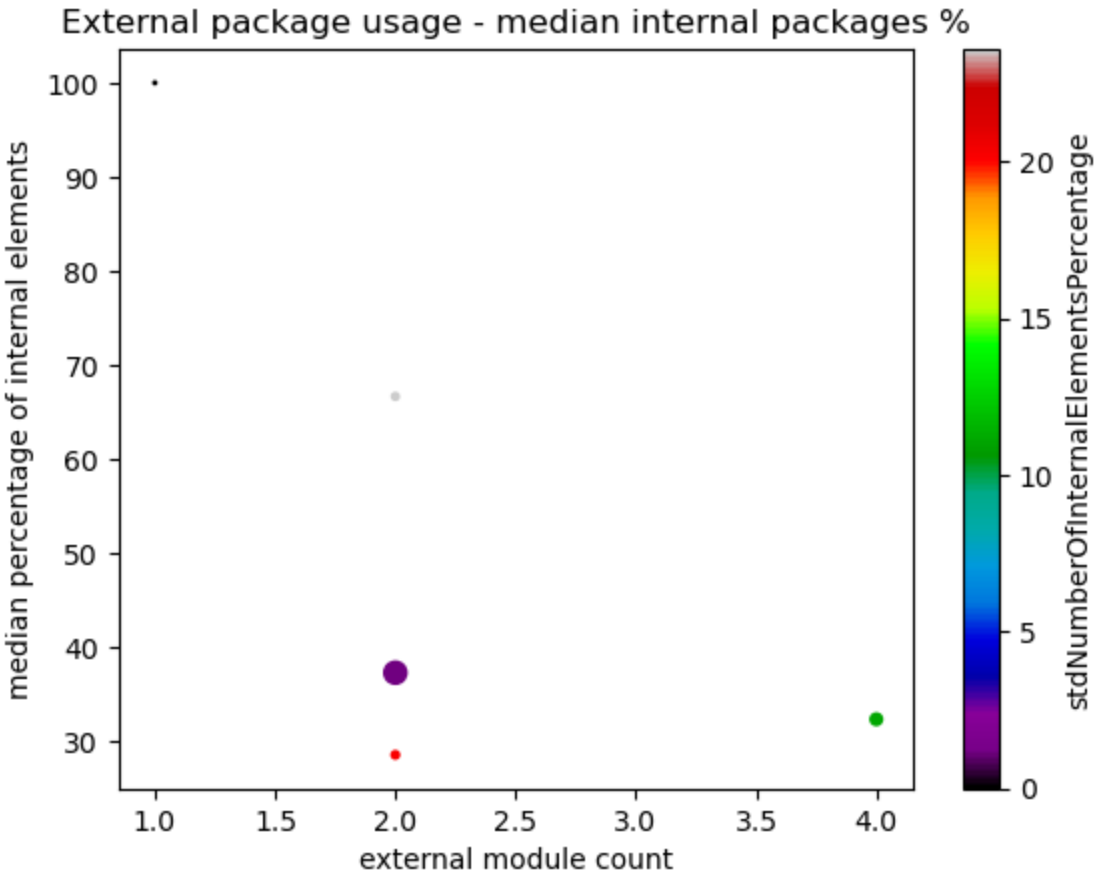


Table 8 Chart 2 - External package usage - median percentage of internal types

This chart shows per internal module the median (0.5 percentile) of internal packages (compared to all packages in that internal module) that use one specific external package.

Example: One internal module might use 9 external packages where 3 of them are used in 1 internal package, 3 of them are used in 2 package and the last 3 ones are used in 3 packages. So for this internal module there will be a point at x = 10 (external packages used by the internal module) and 2 (median internal packages). Instead of the count the percentage of internal packages compared to all packages in that internal module is used to get a normalized plot.

```
<Figure size 640x480 with 0 Axes>
```


External package usage - median internal packages %
median percentage of internal elements
100
90
80
70
60
50
40
30
1.0
1.5
2.0
2.5
3.0
3.5
4.0
external module count
stdNumberOfInternalElementsPercentage
0
5
10
15
20