Gr. 1: Kurschl
Gr. 2: Kurschl
Gr. 3: Nadschläger

Software Engineering

Übungen zu SWK5/WEA5 im WS 2019/20

Name _____ Aufwand in h _____

Hurace

Ein System zur Abwicklung von Ski-Events

Trotz des Rücktritts von Marcel Hirscher ist Österreich nach wie vor die Skination Nummer Eins. Nirgendwo sonst ziehen Skirennen derart viel Publikum an wie bei uns. Die Kitzbüheler Hahnenkammrennen und das Schladminger Night Race zählen zu den Höhenpunkten der Skisaison. Um die Veranstalter von derartigen Ski-Events zu unterstützen, werden Sie im Rahmen des SWK5/WEA5-Semesterprojekts mit der Implementierung eines Softwaresystems beauftragt, das die Abwicklung und die multimediale Aufbereitung von Skirennen auf eine neue Ebene heben soll.

Funktionale Anforderungen

Das Softwaresystem muss auf zwei Benutzergruppen ausgerichtet werden. Dem Veranstalter eines Skirennens soll es helfen, das Event regelkonform durchzuführen und die Zuschauer während des Rennens mit den nötigen Informationen zu versorgen. Die Fans im Skistadion sollen transparent über den Stand im Rennen informiert werden. Ein wesentliches Merkmal von *Hurace* ist es aber, dass man das Rennen auch über das Web verfolgen kann und interessante Informationen über die Skihelden abrufen kann. Im Wesentlichen bietet *Hurace* folgende Funktionen:

- *Verwaltung der Rennläuferinnen:* Die Stammdaten der Rennläuferinnen können über ein Web-Portal gepflegt werden. Fans haben auf diese Daten lesenden Zugriff.
- *Auslosung:* Am Vorabend eines Skirennens werden die Startnummern ausgelost. Die dadurch festgelegte Startreihenfolge muss über die Benutzeroberfläche erfasst werden können.
- *Durchführung des Rennens:* Die Rennleitung kann über eine Bedienoberfläche den Ablauf des Rennens und die Anzeigetafel im Skistadion steuern. Die Anzeigetafel enthält verschiedene Screens, über welche die Zuseher beispielsweise über den Zwischenstand des Rennens und die Zeit des aktuellen Fahrers informiert werden.
- *Abbildung des Reglements:* Das Softwaresystem muss die unterschiedlichen Regeln für die Durchführung der vier Wettbewerbstypen Abfahrt, Super-G, Riesenslalom und Slalom (aber keine Parallelbewerbe oder die alpine Kombination) berücksichtigen. Auch die unterschiedlichen Regeln für die Bestimmung der Startreihenfolge sind umzusetzen. Der zu realisierende Funktionsumfang ist von der Teamgröße abhängig:
    -  Wird das Projekt von einer Person entwickelt, müssen Abfahrtsrennen und Super-Gs (Rennen wird in einem Durchgang durchgeführt) unterstützt werden.
    -  Zweierteams müssen die Disziplinen Slalom und Riesenslalom (Rennen wird in zwei Durchgängen durchgeführt) realisieren.

Das Softwaresystem kann natürlich sowohl für Damen- als auch für Herrenrennen eingesetzt werden. Aus Gründen der Lesbarkeit wird in diesem Dokument bei Personenbezeichnungen die weibliche Form gewählt, es ist jedoch auch immer die männliche Form gemeint.

