In this course unit [CM-WWEB] a structured software development process of (web) applications is illustrated. The application development makes use of current software development concepts, such as Behavior-Driven Development (BDD), Domain-Driven Design (DDD), microservice architectures including a systematic design of the web application programming interfaces (API) of the microservices.

(1) In the first chapter a rough overview of the relevant development concepts is given and the application development process using these concepts is introduced.

(2) The organization of software development projects is supported by a specific set of project management and version control tools which are described in this chapter.

(3) This chapter deals with the first two phases of the development process and describes the way how the analysis and the design of a web application are carried out.

(4) In the subsequent implementation phase the functionality as is was analyzed and designed is coded based on a microservice architecture and frameworks supporting the frontend and backend implementation. In the deployment phase the tested application is deployed in a Docker-based execution environment. Finally, a web application dealing with the management of todo lists is introduced to demonstrate the use of Spring Boot and related technologies.

API  Application Programming Interface
BDD  Behavior-Driven Development
C&M  Cooperation & Management
DDD  Domain-Driven Design
KIT  Karlsruhe Institute of Technology

[CM-WWEB] Cooperation & Management: WEB APPLICATION DEVELOPMENT, WASA Course Unit. https://team.kit.edu/sites/cm-tm/Mitglieder/2-1.WASA
The motivation behind Behavior-Driven Design (BDD) is to build and deliver better software [Sm15:3].

(1) According to a number of studies, nearly half of all software projects fail to deliver in some significant way.
(1.1) In a CHAOS Report 2011 published by the Standish Group in 42% of the software projects one of or more of these problems occurred.
(1.2) This results in billions of dollars in wasted effort.

(2) The two goals center around the questions of WHAT to develop and HOW to develop.
(2.1) This goal describing the WHAT is shown by the vertical (y) axis in the figure.
(2.2) This goal describing the HOW is shown by the horizontal (x) axis in the figure.

(3) BDD encourages business analysts, software developers, and testers to collaborate more closely by enabling them to express requirements in a more testable way, in a form that both the development team and business stakeholders can easily understand.

BDD Behavior-Driven Development

https://team.kit.edu/sites/cm-tm/Mitglieder/2-4.Literatur/BDD
The figure introduces the different phases that have to be carried out to develop software systems, esp. Web applications. Besides the programming language, natural language and several description languages are needed to develop a web application in a sound and systematic way.

(1) Iterative in this context means that the phases are not completely passed in a sequential order but jumps to a the preceding phase are the usual case.

(Feature Specification, Domain Model) These are the main artifacts from BDD (features) and DDD (domain model).

(Unified Modeling Language, eXtensible Markup Language, Hypertext Markup Language) are widely used description languages in the development process of web applications. Gherkin is a language which is used in the BDD approach.

(OpenAPI) This is a standardized language by which an API (Application Programming Interface) of a microservice based on REST (REpresentational State Transfer) can be specified.
This figure illustrates the various architectural concepts and their interrelationships. The arrow symbolizes an superior influence. This is due to the fact that the definitions are not standardized and there are more than 30 definitions [CMU-SA] for the software architecture of the CMU alone.

(Software Architecture) The software architecture [VA+08] takes a logical view on the systems by dividing it into logical components. These decisions are difficult to reverse in retrospective.

(System Architecture) The system architecture [VA+08], also known as the physical view, describes the structure of the system, which consists of software and hardware components. Furthermore, their properties and relationships to each other as well as to their environment and other systems are shown. A possible modeling type is the deployment diagram.

(Implementation Architecture) The implementation architecture [RM+06] describes the structure of the system from a technical point of view. This includes packages, libraries and frameworks.

(Macro Architecture) The macro architecture [Ot13] is used to describe the principles of the subsystems, e.g. microservices or modules, on a black box level. This type of architecture only concerns cross-subsystem aspects and does not determine the internal structure of the subsystems. Exemplary principles are "Shared Nothing" or the implementation of a RESTful architecture.

(Micro Architecture) The micro architecture [Ot13] deals with the internal structure of a single subsystem. For organizational or management reasons, agreements such as the use of common basic technologies can be made.

(Architectural Style) Architectural styles [RH08] are application-independent solution principles that are used throughout the project. These can be assigned to the following categories: communication (e.g. message bus oriented), deployment (e.g. N-tier architecture) and structure (e.g. layer architecture).

(Design Pattern) Design patterns [RH08] provide program code-specific patterns (e.g. visitor pattern) for recurring problems.

(Architectural Pattern) Architectural patterns [RH08] are solutions to recurring problems which, unlike design patterns, affect several architectural elements.

(Component / Deployment / Class Diagram) Deployment View) The Unified Modeling Language (UML) provides different types of diagrams to specify the different types of architecture as models.


(1) The fundamental principles of each plane are as follows:
(Network Plane) Web applications run on several independent systems which heavily use Internet technologies for communication purposes [Ta06, KR05].
(System Plane) Internet protocols (such as the web protocol HTTP) are based on a client-server principle. The system plane is characterized by different operating systems leading to heterogeneous IT infrastructures.
(Application Plane) Web-based applications are located on the application plane. The term "web-based" means that the presentation layer is implemented by a standard web browser.
(DATA, BUSINESS LOGIC, PRESENTATION) The logical separation concerns the different (logical, conceptual, architectural) aspects a distributed application has to cover. There are three aspects – data, function (or business) and presentation – found in nearly every application.

(2) UML is an adequate language to graphically describe the architecture of a software application. Modeling is necessary when the application has to be developed in a systematic way.
(Stereotype describing a component) A component in UML is modeled as a rectangle with a specific icon in the right upper corner. This icon is a graphical representation of a stereotype which is further defined in the UML meta model. An equivalent textual representation of this stereotype defining a component is <<component>>.
(Ball-Socket symbol describing an interface) The ball represents the provided part and the socket represents the required part of the component interface.

DB                  Data Base
GUI                 Graphical User Interface
UML                 Unified Modeling Language

A microservice architecture consists of three parts (backend, backend-for-frontend BFF, frontend) and two types of application programming interfaces (BFF) via which these parts are interacting (backend microservices API, BFF microservices API).

(Presentation Layer) This layer renders the UI elements in the browser. Technologies that support the implementation are Angular and Bootstrap.

(Presentation Logic Layer) The presentation includes a logic which controls the interaction with the BFF microservice API.

(Application Logic Layer) This layer realizes the orchestration of the backend microservices in order to provide the BFF microservices required by the frontend. A technology that supports the implementation of this functionality is Spring.

(Domain Logic Layer, Infrastructure Layer) These layers implement the backend services which are CRUD operations on the domain objects.

In contrast to a traditional three-layer application architecture the business logic layer in a microservice architecture is split into two layers, the domain logic layer and the application logic layer. The reason for that is to promote the reuse of business logic functionality by distinguishing between application-agnostic (= domain logic) and application-specific (= application logic) functionality.

API     Application Programming Interface  
BFF     Backend For Frontend  
CRUD    Create, Read, Update, Delete  

The software development process applied by C&M combines the concepts of Behavior-Driven Development (BDD) and Domain-Driven Design (DDD). Both concepts provide complementary contributions to the layered microservice architecture as the figure illustrates.

1. An implemented feature can be seen as a deployable increment of the software system. (Feature1, Feature 2, ...) The ordering of the features in the figure implies that the first feature should cover the core functionality of the software system.

2. The domain model makes sure that the static and dynamic domain knowledge is consistently used by each feature. This ensures that the features build a consistent whole although each feature is developed and deployed independently from other features.

BDD  Behavior-Driven Development
DDD  Domain-Driven Design
The BDD/DDD-based software development process as it is applied in the research group C&M is an agile approach in which the software system is built feature by feature. As the activity diagram shows the feature-driven cycle starts with the specification of the next feature with the highest priority and ends up with the test of the implemented feature.

(Description of the Vision, Business Goals and Capabilities) Before developing an application the vision and goals that should be achieved should be clear. Business goals describe the benefits that the business gets from of the system. The capabilities describe the functionality that is needed to achieve the business goals in a non-technical way. Features are derived from the business capabilities.

(Specification of the Main Features and their Step Definitions) The requirements elicitation is conducted by applying the BDD concept. Note: It is recommended to reflect about the domain of the software before the first feature is specified. If a domain model for this domain already exists it necessarily should be taken into account to make sure that the domain logic is consistently used by the software system to be developed.

(Design of an Initial Domain Model) (Designing an Initial Context Map Based on the Initial Domain Model) Feature specification is an analysis activity and domain model specification is a design activity. The goal of this combined approach is to make clear which domain knowledge is included in each specified feature. After the initial domain model is created, the first version of the context map is derived.

(Implementation of the Step Definitions and the Domain Model) In this step the domain logic layer is implemented according to the specification of the domain model.

(Definition of the Technology-Independent / Dependent Architecture) The technology-independent description is based on the specification of (partly web) APIs whereas in the technology-dependent description the used frontend and backend frameworks appear.

(Unit Tests, Test of the Feature) Testing both on the technical unit level and the user-oriented feature level is an important characteristic of the development process.

(Specification of the Next Feature with Highest Priority) If there are further features available, the features with the highest priority are specified and implemented.

(Adjustment of the Domain Model) A new feature may lead to new insight. This insight can lead to changes in the domain model. In some cases it is necessary to change the relation between bounded contexts. This leads to changes that are considered in the next step.

(Adjustment of the Context Map) After each cycle, the context map may need some adjustments. In early stages, there may be more adjustments than in a later stage. Depending on the strategy used, affected teams need to communicate the necessary changes. The context map shows the teams that are involved.
(1) ++Challenges in Software Development++
- Goal 1: To develop the right software (WHAT should be developed?)
- Goal 2: To develop the software right (HOW should the software be developed?)

