Here is a Scala implementation of the challenge.

```
import java.time.LocalDate
import java.time.temporal.ChronoUnit
import scala.math.Ordering.Implicits.infixOrderingOps

case class Period(from: LocalDate, to: LocalDate) {
  def intersectWith(period: Period): Option[Period] = {
    if (from > period.to) None
    else if (to < period.from) None
    else {
      val f = if (from < period.from) period.from else from
      val t = if (to > period.to) period.to else to
      Some(Period(f, t))
    }
  }

  lazy val days = ChronoUnit.DAYS.between(from, to)
}

object Period {
  def of(from: String, to: String): Period = {
    Period(LocalDate.parse(from), LocalDate.parse(to))
  }
}

case class Residence(from: String, to: String, address: String) {
  val period = Period.of(from, to)
}

case class Application(applicantResidences: List[Residence],
                       spouseResidences: List[Residence],
                       marriedPeriods: List[Period])

case class Outcome(eligible: Boolean, eligibleDays: Long, marriedLivedTogether: List[Period], yearsRequired:Int,
since:LocalDate)
```

```
class Decision( yearsRequired: Int, withinLastYears: Int) {

  val cutoffPeriod = {
    val today = LocalDate.now()
    Period( today.minusYears(withinLastYears), today)
  }

  def eligibility(app: Application): Outcome = {
    val marriedLivedTogether = livedTogetherWhileMarried(app)
    val days = marriedLivedTogether.map(period => period.days).sum;
    val eligible = yearsRequired * 365 < days
    Outcome(eligible, days, marriedLivedTogether, yearsRequired, cutoffPeriod.from)
  }

  private def livedTogetherWhileMarried(app: Application): List[Period] = {
    for (appRes <- app.applicantResidences;
         spouseRes <- app.spouseResidences;
         married <- app.marriedPeriods;
         livedTogether <- appRes.period.intersectWith(spouseRes.period) if appRes.address == spouseRes.address;
         livedTogetherWhileMarried <- livedTogether.intersectWith(married);
         eligiblePeriod <- livedTogetherWhileMarried.intersectWith(cutoffPeriod)
         )
    yield eligiblePeriod
  }
}

object Test extends App {
  val applicantResidences = List(
    Residence("2010-01-01", "2015-12-31", "123 Main St, Anytown, USA"),
    Residence("2016-01-01", "2020-12-31", "456 Oak St, Anytown, USA"),
    Residence("2021-01-01", "2023-03-04", "789 Elm St, Anytown, USA")
  )

  val spouseResidences = List(
    Residence("2010-01-01", "2015-12-31", "123 Main St, Anytown, USA"),
    Residence("2016-01-01", "2020-12-31", "120 Maple St, Anytown, USA"),
    Residence("2021-01-01", "2023-03-04", "789 Elm St, Anytown, USA"),
  )
```

```
  val married = List(
    Period.of("2010-01-01", "2015-12-31"),
    Period.of("2021-01-01", "2023-03-04")
  )

  val application = Application(applicantResidences, spouseResidences, married)

  val outcome = new Decision(yearsRequired=7, withinLastYears=10).eligibility(application)

  val time = s"${outcome.eligibleDays / 365} years ${outcome.eligibleDays % 365} days"
  val result = if (outcome.eligible) "" else "not"

  println("Applicant is " + result + " eligible for the permit")
  println("Total time lived together since " + outcome.since + " is " +  time )
  println("Eligible periods are " + outcome.marriedLivedTogether )
}
```

The code can be tested with online Scala REPL. Copy/paste the code into https://scastie.scala-lang.org/
Here are the results:

```
Applicant is not eligible for the permit
Total time lived together since 2013-04-06 is 4 years 331 days
Eligible periods are List(Period(2013-04-06,2015-12-31), Period(2021-01-01,2023-03-04))
```