Das zu entwickelnde Softwaresystem besteht aus den folgenden Komponenten:

- *Hurace.Core* ist die zentrale Komponente, welche die Geschäftslogik beinhaltet und für die Datenverwaltung zuständig ist. Sie stellt die von *Hurace.RaceControl* und *Hurace.Web* benötige Funktionalität zur Verfügung und kapselt den Zugriff auf die Datenbank. Auch die Interaktion mit externen Systemen wie der Zeitnehmung gehört zu den Aufgaben dieser Komponente.
- *Hurace.Api* exportiert die in *Hurace.*Web benötigte Funktionalität von *Hurace.Core* in Form eines REST-basierten Web-Service.
- *Hurace.RaceControl* dient primär zur Darstellung des aktuellen Rennstandes und zur Visualisierung der Rennereignisse. Die Rennleitung kann über diese Bedienoberfläche die Anzeigetafel steuern und so auf aktuelle Ereignisse während des Rennens reagieren. Die Anwendung ermöglicht es dem Veranstalter auch, diverse vorbereitende Maßnahmen zu treffen, wie beispielsweise die Verwaltung der Startreihenfolge.
- *Hurace.Web:* Der Web-Client stellt ebenfalls die aktuelle Rennlage dar und bietet Fans die Möglichkeit, die Profile der Rennläuferinnen abzurufen. Über eine mit Login abgesicherte Seite können die Stammdaten der Rennläuferinnen gepflegt werden.
- *Hurace.Simulator:* Diese Komponente ist ein Werkzeug für den Entwickler, das zum Testen und zur Präsentation des Systems benötigt wird.

(a) Datenmodell

Im Folgenden werden die wesentlichen Attribute der wichtigsten Entitäten des Systems definiert. Aus der Anforderungsspezifikation können sich zusätzliche Entitäten und weitere Eigenschaften der angeführten Entitäten ergeben, die entsprechend zu ergänzen sind.

(a.1) *Race:* Für jedes Skirennen sind das Datum, die Bezeichnung, die Rennart (Slalom, Riesenslalom, Super-G, Abfahrt), der Standort, sowie eine Beschreibung als Freitext zu speichern.

(a.2) *Skier:* Hier werden die Stammdaten (Name, Herkunftsland, Geburtsdatum, Profilbild etc.) der Rennläuferinnen gespeichert.

(a.3) *StartList:* Diese Entität bildet die Zuordnung einer Rennläuferin zu einem Rennen ab. Hier wird auch ihre Startnummer gespeichert.

(a.4) *RaceData:* Sämtliche während eines Rennens anfallenden Zwischenzeiten werden in dieser Entität gespeichert. Auch andere Statusinformationen wie eine Disqualifikation kann hier abgelegt werden.

(b) Hurace.Core

Hurace.Core ist die zentrale Stelle im System, in dem der Zustand eines Rennens verwaltet wird. Diese Komponente ist dafür verantwortlich, die von die Clients und vom Zeitmesssystem gelieferten Daten entgegenzunehmen, diese aufzubereiten und in der Datenbank zu persistieren.

(b.1) *Verwaltung der Rennläuferinnen:* Die Serverkomponente muss eine geeignete Schnittstelle zur Verwaltung der Stammdaten der Rennläuferinnen (hinzufügen, löschen, ändern) anbieten. Bei Löschoperationen ist ein Datensatz als gelöscht zu markieren, falls bereits Fremdschlüsselbeziehungen existieren.

(b.2) *Interaktion mit dem Zeitmesssystem:* Auf das Zeitmesssystem kann mithilfe eines Treibers zugegriffen werden, der vom Hersteller zur Verfügung gestellt wird. Dieser Treiber wird als .NET-Standardkomponente implementiert, die über folgende Schnittstelle angesprochen werden kann:

```
namespace Hurace.Timer
{
  public delegate void TimingTriggeredHandler(int sensorId,
                                              DateTime time);

  public interface IRaceClock
  {
    public event TimingTriggeredHandler TimingTriggered;
  }
}
```

Die Zeitnehmung liefert bei der Auslösung durch die Rennläuferinnen den ersten Impuls. `sensorId` ist eine fortlaufende Zahlenfolge, die mit 0 beginnt. *Hurace.Core* ist für die Zuordnung der Zeitmessungen zu den Rennläuferinnen zuständig. Um die Zuordnung zu erleichtern, wird über die Bedienoberfläche das Rennen explizit freigegeben. Zeitimpulse sind nur von der Rennfreigabe bis zum Überqueren der Ziellinie weiterzuverarbeiten.