(2) ++Types of Architectures and Relationships++
- Software architecture: logical view, component diagram
- System architecture: physical view, deployment diagram

(3) ++Traditional Three-Layer Architecture++,
++Microservice Architecture and Implementation Technologies++
- Separation of the business logic layer (middle layer) into two layers, the application layer and the domain layer
- Explicit introduction of two APIs, the Backend Microservice API and the Backend for Frontend Microservice API

(4) ++Features and the Domain Model in a Microservice Architecture++
- BDD: Each feature is a vertical cut through all layers of the microservice architecture
- DDD: The domain model is implemented in the domain logic layer and provides the semantical foundation for all features

(5) ++BDD/DDD-Based Software Development++
- BDD: Influences the analysis by providing the specification based on features. In addition, a feature defines the software increment which is implemented in an agile way.
- DDD: The domain model is the central design artifact of the development process.
(1) On the one hand, the tools support the software engineers and make software development easier and more efficient. On the other hand, the broad spectrum of different tools makes software development more complex since the correct use of the tools is a time-consuming challenge for each developer.

(1.1) The phases are: analysis, design, implementation, testing, deployment. An example of development tools which cannot be assigned to one specific phase are management tools which are used by the teams for communication purposes and for the organization of the software project.

(2) This type of development tools is indispensable for the development of software.

(2.1) Construction tools are used in the implementation phase.

(3) An example of a software requirements tool is Cucumber which supports the approach of behavior-driven development (BDD). Examples of software design tools are UML modeling tools or tools to specify the APIs (application programming interface) of a software architecture.

(4) Software testing tools include test generators, test execution frameworks, test evaluation, test management and performance analysis. Tools which support the deployment are called software configuration management tools [Ca01].

(5) The main support provided by this type of development tools is project planning and tracking of tasks.

Software engineering tools cover two major aspects which must be handled in every software project: the management aspect dealing with organizational and project-related problems and the development aspect dealing with technical and (software) system-related problems.

(Communication and Collaboration Platform) The tools for solving either organizational or technical problems of software engineering are not decoupled. If a technical problem occurs this has influence on the overall planning. In order to manage these relationships a common communication and collaboration platform for management and development is useful.

(Software Repository) A software repository enables the storage of different versions of the software in a way that each software engineer can communicate the changes he has made in a systematic way to the other team members.

(Project Management and Organization) The relevance of this group of software engineering tools rises with the size of the projects and the software engineers involved.

(Project Planning and Tracking) This type of tool is used by a project manager (in Scrum this role is called project owner) to plan the resources (esp. the developers) that are available in the project. In [Ca01] the project planning and tracking is part of the so-called software engineering management tools.

(Issue and Problem Tracking) A team of software engineers uses this type of tool to organize technical, i.e. software-related tasks.

(Software Development) In this group of tools the development process starting with the analysis and ending with the deployment of the software is supported. The list of tool types mentioned in the figure is not complete. Tool types which are missing include the software quality tools or software maintenance tools.
The main dimension according which the development tools can be ordered are the development phases (from analysis to deployment). The tools supporting the analysis and design and the implementation and testing can be grouped according to the software architecture (which in our case is a microservice architecture).

(Project Management and Version Control) These tools support the overall organization of the software project and the communication between the project members. At C&M two different tools sets are used: the Atlassian toolset extended by Microsoft tools, esp. SharePoint on which the C&M Teamserver is based.

(Analysis and Design) Analysis requirements at C&M is done by taking the approach of behavior-driven development (BDD) based on the tool Cucumber. For the design the two most relevant tools are Enterprise Architect for the domain model and Swagger for the API specification. In addition to these tools the Microsoft Office tools (Word, PowerPoint) and Atlassian Confluence (in the case of iCC/xdi projects) are applied for documentation purposes.

(Implementation and Testing) In this phase the frontend and backend of the web applications are constructed. The integrated development environment (IDE) used for frontend development is JetBrain's WebStorm and frontend frameworks are Angular and Bootstrap. As IDE for backend development Eclipse is used and Apache Spring (esp. Spring Boot for the microservice implementation) is used as backend framework.

(Build and Deployment) The build and deployment of the microservices is carried out via a build pipeline by which the concept of continuous integration and continuous deployment is provided. The result of the build pipeline is a Docker image (= application container image) since Docker is used as the container environment at C&M. Kubernetes is used to manage the Docker containers in order to reach a high scalability and robustness of the service landscape.

BDD Behavior-Driven Design
IDE Integrated Development Environment
EXERCISES: Core Tools, Main Groups, C&M Tools

(1) ++Spectrum and Types of Tools++
- Software construction tools: Program editors, compilers / interpreters, debuggers

(2) ++Classification of Software Engineering Tools++
- Project Management and Organization: The relevance of this group of software engineering tools rises with the size of the projects and the software engineers involved.
- Software Development: In this group of tools the development process starting with the analysis and ending with the deployment of the software is supported.

(3) ++Tool Environment Used by C&M++
(3.1) SharePoint, Confluence, Jira, Bitbucket, Trello
(3.2) Git, Bitbucket
(3.3) Cucumber
(3.4) Enterprise Architect
(3.5) Angular and Boostrap
(3.6) Spring
(3.7) Docker, Nexus, Portainer
In this chapter the software engineering tools are introduced that are used to manage and organize the project and the people involved in the whole software engineering process.

(1) Software systems tend to become complex which means that a team of software engineers is needed to develop the system. That is why software engineering means teamwork.
   (1.1) The project teams are built in the beginning of each semester.
   (1.2) The coach takes care that all team members contribute to the teamwork.

(2) In this context, project management comprises all tasks that should be carried out to organize the development tasks in a systematic and clear way.
   (2.1) Features specify the requirements the software system must fulfill. They are described in a language called Gherkin. Since the agile development process is organized along the features with highest priority this artifact plays an important role for the project planning.
   (2.2) The division into frontend-related and backend-related tasks make sense since they demand different competences.
   (2.3) At C&M, the Bitbucket wiki is used to store the artifact descriptions which are edited collaboratively by the members of the development team.

(3) The development of Git started in 2005. It is currently the most widely used version control system in the world today.
A version control system (VCS) is one of the most relevant software tools to develop complex software in a team [Atl-Wha].

1.1 A local VCS is a database that keeps all the changes to files under revision control. The differences from one file version to the next file version usually is stored in a certain format. An example of a local VCS is RCS (Revision Control System). The central and distributed VCS are explained in what follows [CS14].

2 A disadvantage of local VCS is that a collaboration of developers working on distributed systems is not supported which has led to the central VCS. In a central VCS, clients running on different systems check out files from the central place.

2.1 Subversion is the successor of CVS which stands for Concurrent Versions System.

3 In a central VCS, clients only check out single files and not the repository (i.e. the whole file system).

3.1 The full mirror includes the full history of all the files in the repository. This leads to a high stability.

3.2 BitKeeper today is an open-source distributed VCS which was used to develop the Linux kernel. Some conflicts caused by the fact that BitKeeper at that time was proprietary (owned by a company called BitMover) has led to the development of Git.

Mercurial is a free software which was announced and developed at the same time as Git in 2005.

(Computer A <---> Computer B) Note: There is only an "indirect" relationship between the participating computers A and B (leading via the server computer).

<table>
<thead>
<tr>
<th>VCS</th>
<th>Version Control System</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVS</td>
<td>Concurrent Versions System</td>
</tr>
<tr>
<td>RCS</td>
<td>Revision Control System</td>
</tr>
<tr>
<td>VCS</td>
<td>Version Control System</td>
</tr>
</tbody>
</table>

(1) Git was developed by Linus Torvalds, the initiator of the Linux kernel. Development started in 2005 [HP16]. The reason for the development of Git was the need of a version control system for the Linux kernel development.

(1.1) Git allows to record and manage the changes of files.

(1.2) Git is the appropriate tool both for single persons and for large projects in which hundreds of developers cooperate (such as the Linux kernel development).

(1.3) Examples of other VCS include CVS (Concurrent Versions System) or Subversion (successor of CVS) which store data as a list of file-based changes (since the changes of files are stored as deltas, they are named delta-based version control).

(2) A snapshot in Git is a picture of all the files of the project. For efficiency reasons for all files that did not change from version to the other only a link to the previous identical file is stored.

(Version 3, A1, B, C2) The snapshot Git takes for Version 3 are links for A1 and B since they did not change (illustrated by the dotted lined ellipses). Only C2 is stored since it is different from C1 (which is file C in Version 2).


This page gives an overview of the most relevant commands provided by the distributed version control system Git [Atl-Get].

1. A Git repository (also called Git repo) is a virtual storage of the versions of code from a project.
   1.1. `git init` creates a new .git subdirectory in the current working directory. It will also create a new master branch.
   `git clone <repo url>` creates a copy of the indicated remote repository. The URL format depends on the network protocol that is used.
   `git config` sets Git configuration values (e.g. email or username) on a global or local project level. It also allows to create shortcuts for frequently used Git operations in order to work with Git more efficiently.

2. In Git, each change of the local repository is done via a so-called staging area which can be seen as a buffer between the working directory and the project history.
   2.1. `git add` and `git commit` compose the fundamental Git workflow. They are the means to record versions of a project into the repository's history. The stage is used by Git to group related changes into highly focused snapshots before actually committing it to the project history (when `git commit` is executed).
   `git stash` (germ. bunkern) temporarily saves uncommitted changes and reverts them when a developer later wants to continue his work.
   `.gitignore` is a file by which files that should be ignored by Git (e.g. build artifacts or machine generated files, such as compiled code) are tracked.

3. `git status` displays the state of the working directory and the staging area.
   `git log` provides information regarding the committed project history.

4. These commands allow to explore old commits and undo the changes in different ways.

5. By using these commands the Git history can be rewritten or altered.

6. These commands are used by a developer to share a series of commits with other developers.

Bitbucket owned by Atlassian provides a web-based hosting service. It is written in Python using the Django web framework.

1. The first version of Bitbucket (developed in 2010 by an independent startup) only offered hosting support for Mercurial projects.

1.1 In contrast to other open source hosters, such as SourceForge, in GitHub not the project as a collection of source code is in the center, but the user and his repositories which are managed by Git. Operations, such as branching (e.g. creating a new repository) and merging of forked repositories are supported (https://de.wikipedia.org/wiki/GitHub).

1.2 The integration of Bitbucket with other Atlassian software, such as Confluence (communication), Jira (incident management) and Bamboo (continuous integration and continuous deployment ) leads to complete development solution.