Berücksichtigen Sie bei Ihrer Implementierung, dass es verschiedene Hersteller von Zeitmesssystemen gibt und diese daher einfach ausgetauscht werden können. Gehen Sie auch davon aus, dass die Initialisierung dieser Komponente ein zeitaufwändiger Prozess sein kann und daher nur eine Instanz dieser Komponente existieren sollte.

Gehen Sie auch davon aus, dass das Zeitmesssystem zahlreichen Störungen unterliegen kann. Beispielsweise können Zwischenzeitnehmungen teilweise ausfallen oder Lichtschranken versehentlich durch Pistenarbeiter ausgelöst werden. Das Messsystem liefert immer Zeitimpulse, auch zu Zeiten, in denen das Rennen nicht freigegeben ist. Implementieren Sie daher diese Komponente besonders robust und fehlerresistent.

(b.3) *Erfassung der Renndaten:* Die während des Rennens anfallenden Daten werden nicht nur den Clients zur Visualisierung zur Verfügung gestellt, sie müssen auch in der Datenbank abgelegt werden. Dieser Datenbestand ist die Basis für Abfragen während des Rennens und Analysen nach dem Rennen.

(b.4) *Wiederaufsetzen:* Der Zustand der Anwendung ist laufend und konsistent in der Datenbank zu speichern, sodass nach einem (geplanten oder erzwungenen) Neustart der Anwendung die Clients Ihre Arbeit fortsetzen können und es zu keinem Datenverlust kommt. Wird die Anwendung während einer Fahrt beendet, muss die Anwendung imstande sein, ihre Arbeit mit dem Start des nächsten Rennläufers wieder aufzunehmen.

(b.5) *Abfragen: Hurace.Core* stellt den Clients die von diesen benötigten Daten (z. B. aktuelle Rangliste) zu Verfügung. Die Daten werden entsprechend den Anforderungen der Clients gefiltert und aggregiert.

(c) Hurace.Api

Die für *Hurace.Web* und *Hurace.Simulator* erforderliche Funktionalität ist in Form eines REST-basierten Web-Service zu exportieren. Der Zugriff auf das Web-Service muss nicht abgesichert werden.

(d) Hurace.RaceControl

Hurace.RaceControl stellt eine grafische Benutzeroberfläche zur Verfügung, mit dem der Veranstalter in der Vorbereitungsphase die für ein Rennen notwendigen Daten erfassen und administrieren kann. Die zentrale Funktionalität dieses Clients ist aber die Unterstützung bei der Rennabwicklung.

(d.1) *Verwaltung von Rennen:* Bei der Erstellung eines neuen Rennens werden die entsprechenden Stammdaten erfasst (Datum, Bezeichnung etc.). Es können auch bestehende Rennen ausgewählt werden, zum Beispiel nach dem (gewollten oder ungewollten) Neustart des Programms.

(d.2) *Zuordnung von Rennläuferinnen:* Vor dem Start eines Rennens können die teilnehmenden Läuferinnen ausgewählt und die Startreihenfolge verwaltet werden. Achten Sie hier auf eine auch für wenig computeraffine Menschen intuitive Bedienung.

(d.3) *Steuerung des Rennablaufs*: Während eines Rennens sieht der Rennleiter, welche Fahrerin gerade aktiv ist sowie die nachfolgende Fahrerin. Er hat hier sämtliche Steuerungsmöglichkeiten, die für die Abwicklung des Rennens notwendig ist, zum Beispiel:

- Freigabe der Strecke für die nächste Fahrerin (und damit Aktivierung der Zeitnehmung). In der Regel wird die vor Rennbeginn festgelegte Startreihenfolge streng eingehalten. Es kann jedoch auch vorkommen, dass eine Rennfahrerin nicht antritt.
- Disqualifikation der aktuellen Fahrerin.
- Nachträgliche Disqualifikation einer Fahrerin, die sich beispielswiese aus der Videoanalyse ergeben hat.
- Beenden des Rennens

(d.4) *Steuerung der Anzeigetafel:* Die Anzeigetafel des Stadions wird von *Hurace.RaceControl* befüllt. Dazu kann einer von mehreren vordefinierten Screens ausgewählt werden. Der aktuell ausgewählte Screen wird in einem zweiten Fenster dargestellt, welches auch auf der Anzeigetafel sichtbar ist. Implementieren Sie für die erste Version zumindest folgende Screens:

- Aktuelle Läuferin: Dieser Screen wird vor allem verwendet, während eine Läuferin gerade auf der Strecke unterwegs ist. Dieser zeigt Daten zur aktuellen Läuferin (Name, Nationalität, Foto, Startnummer, …) sowie die laufende Zeit. Außerdem werden die Zwischenzeiten und der jeweilige Vorsprung/Rückstand auf die führende Läuferin sowie die aktuelle Platzierung angezeigt.
- Zwischenstand: Hier werden tabellarisch die Zeiten und Differenzen aller bisher gefahrenen Läuferinnen dargestellt. Die zuletzt gefahrene Läuferin wird grafisch hervorgehoben.

Bei Rennen, die in zwei Durchgängen durchgeführt werden, müssen im zweiten Durchgang zusätzliche Daten angezeigt werden. Dazu zählen der Stand nach dem ersten Durchgang, der die Startreihenfolge im zweiten Durchgang festlegt, oder Informationen darüber, wie sich die Gesamtzeit zusammensetzt (Platzierung im ersten und zweiten Durchgang sowie die Gesamtplatzierung).

Berücksichtigen Sie bei der Implementierung, dass in zukünftigen Versionen noch weitere Screens hinzukommen können (z.B. Logo des Veranstalters, Werbeeinblendungen, Live-Bilder…) und stellen Sie sicher, dass diese einfach in Ihre Architektur integriert werden können.

Die Anzeigetafel dient vielen tausend Menschen im Stadion als Hauptinformationsquelle, legen Sie hier also besonders viel Wert auf eine grafisch ansprechende Umsetzung.

(d.5) *Kapselung der Logik: Hurace.RaceControl* ist als leichtgewichtiges Frontend zu implementieren. Achten Sie darauf, dass die Rennlogik (welche Fahrerin ist gerade aktiv, wie ist die aktuelle Rangliste, …) und die Behandlung der Zeitnehmung in *Hurace.Core* gekapselt sind und *Hurace.RaceControl* diese Komponenten nutzt, selbst aber nur wenig Logik beinhaltet.

(e) Hurace.Simulator

Diese Komponente ist ein wichtiges Werkzeug für den Entwickler, mit dem er Integrationstests durchführen kann und die Funktionalität des Softwaresystems demonstrieren kann. *Hurace.Simulator* muss

nicht als eigenständiger Client realisiert werden, sondern kann in *Hurace.RaceControl* integriert werden.

(e.1) *Simulation der Zeitnehmung:* Der Simulator stellt eine Implementierung von `IRaceClock` zur Verfügung, mit der die Zeitnehmung simuliert werden kann. Diese Implementierung muss es ermöglichen, die Geschwindigkeit der Zeitimpulse zu beeinflussen. Die Zeitnehmung muss jederzeit angehalten und wieder gestartet werden können. Anomalien in der Zeitnehmung wie ausbleibende Messimpulse oder versehentlich ausgelöste Lichtschranken müssen ebenfalls nachgestellt werden können.

(e.2) *Benutzerinteraktion*: Die in (e.1) definierten Funktionen können über eine einfache grafische Bedienoberfläche angesprochen werden.

(e.3) *Kapselung*: Da diese Komponente nicht Bestandteil des Produktivsystems ist, soll sie gut gekapselt werden und lose an die anderen Komponenten gekoppelt sein.

Sie müssen auch damit rechnen, dass Ihnen vor Auslieferung des Produkts vom Auftraggeber eine weitere Implementierung dieser Komponente zur Verfügung gestellt wird, die Sie in Ihr Produkt integrieren müssen.