The main features provided by Bitbucket allow a team to efficiently apply the Git-based version control concepts. The following description is based on the "Bitbucket Support" documentation provided by Atlassian [Atl-Bit].

2.1 This feature includes all operations to create and change a team and set permissions for the team members.

2.2 The set up includes the operations of creation, import, and cloning of repositories.

2.3 The work on a repository includes operations to push code from the local repository to Bitbucket or to pull code from Bitbucket to get the most up-to-date version onto the local repository.

2.4 A developer who wants to work on a separate line of code first creates a new branch or checks out an existing branch. When the code is ready to be reviewed a pull request is created. After the review the branch is merged into the main branch.

2.5 The communication takes place via issues by which tasks and bug reports can be exchanged. By the creation and editing of wiki pages everything related to the project, team, and code can be documented.


(1) C&M has got a free academic licence to use Bitbucket in its research projects. Students can get a Bitbucket account via [Atl-Cre]. It is important to use the KIT mail address. If a student already has a Bitbucket account based on his/her private mail address this account should not be used in C&M projects. The licence-related aspects are further described in [CM-C-Dur]. A confluence account offered by iCC/xdi does not include a Bitbucket account.

(2) The Bitbucket repositories are owned by teams. For each cooperation partner a team with different repositories exists. Each C&M member should be part of one or more teams (click on profile icon | View profile | Teams).

(3) The example shows the Bitbucket repositories used for the software that is being developed in cooperation with xdi360. In the Bitbucket documentation, repositories and their use are explained in detail.

The workflow conducted at C&M is inspired by the Git workflow described in [Atl-Git]. A more detailed description (in German) of the workflow outlined on this page can be found at [CM-C-Dur].

(1) In the graphical presentation of the workflow the actual development tasks are left out. Examples of such tasks are the development of the software implementing the feature or the solution of an issue (e.g. fixing of a bug).

(Check Create and Initialize Repository (BB)) The precise Bitbucket commands to create a new repository can be found at [Atl-Cre].

(2) A branch represents an independent line of development for the repository. A branch includes a new working directory, staging area, and project history. Before any new branches can be created, the main branch called master is automatically created.

(2.1) (origin/stable) This branch is optional and contains the last stable version which was deployed.

(2.2) In the feature branch new functionality is developed whereas in the bug branch faults in the software are solved. There is not difference how to work with these branches by using Bitbucket and Git operations.

(3) The (feature-related or bug-related) changes on the software are carried out by the following three steps: (i) locally create and work on a new branch, (ii) merge the changed branch into the local master branch and (iii) push the change to Bitbucket. Theses steps are described in the Bitbucket documentation [Atl-Use].

(3.1) The terminal window should be opened and the current folder should be the top level of the repository: cd ~/repos/<repo-name>

A local branch can be created by using the following git command: git branch <feature/featurename>

To begin working on the new branch, it must be checked out: git checkout <feature/featurename>

Stage the changed file: git add <filename>

Commit the changed file: git commit <filename> -m <comment explaining the change that is committed>

(3.2) Again, the terminal window should be opened and the current folder should be the top level of the repository: cd ~/repos/<repo-name>

It should be made sure that the feature branch to be merged is checked out and all changes have been committed. This information is provided by: git status

The master branch is to be checked out: git checkout master

Now, the changed feature branch can be merged to the master branch: git merge <feature/featurename>

(3.3) In order to make the changes available to the other developers of the team the state of the local repository (Local/Master) must be pushed to the central Bitbucket repository (Origin/Master): git push origin master

BB Bitbucket


The name Jira stems from the Japanese name for Godzilla (the name of a monster) which itself is a reference to Jira's main competitor, Bugzilla (an open source software released by Netscape Communications). It is pronounced "Jee-rah". The software is developed by Atlassian since 2002 and it is written in Java [Wik-Jir].

(1) Jira is used for Project Management, Bug Tracking, Issue Tracking and Workflow [tut-Jir].
(1.1) This functionality is needed in help desk, support and customer services.
(1.2) This includes task tracking, requirement management and process management.
(1.3) Such tools include source control programs such as Git or Version Control Systems (CVS).

(2) Issues are the central concept since a project is a collection of issues and a workflow tracks the lifecycle of an issue.
(2.1) Examples for projects are: software development project, marketing project, employee performance system.
(2.2) Examples of issues are: task or sub-task of a story, bug or defect, helpdesk ticket can be logged as an issue.
(2.3) Examples of statuses an issue are: open, in progress, resolved, reopened, close

(3) The Jira dashboard is the first page that shows up [tut-Jir].
(3.1) These links are projects, issues, boards and create.
(3.2) In this section the issues list assigned to users is displayed. Another example of a main section is "Activity Stream" which display activities done by the user.

Trello is a lightweight alternative for professional project management tools like Jira.

(1) Originally, Trello was developed by Fog Creek Software in 2011 and sold by Atlassian in January 2017.

(1.1) There is extensive information available ([Atl-Ein], in German) for what purposes and in which way Trello can be used.

(2) The functionality of Trello is used by C&M to establish a Scrum board which provides a structured way to organize the development tasks of all members of the project team.

(2.1) It is recommended to use the KIT mail address when signing up the Trello account.

(2.2) The guidelines include a common list structure of each Trello board. A detailed description is part of the C&M teamwork [CM-CMT].


EXERCISES: Project Management, Version Control Systems, Git

(1) ++Overview++
- Chronological planning
- Distribution of tasks
- Documentation

(2) ++Version Control System++
- Local: VCS Client and VCS server run on one system. Disadvantage: collaboration of developers working on distributed systems is not supported.
- Central: Clients running on different systems check out files from the central place.
- Distributed: Client fully mirrors the repository. Each system holds a full backup of all data.

(3) ++Git++
- Git stores snapshots in each client. A snapshot is a picture of all the files of the project. For efficiency reasons for all files that did not change from version to the other only a link to the previous identical file is stored.
- Subversion (successor of the Concurrent Versions System CVS) stores data centrally on the server as a list of file-based changes (since the changes of files are stored as deltas, they are named delta-based version control).

(4) ++Bitbucket / Git Workflow @ C&M++
- Branch: represents an independent line of development for the repository. A branch includes a new working directory, staging area, and project history.
  - Types
    -- Master (origin/master): Before any new branches can be created, the main branch called master is automatically created.
    -- Stable (origin/stable): This branch is optional and contains the last stable version which was deployed.
    -- Feature (feature/featurename): In this branch new functionality is developed
    -- Bug (bug/bugname) In this branch faults in the software are solved. There is not difference how to work with these branches by using Bitbucket and Git operations.

(5) ++Jira++ ++Trello++
- Jira: Professional incident management tool which allows to track incidents based on flexible workflows
- Trello: A simple project management tool which provides boards on which a team can organize its tasks
This chapter deals with the first two phases of the development process and describes the way how the analysis and design of a web application are carried out based on the BDD/DDD approach and the systematic derivation of web APIs from the domain model.

(1) The one activity is concerned with two artifacts, the features and the domain model, whereas the second activity concentrates on the APIs of the microservice architecture.

(1.1) Feature specification is a pure analysis activity and domain model specification is to the better part a design activity. The goal of this combined and strongly coupled specification of features and the domain model is to work out the (domain knowledge) core of a feature. Since each feature and the domain knowledge which is used in the feature must be related to the one and only domain model a consistency of all features is guaranteed.

(1.2) This activity leads to the technology-independent, microservices-based architecture of the software systems to be developed.

(2) This is an early implementation activity which can be carried out before any architectural design is specified. From an early implementation of the domain logic a deeper understanding of the domain knowledge, especially its dynamic aspects will be gained.

API Application Programming Interface
BDD Behavior-Driven Development
DDD Domain-Driven Design
Analysis and design comprises the development work that is needed to implement the right software in the right way.

(1) Analysis is concerned with the question which software the user requires [CM-W-BEH].

(1.1) Gherkin is a language which introduces a minimal structure into the largely informal description of the requirement description.

(1.2) Cucumber allows to test an implemented software based on the specified features. The test are called user acceptance tests.

(2) Design is concerned with the static and dynamic structures of the software system that is to be implemented [CM-W-DOM].

(2.1) The concept of domain-driven design (DDD, [Ev04]) does not include a way how to specify the parts of a domain model. The Unified Modeling Language (UML) is a widely accepted modeling language which provides the flexibility to express the needed DDD modeling elements. Enterprise Architect is one of the leading professional UML tools to implement the DDD-to-UML approach.

(2.2) Based on a formalized UML-based domain model, the web application programming interfaces (API) of the microservice architecture can be systematically derived. The specification of the APIs is based on the OpenAPI defacto standard. Swagger is a well-known tool to specify and test the web APIs.

[CM-W-BEH] Cooperation & Management: BEHAVIOR-DRIVEN DEVELOPMENT, WASA Course Unit. https://team.kit.edu/sites/cm-tm/Mitglieder/2-1.WASA

[CM-W-DOM] Cooperation & Management: DOMAIN MODELING, WASA Course Unit. https://team.kit.edu/sites/cm-tm/Mitglieder/2-1.WASA

Developers use lots of different words to describe what they want to build. The following terminology taken from [Sm15:88] is used to provide a clear understanding in this confusing part of software development.

1. The project vision provides a high-level guiding direction for the project. The problem must be clarified to understand which software system is needed by which users to solve the problem [:61].

2. The software system must have a measurable, positive impact on the business of the customer the software is built for.

3. Capabilities give users or stakeholders the ability to realize some business goal or perform some useful task. A capability represents the ability to do something; it does not depend on a particular implementation. For example, "the ability to book a flight" is a capability [:89].

4. A feature is a piece of functionality that are delivered to the end users or to other stakeholders to support a capability that they need in order to achieve their business goals [:98].
A feature is something that users can test and use in isolation. A feature can deliver business value in itself; once a feature is completed, it could theoretically be deploy into production immediately, without having to wait for any other features to be finished first [:93]. The description of the features is generally more effective than describing what the application does.

5. The typical format of a story is "As a – I want – So that". In opposite to the format of a scenario it is not prescribed by Gherkin.

6. Scenarios written in the format "Given – When – Then" make up the core of an executable specification [:115].

The terms introduced on the page ++Used Terms Related to the Requirements Analysis++ are explained with the example of the SmartCampus service NavigationServiceGroup (NavSG) which provides a smart IoT-based (Internet of Things) service to get campus information and navigate on the campus and in certain buildings of the campus.

1. A specific aspect of NavSG is to regard the accessibility of the navigation functionality provided to its users in order to support especially handicapped persons.

2. Another business goal is to accelerate relevant processes in which it is important to efficiently find certain locations on the campus or inside a building.

3. Besides the location further information related to points of the campus might be of interest. This information is stored in so-called PointsOfInterest (POI).

4. The features are the actual outcome of the requirements analysis of the software system since these are the artifacts which describe what the software should provide to its users

   4.1 The feature to request the location is the NavSG core functionality.

   4.1.1 The scenario "Invalid beacon IDs" specifies an error case of the feature.

   4.2 (4.3) These features concern the information delivery aspect of the NavSG. One of the features is especially intended for handicapped persons.

---

**Example: SmartCampus NavigationServiceGroup (NavSG)**

1. **Vision:** Provide campus information and indoor campus navigation which can also used by handicapped persons

2. **Business Goal:** Make the campus more attractive and accessible for its students, employees and guests (exemplary)

3. **Capability:** Determine the location on the campus inside a building or a room (exemplary)

4. **Features and corresponding scenarios**
   - Determine location based on indoor beacons
     - Beacons available, Note enough beacons available, Only invalid beacons available
   - Get information about a point of interest
     - Inside building, Outside building
   - Get information about the accessibility of a point of interest
     - Show available information, Filter entries by accessibility types

---

IoT Internet of Things
NavSG NavigationServiceGroup
POI PointOfInterest
The NavigationServiceGroup was developed by C&M [CM-E-NAVSG-PSE] according to the concepts of BDD (Behavior-Driven Development). The feature "Determine location based on indoor beacons" describes the core functionality that the navigation service must provide.

(3. – 5.) These lines provide the story of the feature. It specifies the role which uses this feature (student) and what are the benefits of the feature for the role (good orientation on the campus).

The functionality of this feature is specified by three scenarios which describe the behavior of the NavSG software by a scheme

Given (context, precondition) – When (event, action) – Then (testable outcome)

as it is prescribed by the language Gherkin. Each scenario serves as a user acceptance criterion which can be automatically tested by a BDD supporting tool environment such as Cucumber.

(7. – 12.) In the first scenario "Beacons available" the situation is specified which allows to conduct the triangulation in order to determine the location.

(14. – 22.) The second and third scenario describe two situations in which the location cannot be determined because not enough or invalid beacons are available.

BDD Behavior-Driven Development


The artifact which describes the Gherkin features according to the BDD is the main outcome of the requirements analysis. Since all further development steps, esp. the modeling of the domain, depend on this artifact it should be documented in an adequate and defined way.

(1) The Gherkin features provide a requirements specification which is an integral artifact of the software development process. Therefore, developers must have access to the features in their IDE (Integrated Development Environment) since they run through the features in order to carry out user acceptance tests of their software implementation. Nevertheless, features are not only relevant for the developers, but also for the stakeholders for which the software is developed. Therefore, an adequate way of documentation of the feature collection must be made available not only for the developers, but also for the stakeholders.

(1.1) The use of Microsoft Word is one possible way to document the features for the stakeholders. It should be ensured that the document provides a good overview of the features and scenarios. (Word Navigation Area) In Word the navigation area can be used for this purpose as it is shown with the example of the NavSG feature document [CM-A-NAVSG-PSE]. An alternative is to use Confluence which is on a wiki concept and which also provides features to organize the features and scenarios in a clear way.

(2) The specification of features is a creative process which makes it impossible to state complete rules how to go through this process. Nevertheless, best practices can be formulated which give some valuable hints for the process.

(2.1) The first (one to three) features should specify the functionality of a so-called Minimum Viable Product (MVP).

(2.2) A business value will usually not be provided by one single functionality but by a set of coherent functions by which a user can fulfill a specific task.

(2.3) The terms of the ubiquitous language should be used. These terms should be understood by the software developer and the business owner/analyst.

A fourth best practice which is formulated in [CM-W-BEH] in more detail is to use a declarative and not an imperative way to express the scenario steps:

MVP Minimum Viable Product

[CM-W-BEH] Cooperation & Management: BEHAVIOR-DRIVEN DEVELOPMENT, WASA Course Unit. https://team.kit.edu/sites/cm-tm/Mitglieder/2-I.WASA

(1) **Overview**

The analysis phase (which is the first phase and which is followed by the design, implementation & test, deployment, operation & maintenance phases)

(2) **Tools Used in the Analysis and Design Phases**

- Gherkin features: specified and tested by cucumber
- Further tools: Enterprise Architect (domain model), Swagger (web APIs)

(3) **Used Terms Related to the Requirements Analysis**

- The starting point are the vision which states what the project wants to achieve.
- From the vision the business goals (what will the business will get out of the project) and the capabilities (what should users and stakeholders be able to do to deliver these goals) are derived which lead to the features.

(4) **Example: Main NavSG Gherkin Feature**

- The structure of a scenario is: Given (precondition, context) – When (action) – Then (testable outcome)

A scenario in general expresses a concrete example in a defined format by which a feature is illustrated. It serves a user acceptance test of the implemented software.

- In the example of the NavSG feature "Determine location based on indoor beacons" the first scenario "Beacons available" describes the case that the location can be determined (When). This the case when three valid beacons are available (Given). The outcome (Then) is the location which is shown to the user.

(5) **Documentation of and Best Practices for Gherkin Features**

- The developer and the stakeholder.

This leads to the requirement that features should not only be documented in the IDE of the developer but also in external format (e.g. Word or Confluence pages).
According to the overall BDD/DDD-based development process in the subsequent step the domain logic contained in features should be added to the domain model.

Note: In what follows the term feature is synonymous to a Gherkin feature.

(1) A feature specifies a specific part of the business logic from the viewpoint of a user who applies the functionality in a certain role ("As a ..." in the story) and in certain situations ("Given ..." in the scenario).

(1.1) The other part is the application logic. It is important to notice that in the DDD approach the business logic is divided into the domain logic and the application logic. The domain logic is application agnostic which means that this logic is relevant for other applications of this domain. Therefore, the domain logic is the re-usable part of the business logic appearing in a feature.

(2) Like the process of feature specification the derivation of domain knowledge from the features is a highly creative process. Nevertheless, some hints given in [So17] are helpful.

(2.1) By answering this question those terms of the feature should be identified that are relevant for the understanding of the user requirement.

(2.2) Only those aspects identified by Question 1 should be added to the domain model which are not specific to the regarded application.

(2.3) Three types of relationships can be distinguished: general associations, containment (composition, aggregation), inheritance.
The derivation of domain knowledge is shown with a part of the NavSG feature that was introduced on a previous page.

(1) The approach is to go through the lines of the feature and identify those terms which are presumably relevant for the understanding of the domain and the functionality related to the domain, i.e. the domain logic.

Note: The candidates are written in UpperCamelCase which is the format used in the domain model.

(1.1) The list should be seen as a candidate of terms that might become part of the ubiquitous language of the domain. For those terms which certainly will be part of the language, a definition should be formulated. An example of such a term is "Beacon".

Beacon: A device which sends in regular time intervals a data packet containing a unique identifier. The data can be received by devices which support Bluetooth 4.0 within a range of 20 to 30 meters.

(1.2) As a convention, activities should be described by a verb and a noun which are written in UpperCamelCase.

(2) The relation view is specified as a UML class diagram as illustrated on a following page.
The domain model emerges from two types of modeling, the tactical modeling and the strategic modeling.

(1) The tactical modeling corresponds to the well-known proceeding of modeling knowledge as conceptual class diagrams added by dynamic diagrams to model certain time-related aspects. The result from the tactical modeling can be seen as the core content of the domain model.

(1.1) Four types of views are distinguished [CM-W-DOM]. Only the process view allows to model dynamic aspects, whereas the relation view, domain object view, and interaction view focus on the static aspects.

(1.2) The relation view corresponds to conceptual class diagrams as will be shown with an example.

(2) The strategic modeling is the real new part of domain modeling and the result, a so-called context map, provides the link to the microservice architecture.

(2.1) The structuring of the domain knowledge (as it is described by tactical modeling based on the four view types) is carried out in two steps: The whole domain is structured into subdomains and the subdomains are structured into so-called bounded contexts. To find an adequate structure coupling that domain knowledge which is coherently connected "from its nature" is one of the main challenges in domain modeling. The term "bounded context" is derived from the approach of a Domain-Driven Design (DDD, [Ev04]). A bounded context form the architectural perspective can be seen as a microservice candidate. This is the reason why a domain model serves as a blueprint for the microservice architecture which implements the model in the domain logic layer of the layered architecture. Domain objects can be shared by bounded contexts. These objects are called shared entities.

(2.2) DDD does not prescribe a concrete representation of the model. At C&M, the modeling elements from the Unified Modeling Language (UML, CM-W-UNI) are used to model the results from the strategic (and also from the tactical) modeling.

DDD  Domain-Driven Design
UML  Unified Modeling Language

[CM-W-DOM] Cooperation & Management: DOMAIN MODELING, WASA Course Unit. https://team.kit.edu/sites/cm-tm/Mitglieder/2-1.WASA
A central design artifact in the C&M development process is the domain model for which the Enterprise Architect is used as modeling tool.

(1) Enterprise Architect is one of the most popular UML (Unified Modeling Language) modeling tools with more than 200,000 licensed users in over 100 countries.

(2.1) Always the latest UML specifications (www.omg.org) are supported by EA.
(2.2) A wide range of software systems can be designed and constructed. For business purposes the most popular notation for business process modeling, the Business Process Model and Notation (BPMN), is supported by EA.
(2.3) EA provides model repositories that can be used by large teams to collaboratively work on models that are built in the design phase of software development.
(2.4) This property of EA is used to adapt the tool to the domain modeling concepts followed by C&M.
Note: The term domain has different semantics in the context of domain-specific modeling (or domain-specific language DSL) and domain modeling according to the approach described by Evans [Ev04].