(f) Hurace.Web

Der Web-Client soll allen Ski-Interessierten die Möglichkeit bieten, im Internet das Rennen live zu verfolgen, einzelne Rennen zu analysieren, sowie sich einen Saison-Überblick zu verschaffen. Darüber hinaus soll es aber auch der FIS (Fédération Internationale de Ski) als Veranstalter von Rennen möglich sein, Stammdaten (d.h. die Skirennläuferinnen einer Nation) zu verwalten und Sportlerinnen für Rennen zu nominieren.

Dafür sind folgende Funktionalitäten bereitzustellen:

(f.1) *Live-Anzeige eines Rennens:* Hier soll die temporäre Rangliste (Name, Nationalität, Endzeit) während des Rennens, die Zwischenzeiten der aktuellen Läuferin sowie das Endergebnis angezeigt werden. Beachten Sie bitte, dass manche Rennen (bspw. Slalom) aus zwei Durchgängen bestehen.

(f.2) *Verwalten von Stammdaten:* Für ein Rennen nominiert die FIS einzelne Läuferinnen. Dazu werden alle notwendigen Stammdaten erfasst. Beachten Sie, dass Läuferinnen nicht für jedes Rennen neu angelegt werden, sondern mit *Hurace.RaceControl* verschiedenen Rennen zugeordnet werden können. Da der Pool an Läuferinnen (Frauen und Männer) meistens mehr als 100 umfasst, kann eine Suche nach Läuferinnen hilfreich sein.

Jene FIS-Mitarbeiterinnen, die Stammdaten bearbeiten, müssen sich vorher über OAuth2 authentifizieren. Nur Ihnen ist es gestattet, die Daten zu bearbeiten. Hinweis: Es ist NICHT erforderlich, dass Sie das REST-Service diesbezüglich absichern!

Sie können hier auch auf den in der Übung eingesetzten Identity-Server, der von Manfred Steyer in der Azure-Cloud gehostet wird, zugreifen. Wer möchte, kann aber auch andere Dienste wie Auth0 oder Open-Source-Lösungen wie Keycloak verwenden.

(f.3) *Saisonübersicht:* Eine Saison besteht üblicherweise aus unterschiedlichen Austragungsorten. An einem Austragungsort werden dann Rennen unterschiedlicher Disziplinen (Slalom, RTL, Super-G und Abfahrt) durchgeführt. Beachten Sie, dass bspw. in Kitzbühel nur Abfahrt, Super-G und Slalom ausgetragen werden. Führen Sie also nur jene Disziplinen pro Ort an, die tatsächlich vorgesehen sind. Gruppieren Sie Ihre Saisonübersicht nach den einzelnen Disziplinen.

(f.4) *Rennanalyse [optional]:* Damit sich unsere Hobby-Skirennläuferinnen sowie die Co-Kommentatoren (Assinger, Sikora, Meisnitzer, etc.) einen raschen Überblick über die vergangenen Erfolge einer Skirennläuferin machen können, wäre eine Suche nach Person und Anzeige der Ergebnisse einer Saison sehr hilfreich. Eventuell lassen sich sogar die Anzahl der 1., 2. und 3. Plätze auch noch kumulativ aufführen.

Für die Rennanalyse ist es u.a. hilfreich, wenn man bei einem Rennen auch die Zwischenzeiten ausgewählter Personen vergleichen könnte. So sieht man, wo die Skirennläuferinnen Zeit gewonnen bzw. verloren haben. Eventuell ist dafür eine graphische Aufbereitung hilfreich.

Ergebnisse

Die in der Anforderungsdefinition festgelegten Funktionen sind in Einer- oder Zweierteams zu realisieren. Anforderungen, die mit dem Symbol 👪 gekennzeichnet sind, müssen nur von Zweierteams, jene, die mit 👤 markiert sind, nur von Einerteams ausgearbeitet werden. Alle anderen Anfordergen gelten für beide Gruppen. Die WEA5 zuzuordnende Funktionalität ist in Form von Einzelarbeiten umzusetzen.