BPMN Business Process Model and Notation
DSL Domain-Specific Language
EA Enterprise Architect
UML Unified Modeling Language

This page gives an overview of the steps that must be carried out to get started with EA at C&M. Further information on the EA and its use at C&M can be found in [CM-L-ENT].

1. The EA software must be installed locally on each user's computer. The download link of the EA software provided by the software distribution server is given in the reference [CM-SWD-EA]. The link of the installer script "1.enterprise_architect_v12.0_installer.msi" should be clicked to start the installation.

2. EA stores the model content in a remote MySQL repository for easy collaboration between team members. In order to get access to the repository, a MySQL ODBC driver has to be installed and configured system wide.
   2.1 A current installer of the MySQL ODBC connector can be found in [MyS-Dow]. The 32-bit version should be selected regardless of the own system architecture, as this is a requirement of EA. After clicking on "Download", the final download link can be accessed by clicking on "No thanks, just start my download." below the login and sign up buttons. If the installation fails, most likely the x86 Visual C++ Redistributable 2013 (choose "vcredist_x86.exe") needs to be installed which can be found in [MS-Vis].
   2.2 The configuration of a MySQL data source is described in [Spa-Set]. In step 3 the Unicode version should be chosen. In the C&M environment, the configuration parameters should be set as follows: Data Source Name: CM-EA-Repository, Description: empty, TCP/IP Server: mysql2g.scc.kit.edu, Port: 3306, User: tm-cm-ea-0001, Password: ChiMaira$EA, Database: tm-cm-ea-0001_ea

3. To be able to use EA, a license key has to be added on first launch.
   3.1 To add a license key to a local EA installation, the steps under "Add a Shared Key" in [Spa-Add] can be followed. The "..." in the first shown menu must be clicked to open the appropriate dialog in which the "Sparx Key Server" must be selected. When prompted for a keystore, the following parameter values should be entered:
   - Server Address: license.cm.tm.kit.edu
   - Password: ChiMaira$EA!

4. Finally, a connection to the already configured MySQL ODBC driver has to be established from within EA. For this purpose, the steps 1 to 10 (excluding 6 and 7) in [Spa-Con] can be followed. Additionally, "Lazy Load" can be disabled to allow for a faster workflow, once the entire model has been loaded. Since the repository is located in the KIT Intranet a VPN connection is needed if it is accessed from outside.

**EA @ C&M**

1. Installation of the EA software available on C&M's software distribution server
2. Setup of a MySQL ODBC Driver to be able to connect to C&M's model repository
   1. Installation of a 32-bit ODBC driver
   2. Configuration of the TCP/IP server, port, user, password and database
3. Adding a license key from C&M's keystore on first launch
   1. Configuration of the server address and password
4. Connecting to C&M's shared MySQL model repository

---

**EA** Enterprise Architect  
**ODBC** Open Database Connectivity  
**SWD** Software Distribution Server  
**SQL** Structured Query Language

---

(1) ++From Features to the Domain Model++
- Features contain domain knowledge which should be formalized in a domain model according to DDD
- Features specify that part of the business logic of the software system that should be implemented in the domain logic layer and which should be separated from the application logic layer.

(2) ++Example: Domain Knowledge in the Exemplary NavSG Feature++
- The approach is to go through the lines of the feature and identify those terms which are presumably relevant for the understanding of the domain and the functionality provided by the domain, i.e. the domain logic.
- In the NavSG examples these are
  -- Objects: Location, Beacon, BeaconID, BeaconSignal, Student, Building, Room
  -- Activities: DetermineLocation, RegisterBeacon, TriangulateLocation

(3) ++Stepwise Construction of the Domain Model++
- Strategic modeling provides the overall structure of the domain model by defining subdomains and bounded contexts.
- Bounded contexts are the candidates for microservices.

(4) ++EA @ C&M++
- Language: UML
- Tool: Enterprise Architect
- Model repository: MySQL database accessed via an ODBC driver (Open Database Connectivity)
Method to Systematically Derive Resource-Oriented Web APIs from the Domain Model

(1) Identification of relevant domain objects that should be exposed
   (1) User requirements provide corresponding hints
   (2) Resulting set represents the resource candidates

(2) Determination of resource types based on the resulting resource candidates

(3) Transformation of domain objects into resources
   (1) Heuristics and transformation rules for easier transformation into a so-called resource model

(1.1) For instance, an explorative prototype, based on the user requirements, shows which interactions are needed.

(1.2) The output of this step is called "candidates". For each candidate, the software architect has to choose if the domain object is exposed or not. They are transformed into "real" resources by a transformation described in the following steps.

(2) Based on the resource candidates the resource types are determined.

(2.1) The semantics of each resource type are described in more detail in [Gi18]. The resource types used in this approach are based on the current research results in this field:
- **PrimaryResource**: A resource which represents a domain object and which is directly addressable. A primary resource can be derived from an object of the domain model that has no inbound associations.
- **SubResource**: A resource which is a logical part of another resource and which is directly addressable. These are resources which have one or more inbound associations.
- **ListResource**: A resource which contains more resources as list elements.
- **Domain object with multiple instances or which is in a 1..n relationship and which represents the parent.**
- **InformationResource**: A resource that can not dereference itself but is associated with an identity. A domain object which is represented as a value object and which has an inbound association.
- **ActivityResource**: A resource that represents a business process. Domain object which is modeled as a domain service.

(3) The semantics of each resource type as well as the domain objects and their associations allow to derive heuristics and transformation rules.

(3.1) The heuristics identify a resource type based on the domain model. But only the resource candidates have to be considered for identification. Each heuristic is linked with a transformation rule that transforms a resource candidate into a "real" resource by assigning a stereotype from a dedicated UML profile called resource orientation and placing it into a so-called resource model. For example, the heuristic for a primary resource is the following: A domain object is an entity and has no outgoing associations. The transformation rule for this heuristic consists of these steps: The first step is to copy the entity into the resource model and remove all existing stereotypes, except the identifier. The next step is adding the stereotype "PrimaryResource" to the copied entity. The resource model applies the previously mentioned UML profile and extends the UML with concepts of REpresentational State Transfer (REST). It is important to note that also behavioral aspects can be modeled within the resource model. To keep the model lightweight, CRUD operations were not modeled and instead considered as implicitly given.

**API** Application Programming Interface
**REST** REpresentational State Transfer

Refinement of the Initial Resource Model

(1) Design process for refining the initial resource model (RM1)
   (1) Assigning addressing scheme (RM2)
   (2) Adding interactions by using a pattern language (RM3)
   (3) Creating representation scheme for resources
   (4) Adding version information for subsequent changes of the resource model (RM3)

(2) The whole design process is based on best practices and patterns
   (1) Improving usability, power and discoverability for an enhanced reusability of the resulting microservice

(1) The initial resource model (called RM1 in the refinement process) is created based on the underlying domain model, as showed on ++Example: NavSG Resource Model++. In addition to the structural aspects, the behavior-specific aspects of the domain are also transferred in a suitable form so that the behavior is also reflected in the resource model. For a complete and detailed description of these steps see [Gi18].

(1.1) Each resource is supplemented by a Uniform Resource Locator (URL) component based on its assigned resource type. By following links between resources, the complete URL can be derived. For the creation of corresponding components, linguistic patterns and best practices are used to increase the usability and ultimately the reusability. An example is showed on slide ++Example: NavSG Resource Model++.

(1.2) After adding addressing information to RM2, the interactions with a resource from the client is considered. The offered behavior by the domain has to be mapped onto a uniform interface which is a constraint of the architectural style REST. To simplify the mapping, existing patterns were identified and assigned to the individual resource types. Regarding the resulting influence, each pattern was extended by their influences on quality characteristics. Since the web API is the link between a client and a resulting microservice, the patterns also show their effect on different interaction participants. After this steps, RM3 has been created.

(1.3) The creation of representation schemas is the next logical step. The schemas define the structure of the request/response body. In addition to listing the required and optional attributes, they can also be used to define valid attribute values or to show sample values of some attributes. Based on these schemas, mock values can be generated that may be suitable for web API testing.

(1.4) The last step represents the versioning of the resource model and finally the web API. Version information using Semantic Versioning (SemVer) is added to RM3. Since this is only a small addition from the technical perspective it is still an RM3. The representation of this version information in the web API is treated by patterns. Besides, a process is introduced on how to change the version information. According to the versioning process more than one resource model may exist at the same time. RM3 is the final result and an example is showed on ++Example: NavSG Web APIs (Excerpt++)

(2) Each refinement step can be found in many of today’s web APIs. The refinement process combines the expert knowledge and forms a systematical and traceable process to build well-designed web APIs for the modeled domain. Web APIs are an important asset of digital transformation and have to be designed with care.

API Application Programming Interface
RM Resource Model
REST REpresentational State Transfer
URL Uniform Resource Locator
The domain objects which are structured in this relation view of NavSG are derived from the NavSG features introduced in
++Example: SmartCampus NavigationServiceGroup (NavSG)++.

The three primary domain objects are: Beacon, PointOfInterest, and AccessibilityInformation.

(Beacon) A sensor which periodically sends signals containing a unique identifier which is received by client applications in
order to determine the position based on a triangulation.

(Position) A value object a beacon contains (modeled as an aggregation). The location describes the position of the beacon using
the three attributes longitude, latitude, and altitude.

(PointOfInterest) A PointOfInterest can be represented by a building, a floor or a room. In general it is a public space.

(AccessibilityInformation) This entity contains information related to a PointOfInterest which is relevant for disabled persons.

(Elevator) (Floor) Examples for parts of a PointOfInterest which can help disabled persons.

(TriangulationService) The TriangulationService is a domain service which uses the users current position to find the closest
PointOfInterests which can be navigated to. Therefore the user must be in the range of at least three beacons.

(Region) A region is a defined space which can be covered by beacons. The beacons inside of an region can be used together for
doing triangulations as an example.

(Building) (Floor) (Room) Building is an entity of a physical building which contains at least one floor containing at least one
room.

(Portal) This entity offers an the entrance to a building.