Die Entwicklung der Projektarbeit ist auf GitHub durchzuführen. Alle Teams bekommen ein Repository zugewiesen, auf welches das Team und die Übungsleiter Zugriff haben.

Erstellen Sie für alle Dokumente und Quelltext-Dateien eine übersichtliche Verzeichnisstruktur und packen Sie diese in eine ZIP-Datei. Dieses Archiv stellen Sie Ihrem Übungsleiter auf der E-Learning-Plattform in den Kursen zur SWK5-Übung bzw. zu WEA5 zur Verfügung. Achten Sie darauf, dass Sie nur die relevanten (keine generierten) Dateien in das Archiv geben. Große Archive mit unnötigen Dateien vergeuden Speicherplatz und können daher zu Punkteabzügen führen.

Große Binärdateien, wie Komponenten von Drittherstellern oder Datenbanken dürfen keinesfalls im Repository abgelegt werden. Checken Sie anstatt dessen Scripts zur Erzeugung des Datenbankschemas und zum Befüllen der Datenbank ein. Dokumentieren Sie, wie Ihr System in Betrieb genommen werden kann.

Konzentrieren Sie sich bei der Dokumentation dieser Projektarbeit auf die Darstellung der Architektur und des Designs Ihrer Anwendung. Verwenden Sie dazu gängige Darstellungsformen wie ER- und UML-Diagramme (Use-Case-, Klassen- und Sequenzdiagramme). Zeigen Sie anhand von zwei aussagekräftigen Anwendungsfällen u. a. die Interaktionen der verschiedenen Systemkomponenten mithilfe eines Sequenzdiagramms – auch über Systemgrenzen hinweg (WEA5 und SWK5). Versuchen Sie diesen Teil der Arbeit besonders übersichtlich zu gestalten. Die Dokumente sind ausschließlich in Form von PDF-Dateien abzugeben.

Ausbaustufe 1

(Abgabe: 17. 11. 2019, 24:00 Uhr)

Sämtliche Komponenten von *Hurace* sind mit den in den Lehrveranstaltungen SWK5 und WEA5 behandelten Technologien zu realisieren. In dieser ersten Ausbaustufe benötigen Sie nur das .NET-Core-Framework.

(A1.1) *Datenbank/Datenmodell:* Alle auf der *Hurace*-Plattform erfassten Daten sind in einer relationalen Datenbank zu speichern. Entwerfen Sie ein strukturiertes (normalisiertes) Datenmodell für *Hurace*. Erstellen Sie eine Testdatenbank, welche für die Problemstellung repräsentative und ausreichend umfangreiche Daten enthält.

(A1.2) Gehen Sie von folgendem Mengengerüst aus: In Ihrer Testdatenbank sollten zumindest 70 aktive Skirennläufer und 10 Skirennen erfasst sein. Im Durchschnitt gibt es 5 Zwischenzeiten pro Rennen. Gehen Sie außerdem davon aus, dass an einem Rennen durchschnittlich 50 Läufer teilnehmen. Aus diesen Parametern leitet sich der Umfang der erfassten Renndaten (Zwischenzeiten) ab.

Gestalten Sie Ihr Datenmaterial soweit wie möglich realistisch. Einen Teil der Daten dürfen Sie auch generieren. Verwenden Sie zum automatisierten Testen und für den Produktivbetrieb unterschiedliche Datenbanken.

(A1.3) *Datenzugriffsschicht:* Überlegen Sie sich für die Entitäten Ihres Datenmodells (*Race, Skier* etc.) die erforderlichen Zugriffsfunktionen und fassen Sie diese zu Interfaces zusammen. Die Geschäftslogik soll ausschließlich von diesen Interfaces abhängig sein.