(DomainService) Domain services containing operations which can not be allocated to a specific domain object. Usually it uses
several domain objects. In addition the operations must be stateless.

(Entity) An entity is a reference to an object within the domain. It has its own identity and thus its own identifier. Changing an
attribute of the entity does only affect the attribute while the entity itself does not change.

(Shared) A shared domain object means that it is not only used in this bounded context.

(ValueObject) A value object is a reference to an object within the domain. In comparison to an entity, a value object does not
has an own identifier and it is immutable. For that reason each time an attribute is changed, the value object is replaced by a new
version.
The resource entities which are structured in this resource model of NavSG are derived from the bounded context Orientation with beacons of the NavSG domain model. To identify the type(s) of a resource, the heuristics and transition rules are used. In this example, the domain object "PointOfInterest" has no incoming associations. Therefore, the first heuristic is met and the resulting transition rules are applied. In the first step, the entity is copied to the resource model and all stereotypes except the identifier are removed. The removed stereotypes are "Entity" and "Shared" but not "identifier". Afterwards, the stereotype "PrimaryResource" is added to the copied element. In addition, another heuristic is met. The domain object has a one-to-many (1..n) relation and represents the parent, meaning another transition rule is applied. This transformation rule consists of two steps. The first step is the identification of the corresponding resource. In this case, the identified resource is called "PointOfInterest". The second step is adding the stereotype "ListResource" to the modelled resource. The inheritance of the domain object "PointOfInterest" is resolved with the approach single resource inheritance with unique type (SRIT).

Assigning the addressing schema for the URI hierarchy works as the following:
Using the heuristics and transformation rules, the derivation starts with the primary resource, called "PointOfInterest". "PointOfInterest" has the stereotype "ListResource" in addition to "PrimaryResource", meaning the plural form is used for the path. As a result, the path "pointofinterests/{uuid}" is derived. The directed associations between the different resources are used to derive the complete hierarchy. The starting points are the primary resources. A software architect may add additional URI components. In this example, it leads to the following URI hierarchy:
point-of-interests/{uuid}/regions/{uuid}/beacons/{uuid}
point-of-interests/{uuid}/accessibility-information/{uuid}
point-of-interests/{uuid}/position
beacons/{uuid}
nearbypois

URI Unified Resource Identifier
SRIT Single Resource Inheritance with unique Type
Derivation of Web APIs from the Resource Model

1. Resource model provides technology-agnostic view of the web API

2. Linking the resource model with application level protocols based on their characteristics
   - Safe and idempotent methods
   - Examples: HTTP, COAP, etc.

3. Transformation into a well-known specification format for web APIs
   - Description of meta-information
   - Description of reusable components
   - Description of interactions

4. Optional: Review the resulting web API specification by experts to ensure desired quality characteristics

(1) The resource model itself is the starting point for deriving a web API. The resource model does not contain any design decisions regarding the technology or platform. Therefore the developer can choose any technology during the further development. The technology decisions can be made independently by the respective development teams.

(2) The choice of the appropriate protocol is based on their characteristics which have to comply with the needs of the resource model. In doing so the application level protocol should fit to the service consumers and if necessary their non functional requirements. Some issues for this choice are the connection type (synchronous or asynchronous), the required exchange format and required methods.

   (2.1) Safe methods are methods that do not modify resources. Idempotent methods are methods which outcome won't change no matter how often they are called. HTTP safe methods are GET, OPTIONS and HEAD, while idempotent methods are GET, PUT, DELETE, OPTIONS and HEAD.

   (2.2) Examples of protocols for accessing resources are COAP and HTTP. While COAP is mainly used for machine to machine communication in unreliable environment, HTTP is the most used application layer protocol.

(3) To make further use of the web API it can be transferred into a machine interpretable specification format. By this independent testing of the backend and frontend, automated building of the API and an automated code template creation can be enabled. Examples for machine interpretable API specifications are Open API (formerly Swagger) or RAML. For practical use at least the following has to be done.

   (3.1) First of all there has to be a description of the API's meta-information. This includes its version, general information (title and description) as well as contact information for support queries.

   (3.2) Furthermore reusable components have to be described. This components could be schemas, responses and request parameter. For instance, a query parameter description consists of a name, a short explanation, and a data type.

   (3.3) Using the components in (3.2) the interactions can be formalized, since an interaction can be composed of the previously mentioned reusable components.

(4) If the API is used in a large environment, it is very important that it is as stable as possible and meets a certain quality standard. Otherwise the API becomes less useful and generates more costs as actual benefit. Therefore an review can be performed to check its quality criteria. Possible reviewers are the API consumers as well as dedicated experts in the API design field. If there are required actions, it is necessary to populate it in the resource model and propagate it through the entire process again.

COAP Constrained Application Protocol
HTTP Hypertext Transfer Protocol
RAML RESTful API Modeling Language
In the final step the API specification can be derived from the resource model. In this example an OpenAPI specification is shown.

The specification follows the schema provided by the [OAI-Ope]. The specification is split into meta information, reusable components like schemas, responses and request parameters and interactions like the URI path, the corresponding HTTP method, the transfer of request parameters and the transfer of response parameters.

The meta information (not part of the excerpt) describes the provided web API and the corresponding bounded context the specification is derived from. The basic meta information which should be provided are a description of the domain, the version of the specification and contact information.

(1. paths …) The “paths” object holds the relative paths to the individual endpoints and their operations.
(2. get: ...) A definition of a GET operation on this path.
(3. parameters:) A list of parameters that are applicable for this operation.
(4. - $ref ...) The keyword $ref introduces reusable components which can be used in the specification.
(6. responses) The list of possible responses as they are returned from executing this operation. The responses contain the different status codes and the corresponding description and content.

URI Universal Resource Identification
HTTP Hyper Text Transfer Protocol

EXERCISES: Derivation of the Resource Model, Derivation of Web APIs

(1) +Method to Systematically Derive Resource-oriented Web APIs from the Domain Model++
- The domain model forms the core of a microservice when following a domain-driven design approach. For this reason, it is important that this is adequately represented by the web API with the correct terms as well as its logic coherence. This coherence, in turn, favors the discoverability by using the ubiquitous language on the level of the web API and ensures the reusability by focusing on the domain knowledge.

(2) ++Refinement of the Initial Resource Model++
- Best practices and patterns support the deriving process by showing their advantages and their positive influence on the resulting web API. A well-designed web API can reduce the integration effort for developers as well as enhance the discoverability and reusability.

(3) ++Refinement of the Initial Resource Model++
- Assigning addressing schema heuristics and transformation rules for addressing
- Adding interactions by using a pattern language
- Creating representation schemas for resources
- Adding version information for subsequent changes

(4) ++Derivation of Web APIs from the Resource Model++
- Improve the usability (also known as Developer Experience) and power
- Enhance the discoverability by using the ubiquitous language
- Ensure the interoperability by considering subsequent changes
- Each of these criteria has a positive influence on the reusability of the microservice
In the subsequent implementation phase the functionality as is was analyzed and designed is coded based on a microservice architecture and frameworks supporting the frontend and backend implementation. All implementation work is carried out in a test-driven manner based on unit tests and user acceptance tests which are defined by the scenarios being part of the Gherkin features. In the deployment phase the tested application is deployed in a Docker-based execution environment. As the microservice approach is strictly followed the concept of Continuous Integration/Deployment/Delivery can be supported.

1. The BDD-DDD-based development process is an agile approach which means that implementation of the software should start as soon as possible.

1.1 A (Gherkin) feature specified according to the BDD approach usually is the part of the software which is incrementally designed, implemented and tested.

2. This part of the implementation concentrates on the domain logic being part of the domain logic layer of a microservice architecture. It can be carried before the complex work on the APIs of the architecture is fulfilled.

3. This is the main part of the implementation work. A prerequisite for the start of this implementation is the existence of the API specifications of the microservice architecture.
The implementation of a web application can be separated into two parts, the frontend implementation (client side) and the backend implementation (server side).

1. Since the tasks that have to be carried out during frontend and backend application require different competences and the use of different tools it makes sense to build two separate teams.

2. The backend for frontend (BFF) service bridges the gap between the requirements of the frontend to the backend and the available backend services.

3. The domain model and the ubiquitous language it is abed on defines the common language of both teams. If changes of the domain model become necessary during the development a clear agreement between the two teams is necessary.
The microservice-based implementation architecture consists of a frontend, backend-for-frontend (BFF) and a backend. Angular is used as implementation technology for the frontend and Spring as implementation technology for the BFF and the backend microservices.

(1) A microservice architecture divides the software system into three the distributed subsystem (or system parts) frontend, BFF, and backend. These systems communicate via resource-oriented web APIs.

(Diagram) The Unified Modeling Language (UML) provides the deployment diagram to model this physical view on the system as shown on the right hand side of the slide. The three-dimensional boxes represent the UML modeling element of a so-called node (which describes a computer system). A node can contain so-called artifacts which describe software programs running on the node.

(2) Angular is a popular open-source framework to implement the frontend based on HTML (HyperText Markup Language) and Typescript which is an extension of JavaScript.

(.angular-cli.json) A JSON file (JavaScript Object Notation) which contains configuration information of the whole project setup.

(app.module.ts) A Transcript file which defines the modules that are bootstrapped by the framework at its start.

(3) Spring is one of the most popular open-source development framework for Java-based application servers. Many available open JavaEE technologies, such as Java Persistence API (JPA) are used and extended by Spring.

(4) The three system parts of the microservice architecture (usually) run on distributed client and server systems which communicate via the Web. The way how specification of the web APIs are specified is described in the previous Chapter ANALYSIS AND DESIGN.

(5) Since behavior-driven development (BDD) is used the whole development process is test-driven. The BDD tool provides an automatic execution of user acceptance tests which includes JUnit testing.

BDD Behavior-Driven Design
HTML HyperText Markup Language
JPA Java Persistence API
JSON JavaScript Object Notation
UML Unified Modeling Language
The figure illustrates the overall implementation architecture which is based on the MVC pattern (Model-View-Controller). The focus of the illustrated architecture is on the communication of a Backend-For-Frontend (BFF) service with microservices (Microservice1 and Microservice2) using the Spring technology.