Überlegen Sie sich ein geeignetes Format für die Repräsentation der Daten. Definieren Sie dazu einfache Klassen (die primär aus Konstruktoren und Properties bestehen), die zum Transport der Daten dienen. Diese Klassen werden häufig als *Domänenklassen* oder *Datentransferklassen* bezeichnet. Im Wesentlichen werden Sie für jede Entität eine entsprechende Domänenklasse benötigen. Die Domänenklassen werden in den Interface-Methoden der Datenzugriffsschicht als Parametertypen verwendet und müssen daher auch der Geschäftslogik bekannt gemacht werden.

Implementieren Sie die Datenzugriffsschicht auf Basis von ADO.NET. Der Einsatz eines OR-Mapping-Werkzeugs (NHibernate, Entity Framework etc.) ist nicht gestattet.

(A1.4) *Unit-Tests:* Testen Sie die Datenzugriffsschicht ausführlich, indem Sie eine umfangreiche Unit-Test-Suite erstellen. Achten Sie auf eine möglichst große Testabdeckung. Erstellen Sie eher mehr, dafür aber kompakte Unit-Tests, die nach Möglichkeit nur einen Aspekt der Funktionalität überprüfen. Für jede Methode sollte zumindest ein Unit-Test existieren. Die Wahl des Testframeworks steht Ihnen frei.

(A1.5) *Dokumentation:* Erstellen Sie eine übersichtliche, gut strukturierte Dokumentation. Ihre Dokumentation sollte zumindest das Datenbankmodell und die Struktur der Datenzugriffsschicht enthalten.

Ausbaustufe 2

(Abgabe: 22. 12. 2019, 24:00 Uhr)

(A2.1) *Geschäftslogik:* Implementieren Sie auf Basis der in Ausbaustufe 1 entwickelten Datenzugriffsschicht die gesamte Funktionalität der Komponente *Hurace.Core*. Überlegen Sie sich, wie die Klienten auf die (Geschäfts-)Logik zugreifen sollen. Fassen Sie die erforderlichen Funktionen zu logischen Gruppen zusammen und definieren Sie für jede Gruppe ein Interface. Alle Klienten sollen (statisch) ausschließlich von diesen Geschäftslogik-Interfaces abhängig sein.

Entwerfen Sie die Klassen der Geschäftslogik mit besonderer Sorgfalt und legen Sie besonderen Wert auf eine flexible Software-Architektur. Sorgen Sie insbesondere dafür, dass das Zeitmesssystem einfach ausgetauscht werden kann. Implementieren Sie alle Funktionen, die nicht unmittelbar für die Umsetzung der Benutzeroberflächen benötigt werden, in der Geschäftslogikkomponente von *Hurace.Core*.

(A2.2) *Automatisierte Tests:* Erweitern Sie Ihre Test-Suite um Tests der Geschäftslogik. Die Tests sollen soweit wie möglich die gesamte Funktionalität der Anwendung abdecken.

Isolieren Sie in Ihren Tests die Geschäftslogik von der Persistenzschicht. Ersetzen Sie dazu Ihre DAOs durch Mock-Objekte. Setzen Sie dafür ein Mocking-Framework ein.

(A2.3) *Hurace.RaceControl:* Entwickeln Sie die Benutzeroberfläche von *Hurace.RaceControl* mit Hilfe der WPF. Alternativ können diese Systemkomponenten auch als Windows-Universal-Plattform-App realisiert werden. Koppeln Sie die Benutzeroberfläche mit der Geschäftslogik. Versuchen Sie möglichst weitreichend XAML und andere Konzepte der WPF, wie Ressourcen und Datenbindung, einzusetzen. Sorgen Sie durch Anwendung des Model-View-ViewModel-Musters für eine konsequente Trennung der Logik der Benutzeroberfläche von der Geschäftslogik. Greifen Sie, soweit wie möglich, auf bestehende Komponenten zurück.

(A2.4) *Hurace.Simulator:* Da diese Komponente kein Bestandteil des Produktivsystems ist, kann die Benutzeroberfläche schlicht gestaltet werden. Integrieren Sie *Hurace.Simulator* in *Hurace.RaceControl* und kapseln Sie diese Komponente sauber, indem Sie sie in ein eigenes Assembly (oder ein NuGet-Paket) auslagern. Um die Unabhängigkeit der Komponenten zu gewährleisten, muss auch das Interface zur Zeitnehmung (`IRaceClock`) in ein eigenes Assembly verpackt werden.

(A2.5) *Flexible Software-Architektur:* Alle Funktionen, die nicht unmittelbar für die Umsetzung der Benutzeroberflächen benötigt werden, sollten in dieser Komponente implementiert werden.

(A2.6) *Dokumentation:* Führen Sie die notwendigen Ergänzungen in der Dokumentation durch. Erweitern Sie einerseits die Systemdokumentation aus Ausbaustufe 1 und erstellen Sie andererseits eine rudimentäre Benutzerdokumentation. Aus Ihrer Dokumentation soll die Architektur des Gesamtsystems hervorgehen, insbesondere die Struktur der Klassen und Interfaces und die Kopplung der Systemkomponenten sollte übersichtlich dargestellt werden. Hierzu eignen sich besonders gut UML-Klassen- und Paketdiagramme.

Ausbaustufe 3

(Abgabe: 19. 1. 2020, 24:00 Uhr)

(A3.1) *Hurace.Core:* Führen Sie die notwendigen Erweiterungen an *Hurace.Core* durch. Überprüfen Sie nochmals Ihr Design, vor allem hinsichtlich einer sauberen Trennung der verschiedenen Schichten und überarbeiten Sie gegebenenfalls Ihre Implementierung (Refactoring).

(A3.2) *Hurace.Web*: Die Präsentationsschicht ist in Form einer Single-Page Applikation (SPA) mit Angular 8 zu realisieren. Versuchen Sie, in Ihrer Lösung soweit wie möglich zusätzliche Steuerelemente (z. B.*: Bootstrap, SemanticUI, Devextreme, etc.*) zu verwenden. Diese Zusatzbibliotheken bieten diverse Komponenten an, die für die Erstellung des Web-Frontends hilfreich sind.

Legen Sie das Design Ihres Systems so aus, dass der Web-Klient über *Hurace.Api* auf die Geschäftslogik zugreifen kann. Konfigurieren Sie die Anwendung so, dass sie leicht an die Anforderungen eines Kunden angepasst werden kann (z. B. Host und Port des Web-Service). Achten Sie außerdem auch auf eine ausreichende client- und serverseitige Validierung.

Setzen Sie eine geeignete Authentifizierung ein, sodass nur jene Personen Daten modifizieren können, die dazu befugt sind.

Fügen Sie geeignete Funktionalität zur Präsentationsschicht hinzu, sodass Benutzer von der besonderen Leistungsfähigkeit Ihrer Anwendung beeindruckt sind. Die Benutzerschnittstelle soll dabei den aktuellen Stand in der Gestaltung von Web-Anwendungen wiedergeben.

(A3.3) *Hurace.Api:* Diese Systemkomponente ist mit ASP.NET Core MVC als REST-basiertes Web-Service zu realisieren. Die Anforderungen an die zu exportierende Funktionalität des Web-Service ergeben sich aus der Funktionalität von *Hurace.Web*.

(A3.4) *Tests:* Überprüfen Sie, ob auch nach Abschluss aller Entwicklungsarbeiten alle Unit- und Integrationstests erfolgreich ausgeführt werden können.

(A3.5) Dokumentation: Führen Sie die notwendigen Ergänzungen an der Dokumentation durch. Dokumentieren Sie auch den WEA5-Teil ausführlich (Architektur, UML-Diagramme, kommentierte Screenshots, Installationsanleitung sowie Benutzerschnittstellen-Beschreibung) und erstellen Sie dafür ein eigenständiges Dokument. Achten Sie darauf, dass auch der WEA5-Teil adäquat dargestellt wird.