Note: Each of these services (BFF and microservices) run on different web servers which communicate via HTTP.

(View) The view in a microservice architecture is realized by the frontend. Technologies that can be used for the frontend implementation are Angular and Bootstrap.

There are two types of controllers in the architecture:
(BFF Controller) This is the controller directed to the frontend. It supports the frontend by transforming data for the view.
(Ms Controller) This is the controller related to a microservice. It starts the request and orchestrates the services.

There are also two types of models in the architecture:
(BFF Model) This is the data model that is underlying the BFF service.
(Ms Model) Each microservice has its own data model and its own database to persist the data specific to the one microservice.

BFF Backend For Frontend
MVC Model View Controller
Ms Microservice
POJO Plain Old Java Object

This page gives an overview of the modular architecture the Angular framework is based on.

(1) An Angular module is a class decorated with NgModule. Decorators (such as @NgModule) are functions that modify JavaScript classes.
Each Angular App has at least one such Angular module class, the root module (conventionally called AppModule).
The Angular module system is complementary to the JavaScript module system in which each file is a module and all objects defined in the file belong to that JavaScript module. The two types of module systems must be well separated which is not that easy since both use the same vocabulary of "imports" and "exports".

(Library Module) A library module is a collection of JavaScript modules. Angular ships with several such library modules, called Angular libraries each beginning with the @angular prefix. Examples are: @angular/core, @angular/platform-browser.

(2) A component controls parts of the screen called view and each defined by a template.
A template is a form of HTML that tells Angular how to render the component.
The component supports the view by providing application/presentation logic which is implemented as a class.
The mechanism for coordinating parts of a template and parts of a component is supported by the data binding concept of Angular. Data binding is conducted by adding markup bindings (specific forms) to the HTML template to tell Angular how to connect the component with the DOM (https://angular.io/guide/architecture#data-binding).

(3) A service can be nearly everything, such as a logging service or data service. Or it can be a tax calculator or an application configuration.
There is nothing specifically Angular about services. Angular has no definition of a service. There is no service base class, and no place to register a service.

(Injector) Angular uses dependency injection to provide new components with the services they need. When Angular creates a component, it first asks an injector for the services that the component requires.

Continuous Delivery (CD) and Build Pipelines

(1) CD treats each and every check-in as a release candidate
   (1) The software has to go through multiple stages from check-in to production
   (2) The multiple stages inside a build are modeled as build pipeline

(2) A CD tool provides specific functionality to support the build pipeline
   (1) Define and visualize the pipeline
   (2) Support manual actions when stage transitions are not fully automated

(3) One pipeline per microservice should be the goal
   (1) An exception might be acceptable in the initial project phase

Source: [CM-W-MIC] Cooperation & Management: MICROSERVICES AND API MANAGEMENT, WASA Course Unit. https://team.kit.edu/sites/cm-tm/Mitglieder/2-1.WASA

(1) Continuous Delivery (CD) gives constant feedback on the production readiness of each and every check-in [Ne15:107].
(1.1) The reason for the stages is that there is a sequence of the needed activities. An example is to conduct fast tests before slow tests (more general: to conduct different types of tests in a specific sequence).
(1.2) The figure on the right side shows the different stages of a build pipeline on an abstract level.

(2) The complexity of such a CD tool is often underestimated. It does not make sense to try to hack and extend CI tools to make them do CD.
(2.1) The tool supports the developer to keep a good overview of the process and the current status.
(2.2) In the case of a manually executed "User Acceptance Test" (UAT) the tool supports this process by showing the next available build ready to be deployed in the UAT environment, deploy it and show the test result in the visualized build pipeline.

(3) The service is an artifact (coming with different, platform-specific shapes) that is created and moved through the pipeline.
(3.1) In the initial phase, the service boundaries are often not clear and changes across service boundaries are more likely. As soon as a service API stabilizes it should be moved into an own build.

CD       Continuous Delivery
UAT      User Acceptance Test

After the software has been implemented it must be built and deployed which requires another specific set of tools.

(1) DevOps contains the two terms development and operations. It summarizes all practices that bring together software development and software operations in a way that the development cycles are shortened and the deployment frequency is increased. Microservice architectures support the DevOps concept since microservices have the property that they can be tested and deployed independently from the whole software system. This is not the case for monolithic systems.

(1.1) Continuous integration (CI) is a DevOps practice by which the code from the involved developers are built and tested in short time intervals (e.g. several times a day) in order to discover and solve integration problems as soon as possible.

(1.2) Continuous delivery (CD) extends the CI practice by releasing the software after it was built and tested (i.e. continuously integrated).

(1.3) Continuous deployment (also abbreviated by CD) which is often confused with continuous delivery, means that every change is automatically deployed to production. Since the software must be released before it can be deployed, continuous deployment is an extension of the DevOps practice of continuous delivery.

(Maven) A Java tool by which a project's build, reporting and documentation can be managed.
(Gradle) A Java-based build management tool (comparable to Apache Maven) which uses a domain-specific language
(cucumber) A library by which user acceptance tests based on Gherkin features can be carried out.
(Junit5) A framework to test Java programs based on unit tests.
(Nexus) A repository for build artifacts produced by Docker, Maven, npm, and others.
(Docker) A software system which supports container virtualization in order to deploy applications in isolated containers.
(Portainer) An open-source management user interface to manage a Docker host or swarm cluster.

Docker is a software product developed in Go by Docker Inc. The first version of Docker was published in 2013.

(1) Docker extends the existing container technology from Linux in a way that it provides a complete and easy-to-use solution for the creation and the operation of containers.

(2) Container virtualization offer applications the isolated usage of resources (CPU, RAM, network) without use of heavyweight virtual machines. Each container runs as an isolated process on the host operating system. This offers important advantages compared to virtual machine virtualizations, such as lightweight resource consumption (esp. CPU, storage), shorter start time, easier distribution.

(3) A container is a runnable instance of an image while an image is a read-only template with instructions for creating a Docker container. The standardized description leads to a portability of containers. In addition, deployment becomes easier and more stable.

(4) The instructions are formulated in a simple syntax. An example of a command that can be used in a Dockerfile is "docker run". 
"$ docker run -i -t ubuntu /bin/bash" (i) creates a new container containing an ubuntu image, (ii) allocates a read-write filesystem as its final layer, (iii) creates a network interface including the assignment of an IP address to the container, (iv) starts the container and executes the /bin/bash command.

(4.1) One important property of layers is that they are shared between all installed images.
In the example, the (operating system) layer "Ubuntu" and the (programming language) layer Java are shared by the applications A,B, and C.

[CM-W-MIC] Cooperation & Management: MICROSERVICES, WASA Course Unit. https://team.kit.edu/sites/cm-tm/Mitglieder/2-1.WASA
The figure illustrates an example of a microservice environment as it is established at C&M. In the guideline [CM-Mic] further technical information on the microservice environment of C&M can be found.

(Humane Service Registry) This registry is used during the development to guarantee a consistent design of the architecture. The humane service registry visualizes the service landscape by offering a user-defined list which is stored in the C&M Teamserver.

(Version Control) The source code of the services is organized with the version control system Git and stored at Bitbucket.

(CI/CD Pipeline) The CI/CD pipeline is a customized pipeline is executed whenever code is pushed to the repository. Hereby a step initiates the next one, when its run was successful. Each step runs in its own Linux Docker container and it's defined by the bitbucket-pipelines.yml which is stored in the version control system.

(Test Results) The test results are produced by running maven verify [Mav-Int]. It includes to validate whether all information, like dependencies, are available, compile the source code and to test the code. The project information and dependencies which are needed are given by the pom.xml.

(Runnable Artifact) Furthermore maven verify packages the compiled code to an executable. In our case it is an lightweight jar file. This jar is then stored as a pipeline artifact to be used in the next step.

(Docker Image) The Docker image is a ready to run container to be deployed in Docker. Therefore, information about the execution parameter for the jar, its location and exposed ports are needed. These and other parameters are stored in the Dockerfile.

(Archived Container) After the creation of the Docker image it is pushed to the nexus repository which is like a version control system for build artifacts. Information about the repository's location are given by the bitbucket-pipelines.yml.

(Running Container) Finally, the container previously running is removed from the Docker host and the new one is deployed from the repository. Therefore, any version of another container can be deployed from the repository. This might be necessary if there are certain dependencies.

CD Continuous Deployment
CI Continuous Integration

(1) ++Overview++
++Separation into Frontend and Backend Implementation++
- A prerequisite for the start of this implementation is the existence of the API specifications of the microservice architecture.

(2) ++Spring-Based Backend Architecture++
- The pattern is called MVC (Model-View-Controller).
- Backend components: BFF Controller, BFF Model, Microservice Controller, Microservice Model

(3) ++Continuous Delivery (CD) and Build Pipelines++
- CD treats each and every check-in as a release candidate
- CD requires a build pipeline which consists of several stages (resulting from different fast/slow/user acceptance/performance tests) which the software must pass.

(4) ++Docker++
- A Docker container runs as an isolated process on the host operating system and provides a virtualized environment to run applications.
- A Docker container is a runnable instance of an image.
- Created by instructions in a Docker file

(5) ++Microservice Environment at C&M++
- The result is a application container image (Docker image) which runs in a container (Docker container).
The initiator of the Spring Boot demo project is Joshua Schnabel [Sc19].

(1) The project is available on Bitbucket [Bit-Spr]. It can be checked out and used via Git. It should serve as a template and example of a Spring Boot app and can be used as a foundation for new projects.

Hint: There are several copies of the repository available in Bitbucket. The original repository is part of the team icconsult-kit.

(2) The following installation is recommended for a Spring Boot development:

(2.1) The download page of the Spring Tool Suite (STS, also known as Spring IDE) is [Spr-STS].

Note: If Eclipse is used, the STS can also be downloaded via the Eclipse marketplace ("Help" -> "Eclipse Marketplace" -> Find: "STS").

(2.2) Lombok is an open source library (basically a standalone jar) which is capable of automating the generation of boilerplate code (supporting code with no explicit logic, esp. getter & setter methods, class constructors, hashcode and equals methods) for any Java class. The generated code does not contribute to the understanding of the code and considerably reduces lines of code per class. This makes it possible to keep the source code structured and easy to read. A description how to install Lombok in Eclipse can be found in [How-Lom].

(2.3) With Cucumber, feature descriptions written in Gherkin can be transformed into code. A plugin is required for the transformation. This can be obtained from the Eclipse marketplace [Mar-Cuc].

(2.4) STS already provides a Git integration which quality is inadequate. Therefore, it is recommended to use the Git via command line [Git-Dow].

(3) It is also recommended to use the command line utility of Maven. But, STS also provides an integration here.

(3) The README file in the project folder provides more information on setting up the development environment and using the demo.

STS  Spring Tool Suite


[Sc19] Joshua Schnabel: Entwicklung einer sicheren Zugriffslösung von einem mobilen Endgerät auf Connected-Car-Backendsysteme, Masterarbeit, Karlsruher Institut für Technologie (KIT), C&M (Prof. Abeck), 2018

The sample project is a simple todo administration software.

1. It should be possible to create a new todo list and add it to the existing lists of todos. Lists can have a title and an unlimited number of todos.

2. Existing lists can be edited.

3. The title can be changed or a new todo can be added to the list.

4. It is also possible to mark a todo as done.

5. Another requirement is to delete existing lists.

(Feature: Manage the todos of user) The requirements for the software system are presented as Cucumber features which are written in a language called Gherkin. Most of the feature description is written in a natural language. In order to allow interpretation with the computer, Gherkin prescribes some keywords (in the example: Feature, Background, Scenario, Given When, Then And) which must be considered. Gherkin is used by the Cucumber framework [Cuc-Ger].

(edu.kit.cm.springBootDemo.cucumber.RunCucumberTests) In all later phases of the software development process the definition can be used to test the program. In this way, it is possible to test whether a program meets the requirements and, in case of changes to the code, to check whether all requirements are still being met. The tests are started as junit tests and can be integrated into all existing tools [Jun-Ju4].


The domain model is derived from the requirements expressed by Gherkin features on the previous page. The UML class diagram above shows the domain model which only consists of two entities (TodoList, Todo) and two value objects (Status, Value).

TodoList The TodoList aggregate root represents a todo list. It contains both the title of the list and the todos that belong to it.

Todo The Todo entity represents a single todo of a list. It does not contain any attributes, but has a relationship to a status and value.

Value The Value contains the text of the todo.

User The User aggregate root represents a User. It holds its attributes as well as its todo lists.

Based on the domain model the Java classes are designed. The Java class design contains additional classes that are not directly related to the domain model.

TodoListRepository Repositories allow access to entities. This repository allows access to todo lists.

UserRepository There is also a repository for accessing users.

TodoListId A reference to the concrete TodoList via its Id.

TodoId A reference to the concrete Todo via its Id.

UserId A reference to the concrete User via its Id.

The TodoId and the UserId are introduced to simplify the handling of later adapters.

By separating the relations by introducing Id objects, it is possible to load parts of the domain without instantiating the entire domain. With such a small domain, this is not absolutely necessary, but it should still show this approach.
The infrastructure forms the outermost layer. It contains the ports and adapters. In the demo, this was placed under package edu.kit.cm.springBootDemo.domain.

1.1 The presentation port provides functions which are necessary for the presentation of the data. It is a Spring component so it can be autowired.

1.1.1 The REST (Representational State Transfer) adapter uses the functions of the presentation port to offer the data as REST services. It is an active port, so it receives data from outside. A Spring controller was used here, more specifically aRestController. This allows a complete configuration via Java annotations.

1.1.2 The HTML (Hypertext Markup Language) adapter offers the data as HTML frontend. It also uses the presentation port and is a Spring MVC (Model-View-Controller) controller. Like the REST adapter it is an active port.

1.2 The JPA (Java Persistence API) adapter does not have a port. Especially with passive adapters it happens that no port is needed. This passive port passes data on to the downstream system. In this specific case, it is a database.

(2) The application layer is not implemented. This small example does not require any further components in the application layer. For example, authentication and authorization could be implemented in this layer. In the demo, this was placed under package edu.kit.cm.springBootDemo.application.

(3) The domain layer contains the entities and repositories described above. In the demo, this was placed under package edu.kit.cm.springBootDemo.domain.

(4) This project has implemented several types of tests. Each type of test has a different purpose.

4.1 Cucumber tests are surface tests performed using Selenium. The application is completely started up during this test. It is tested from the interface to the database, a so-called end-to-end test.

4.2 Integration tests are also end-to-end tests. They can be used to test interfaces other than a UI (User Interface). In the demo project the integration tests were used to test the REST interfaces.

4.3 Unit tests are used to test a special class. All required classes must be replaced by mocks.Mocks are representations of classes that only simulate behavior.

REST Representational State Transfer
HTML Hypertext Markup Language
JPA Java Persistence API
MVC Model-View-Controller
UI User Interface
Bootstrap is an open source toolkit for development with HTML, CSS and JS. In addition, it offers many components such as a fast-reacting grid system, extensive prefabricated components and powerful plug-ins [Bot-Bot].

Thymeleaf is a modern server-side Java template engine for web and standalone environments [Thy-Jav].

The Spring Framework is an open source framework for the Java platform. The aim of the Spring Framework is to simplify the development with Java/Java EE and to promote good programming practices. With a wide range of functions, Spring offers an integrated solution for the development of applications and their business logic, focusing on the decoupling of application components [Spr-Fra].

Spring Boot is designed to make it even easier to create Spring applications. It manages the Maven modules and allows configuration which is not based on XML files [Spr-Boo].

Spring MVC is an extension to Spring that allows to present content. For example, it can provide the content via REST, but surfaces can also be generated by including a template engine like Thymeleaf.

Spring Data JPA is a part of the larger Spring Data family which makes it easy to implement JPA-based repositories. This module deals with extended support for JPA-based data access layers. It makes it easier to develop Spring-based applications that use data access technologies [Spr-JPA].

H2 Database Engine is a lean relational database. It is well suited for small software projects and tests, which in addition to the classical server operation can also store the data in a file or in the memory. This is necessary for unit tests. This means that tests can always be performed with a clean database [H2-Dat].

<table>
<thead>
<tr>
<th>API</th>
<th>Application Programming Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSS</td>
<td>Cascading Style Sheets</td>
</tr>
<tr>
<td>JPA</td>
<td>Java Persistence API</td>
</tr>
<tr>
<td>JS</td>
<td>JavaScript</td>
</tr>
<tr>
<td>MVC</td>
<td>Model View Controller</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
</tbody>
</table>


The code example shows a section of a class of the REST adapter (TodoListRestAdapter, line 4).

1.1 The purpose of the class is to provide the todo lists via REST. The class offers methods for creating new lists and editing existing lists.

1.2 Spring MVC functionalities were used for the implementation.

1. The @RestController annotation says Spring has a REST controller here. It is automatically included during startup.

2. The @RequestMapping annotation describes the path under which the REST methods are to be provided. This is the base path. Each method must define its own path based on the base path.

3. The @Api annotation tells Spring that an API is available here. An OpenAPI specification is automatically created from the REST methods. This can be displayed later with Swagger, for example.

7. An Inject annotation indicates that Spring should inject this object. This is left up to Spring, where this instance of the object comes from.

10. The OpenAPI specification is enriched by information specified via the @ApiOperation annotation. This information includes a description, the return type and much more.

11. The @RequestMapping now describes the REST method. As already mentioned, the base path can be extended here. In addition, allowed HTTP methods and the content type can be defined.

13. The Java method only passes the objects provided by the port.
EXERCISES: Lombok, Cucumber, Entity Ids, H2 Database

(1) ++Spring Boot Demo++
- Better code which is easier to understand
- Slimmer and cleaner classes

(2) ++Requirements++
- Test whether a program meets the requirements
- Allows to check after code change if program still behaves as required

(3) ++Domain Model++
- Allow to load only parts of the domain model

(4) ++Technologies Related to Spring Boot++
- Stores Data in memory
- Optimal for unit tests
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>BDD</td>
<td>Behavior-Driven Development</td>
</tr>
<tr>
<td>BFF</td>
<td>Backend For Frontend</td>
</tr>
<tr>
<td>CD</td>
<td>Continuous Delivery</td>
</tr>
<tr>
<td>CI</td>
<td>Continuous Integration</td>
</tr>
<tr>
<td>COAP</td>
<td>Constrained Application Protocol</td>
</tr>
<tr>
<td>CRUD</td>
<td>Create, Read, Update, Delete</td>
</tr>
<tr>
<td>CVS</td>
<td>Concurrent Versions System</td>
</tr>
<tr>
<td>C&amp;M</td>
<td>Cooperation &amp; Management</td>
</tr>
<tr>
<td>DDD</td>
<td>Domain-Driven Design</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hyper Text Transfer Protocol</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>JPA</td>
<td>Java Persistence API</td>
</tr>
<tr>
<td>KIT</td>
<td>Karlsruhe Institute of Technology</td>
</tr>
<tr>
<td>MVC</td>
<td>Model View Controller</td>
</tr>
<tr>
<td>MVP</td>
<td>Minimum Viable Product</td>
</tr>
<tr>
<td>NavSG</td>
<td>NavigationServiceGroup</td>
</tr>
<tr>
<td>POI</td>
<td>PointOfInterest</td>
</tr>
<tr>
<td>POJO</td>
<td>Plain Old Java Object</td>
</tr>
<tr>
<td>RAML</td>
<td>RESTful API Modeling Language</td>
</tr>
<tr>
<td>RCS</td>
<td>Revision Control System</td>
</tr>
<tr>
<td>REST</td>
<td>REpresentational State Transfer</td>
</tr>
<tr>
<td>RM</td>
<td>Resource Model</td>
</tr>
<tr>
<td>SemVer</td>
<td>Semantic Versioning</td>
</tr>
<tr>
<td>TFS</td>
<td>Team Foundation Server</td>
</tr>
<tr>
<td>UAT</td>
<td>User Acceptance Test</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>URI</td>
<td>Universal Resource Identification</td>
</tr>
<tr>
<td>VCS</td>
<td>Version Control System</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
</tbody>
</table>